Горан Шимић

Технологије и системи вештачке интелигенције

УЏБЕНИК

- 1. издање -

АКАДЕМИЈА ТЕХНИЧКО-УМЕТНИЧКИХ СТРУКОВНИХ СТУДИЈА БЕОГРАД

ОДСЕК ВИСОКА ШКОЛА ЗА ЕЛЕКТРОТЕХНИКУ И РАЧУНАРСТВО

2025

Аутор: др Горан Шимић

Наслов: Технологије и системи вештачке интелигенције

Прво издање

Издавач: Академија техничко-уметничких струковних студија Београд, Одсек Висока школа електротехнике и рачунарства, Војводе Степе 283, Београд;

За издавача: др Ана Савић, председник Академије

Рецензенти: др Габријела Димић и др Зоран Шеварац

Београд 2025

ИСБН: 978-86-6090-186-8

Штампа: BIROGRAF COMP D.O.O. BEOGRAD, Атанасија Пуље 22, Београд 11080

Тираж: 50 примерака

CIP - Каталогизација у публикацији Народна библиотека Србије, Београд

004.8(075.8)(076)

ШИМИЋ, Горан, 1967-
Технологије и системи вештачке интелигенције : уџбеник / Горан Шимић.
- 1. изд. - Београд : Академија техничко-уметничких струковних студија, 2025
(Beograd : Birograf Comp). - 242 стр. : илустр. ; 26 cm

Tiraž 50. - На насл. стр.: Одсек Висока школа електротехнике и рачунарства. -
Библиографија: стр. 237-239. - Регистар.

ISBN 978-86-6090-186-8

а) Вештачка интелигенција

COBISS.SR-ID 182283273

Предговор

Уџбеник „Технологије и системи вештачке интелигенције“ настао је као плод вишедеценијског искуства аутора у бављењу вештачком интелигенцијом, како кроз предавања, тако и кроз истраживачки рад. Циљ израде и издавања уџбеника је обезбеђивање основне литературе за предмете везане за вештачку интелигенцију, а који се држе на студијским програмима Одсека Високе школе електротехнике и рачунарства, Академије техничких и уметничких струковних студија у Београду (у тексту ВИШЕР) и то:

1. Интелигентни системи и технологије– предмет који се држи на студијским програмима Нове рачунарске технологије и Рачунарске технологије,
2. Вештачка интелигенција – предмет који се држи на студијском програму Рачунарске технологије,
3. Вештачке неуронске мреже – предмет који се држи на студијском програму Информациони системи и
4. Меко рачунарство – предмет који се држи на студијском програму Рачунарске технологије.

Осим у реализацији ових предмета, уџбеник се може користити и у реализацији појединих наставних јединица на другим предметима и другим студијским програмима, а које се баве вештачком интелигенцијом.

У првом поглављу дат је увод о вештачкој интелигенцији и машинском учењу са кратким освртом на теоријске основе и практичне примене и значају који данас имају системи вештачке интелигенције у различитим областима људских делатности. Наведено поглавље објашњава суштинске разлике између софтверских производа који су без примене вештачке интелигенције (у даљем тексту ВИ) и оних у којима се ВИ примењује. Акценат на самом почетку уџбеника је и на моралној одговорности у коришћењу ВИ, на њеним предностима и недостацима. Поред наведеног, поглавље садржи и уводна разматрања о машинском учењу и меком рачунарству.

Друго поглавље се бави формалним логикама – исказном (пропозиционом) логиком и логиком првог реда (предикатском), као и значајем ове две логике у формирању логичких формула, које се даље користе за

формализацију система правила и/или ограничења које представљају основ за развој експертских система.

Треће поглавље објашњава експертске системе, који представљају полазни основ ВИ и прву примену ВИ у пракси. Са друге стране, експертски системи су још увек врло актуелни и представљају незаобилазне компоненте система за подршку у одлучивању, у дијагностичким системима и системима пословне интелигенције, где омогућавају брзу и адекватну реакцију на промене у пословном окружењу.

Четврто поглавље објашњава расплинуту (*fuzzy*) логику, њен значај у односу на дискретну логику, и примену расплинуте логике у пракси. Наведено се пре свега односи на примену расплинуте логике у системима аутоматске регулације, као што су регулација пакетске брзине, температуре, притиска, или пак усклађивање процеса који имају асинхрону природу, а притом су узајамно зависни. Ово поглавље обухвата и напреднију тематику, односно примену расплинуте логике у експертским системима како би се омогућио закључивање над не егзактним подацима.

Пето поглавље објашњава основне принципе генетског (еволутивног) алгоритма, специфичности проблема на којима је овај алгоритам примењив, као и начине имплементације оптимизационих решења коришћењем овог алгоритма у пракси.

Шесто поглавље се бави системима ВИ на бази дистрибуције условних вероватноћа. Најпре су представљени су примери коришћења Бајесове теореме у проблемима класификације. Поглавље даље обрађује коришћење Бајесових (пробабилистичких) мрежа и њихову примену у комплексним системима у којима постоји велики број узајамно зависних варијабли.

Седмо поглавље се бави припремом података за коришћење у системима ВИ. Нагласак је на спрези проблема који се решава, технологије ВИ која ће се користити за решавање тог проблема и природе података на основу којих систем ВИ треба да се (машинским учењем) обучи, а затим користи у решавањима проблема за који је дизајниран.

Осмо поглавље објашњава груписање и дељење података, односно њихово кластеровање и класификацију. У системима ВИ некада су ове активности

саме по себи циљ, али исто тако могу да представљају и само једну међуфазу, у комплекснијим процесима обучавања и експлоатације.

Девето поглавље обрађује вештачке неуронске мреже, са нагласком на принципе рада и принципе обучавања различитих врста неуронских мрежа. Ово поглавље приказује детаље примене вештачких неуронских мрежа у решавању различитих проблема класификације. Поред тога, поглавље даје одговоре на питања када и како применити одговарајућу врсту вештачке неуронске мреже у односу на специфичност проблема који се решава.

САДРЖАЈ

1. Кратак увод у вештачку интелигенцију

Вештачка интелигенција је рачунарска дисциплина чији је задатак да створи
рачунарске програме који могу да резонују на начин сличан људском резоновању. То је дисциплина која изучава развој програма који показују интелигенцију налик људској.

Вештачка интелигенција је од апстрактног појма до конкретних примена еволуирала захваљујући великом технолошком напретку крајем прошлог и почетком 21. века , као и богатим теоријским истраживањима која датирају од античког доба и појаве филозофије, логике и развојем математике кроз читаву савремену људску историју.

У савременом друштву, вештачка интелигенција је постала део свакодневног живота захваљујући небројеним применама у различитим областима људског живљења и стваралаштва. Некада само привилегија оних који се баве науком, данас је присутна свугде – у фабрикама, на улици, у саобраћају, у болницама, у паметним уређајима који се налазе у домаћинствима, па и у џеповима (паметни телефони), служи нам као помоћу у обављању свакодневних послова, у решавању проблема, али и у сврху разоноде и релаксације.

1.1. Области примене вештачке интелигенције

Вештачка интелигенција има многобројне области примене. Иако не једине, најзначајније од њих су претраживање и представљање знања, решавање проблема, подршка у одлучивању (експертски системи), аутоматско доказивање теорема, роботика, препознавање и разумевање говора, препознавање и разумевање слика, рачунарске игре, планирање, распоређивање, интелигентно подучавање, машинско учење, процесирање природног језика, слика и видео материјала, меко рачунарство и интеракција човек – рачунар.

1.1.2. Машинско учење

Машинско учење је област вештачке интелигенције фокусирана на алгоритме и (мрежне) моделе које могу да *уче* из података, препознају обрасце и дају решења, или доносе одлуке без потребе за експлицитним

програмирањем. Машинско учење се користи у задацима груписања, класификовања и предикције података, као и за генерисање различитих врста садржаја (на пример, текстуалних, графичких, аудио и видео садржаја). Квалитет реализације задатака машинског учења директно зависи од величине и квалитета скупа података за обучавање, као и избора алгоритама и модела за решавање специфичног проблема. Задатак инжењера знања је да одабере одговарајући модел према задатку и доступним подацима за обучавање. Поред тога, инжењер знања има задатак и да припреми податке за обучавање, како би се обезбедила неопходна тачност модела у току коришћења.

1.1.3. Меко рачунарство

Меко рачунарство представља област вештачке интелигенције која је фокусирана на проналажење апроксимативних (приближних) решења рачунарских проблема, који не могу да се реше конвенционалним рачунарским методама. Поред тога меко рачунарство је усмерено на истраживање и развој могућности система вештачке интелигенције да самостално уче и да се адаптирају. Захваљујући томе, меко рачунарство има хеуристичку природу, што значи да је орјентисано ка трагању за новим знањем на основу претходног знања (искуства) и информација садржаних у подацима.

За разлику од конвенционалних рачунарских система заснованих на дискретним и комплетним подацима, меко рачунарство омогућава коришћење података који могу да буду нејасни, двосмислени и непотпуни. Из тог разлога, меко рачунарство је засновано на концептима као што су: расплинута (енгл. *fuzzy*) логика, резоновање на бази теорије вероватноће, еволутивни алгоритми и вештачке неуронске мреже.

2. Представљање знања – логика

У општом смислу, логика се бави методама и вештинама правилног мишљења. Логика поред тога садржи законе чијом применом сваки човек може да се научи правилном мишљењу. Садржаји логике су примењиви у свакој науци, учењу и уметности. Незаобилазна је и примена логике у развоју система вештачке интелигенције.

Ова секција се бави формалном математичком (симболичком) логиком која има свој језик. Као и сваки други, језик математичке логике за представљање знања је дефинисан са два аспекта: синтаксом и семантиком. Ова два аспекта омогућавају представљање знања, представљање правила закључивања, затим представљање веза и односа између чињеница и/или концепата и на крају, извођење нових закључака на основу постојећих информација.

У циљу формализације исказа и добијања логичких формула, математичка логика је увела симболе, којима су представљени елементарни (недељиви) искази који се још називају и атомима. На пример један такав скуп симбола може бити V={ P, Q, ...}.

Поред симбола, математичка логика уводи и операторе. То су унарни оператор негације ¬ (NE) и бинарни оператори дисјункције ∨ (ILI), конјункције ∧ (I), импликације → и еквиваленције ↔. Поред наведеног, користе се и заграде које служе за груписање у случајевима конструкције сложенијих логичких формула.

Код математичке логике постоје две могуће вредности логичких формула, представљене логичким константама: тачно и нетачно, при чему за њих постоје посебни симболи {⊤, ⊥}, али се могу користити и друге ознаке, на пример {T,F}, {1,0}, а у програмским језицима као {*true, false*}.

Логичке формуле су скалабилне. То значи да формуле могу бити елементарни искази (атоми) и њихове негације, али и сложене конструкције које укључују симболе и логичке операторе, као и груписања формула ради дефинисања комплексније структуре.

На пример ако је R атом, онда је R формула, али је и ¬R такође формула. Даље, логичке формуле могу градити атоми, или друге логичке формуле

помоћу бинарних логичких оператора. На пример, ако су P и Q атоми или формуле, онда су формуле и $P \vee Q$, $P \wedge Q$, $P \rightarrow Q$ и $P \leftrightarrow Q$. Најкомплексније формуле исказне логике имају поред свега поменутог још и груписања коришћењем заграда. На пример, ако су P, Q и R атоми, или формуле, онда је формула и $(((P \vee R) \wedge (\neg P \vee Q)) \rightarrow (R \vee Q))$.

Логичке формуле могу бити истините или неистините. Истинитост логичке формуле се доказује помоћу таблица истинитости за све њене интерпретације. На пример, ако треба доказати истинитост логичке формуле $\neg(P \wedge Q) \rightarrow \neg P \vee \neg Q$, онда се формира таблица истинитости за све вредности P и Q, било да су P и Q атоми, или формуле (Табела 2.1).

P	Q	$\neg (P \wedge Q)$	$\neg P \vee \neg Q$	$\neg (P \wedge Q) \rightarrow \neg P \vee \neg Q$
$\bot$	$\bot$	$\top$	$\top$	$\top$
$\bot$	$\top$	$\top$	$\top$	$\top$
$\top$	$\bot$	$\top$	$\top$	$\top$
$\top$	$\top$	$\bot$	$\bot$	$\top$

Табела 2.1: Таблица истинитости за логичку формулу $\neg (P \wedge Q) \rightarrow \neg P \vee \neg Q$

На основу таблице истинитости може се закључити да је логичка формула истинита. Истините формуле се још називају и таутологије. Обрнуто, логичке формуле које су неистините за све интерпретације се називају контраиндикацијама.

За развој система вештачке интелигенције посебан значај имају исказана и предикатска логика.

2.1. Исказна логика и примена у интелигентним системима

Исказна логика се користи за представљање исказа. Искази се још називају пропозицијама, а користе се још и термини: чињенице, тврдње и судови. Искази могу да буду прости (једноставни) и сложени. Простим исказима се представљају атоми, док се сложеним исказима представљају сложеније структуре, тј. логичке формуле које укључују коришћење бинарних логичких оператора и груписање. Претпоставимо да постоје четири исказа писана природним језиком (Табела 2.2).

1	*мотор не може да се покрене;*
2	*напон акумулатора је 12V;*
3	*ако је напон акумулатора 12V и мотор не може да се покрене* *онда је дефект у сопојном путу;*

4	*ако је дефект у сиојном иуту онда је дефект локализован;*

Табела 2.2: четири исказа писана природним језиком

Искази 1 и 2 представљају једноставне чињенице. Искази 3 и 4 су сложени и имају структуру правила за закључивање (*ако је...онда је...*). Посебна је сложеност исказа 3 код ког се у условном (*ако...*) делу појављују два исказа (искази 1 и 2) који су повезани везником *и*. То значи да је исказ 3 тачан само уколико су тачна оба исказа у услову (оба исказа – 1 и 2). У том случају је тачан и закључак (*онда...* део) исказа 3, а то је нова чињеница која гласи: *дефект у сиојном иуту*.

Последица појављивања нове чињенице – *дефект у сиојном иуту* је да и условни део исказа 4 постаје тачан. Како је и исказ 4 сложен, тј. има структуру правила за закључивање, онда тачност његовог услова указује и на тачност његовог закључка, а то је нова чињеница да је *дефект локализован*.

Конструкција правила (у овом случају презентовани као сложени искази) има пресудни значај за дизајн базе знања код експертских система (Поглавље 4). Правила имају униформну структуру, која се може описати као следећи израз (Израз 2.1):

ако је {*услов / иремиса* } ***онда је*** {закључак / последица } И.2.1

У левом делу правила (*ако...* део) налази се логичка формула која се зове услов правила, или премиса правила. У десном делу правила (*онда...* део) налази се логичка формула која се зове закључак правила, или последица правила.

2.1.1. Закључивање у исказној логици

У основи, закључивање (испитивања истинитости тврдњи) у исказној логици могуће је извести на два начина: методом потврђивања и методом оповргавања. Општи облик методе потврђивања (lat. *Modus Ponens*), представљен је изразом (Израз 2.2). Он се може интерпретирати да ако је тачно да P → Q и ако је P тачно, онда је и Q тачно.

$$((P \rightarrow Q) \wedge P) \rightarrow Q \qquad \text{И.2.2}$$

На пример, ако симбол P представља тврдњу "*иада киша*", а симбол Q представља тврдњу "*треба обући кабаницу*", онда израз P → Q може да се интерпретира као усвојено правило понашања: *ако иада киша онда треба*

обући кабаницу. У случају да је тврдња "*пада киша*" тачна, методом потврђивања (применом правила) се долази до закључка (имплицира) да је тврдња "*треба обући кабаницу*" тачна.

Општи облик методе оповргавања (lat. *Modus Tollens*) представљен је следећим изразом (Израз 2.3). Он се може интерпретирати да ако је тачно да P → Q у случају да је Q нетачно, онда је и P нетачно.

$$((P \rightarrow Q) \wedge \neg Q) \rightarrow \neg P \qquad \text{И.2.3}$$

На пример, ако симбол P представља тврдњу "*пада киша*", а симбол Q представља тврдњу "*улице су мокре*", онда израз P → Q може да се интерпретира као усвојено правило понашања: ако *пада киша* онда *улице су мокре*. У случају да је тврдња "*улице су мокре*" нетачна, методом оповргавања се долази до закључка (имплицира) да је тврдња "*пада киша*" такође нетачна.

2.1.2. Превођење исказа у логичке формуле

Ако је потребно да се искази преведу из природног језика у формуле исказне логике, онда најпре треба препознати атоме (просте исказе), а затим дефинисати правила коришћењем тих атома. У примеру датом у претходној табели (Табела 2.2) могу се препознати 4 атома. Формализација ових атома своди се на означавање симболима (Табела 2.3):

Симбол	Исказ
P	*"напон акумулатора је 12V"*
Q	*"мотор не може да се покрене"*
R	*"дефект у спојном путу"*
S	*"дефект локализован"*

Табела 2.3: формализација простих исказа - атома

У следећем кораку се преводе сложени искази. Другим речима, следи дефинисање правила, којих у конкретном примеру има два, дата као искази 3 и 4 (Табела 2.2). У условном делу правила – сложеног исказа 3 су два атома (*P* и *Q*) повезана логичким везником ∧, док је у закључку правила атом *R*, тако да се правило *P1* може представити изразом (Израз 2.4):

P1 $P \wedge Q \rightarrow R$ И.2.4

Исказ 4 (Табела 2.2) се формализује такође као правило. Означено као *P2*, у условном делу садржи један атом - *R*, а у закључку садржи други атом - *S*, тако да се правило може представити изразом (Израз 2.5):

$$P2 \qquad R \rightarrow S \qquad \text{И.2.5}$$

База знања механизма за закључивање у овом једноставном примеру се састоји од 2 правила (*P1* и *P2*). Ова два правила ће се извршити само у случају да се у радној меморији механизма за закључивање појаве чињенице (атоми) *P* и *Q*. Прво ће се извршити правило *P1*. Његовим извршењем, у радној меморији механизма за закључивање ће се појавити нова чињеница (атом) *R*. Овом појавом ће се извршити правило *P2*, чијим извршењем се појављује још једна нова чињеница у радној меморији механизма за закључивање, а то је атом *S*.

Горе представљен пример је детаљ из система за дијагностику кварова на моторном возилу, прилагођен ради објашњења како се врши процес закључивања на бази исказне логике. Искази или тврдње којима се додељује једна и само једна логичка вредност (тачно или нетачно) називају се искази, пропозиције или судови. На основу датих примера формализације може се закључити да исказна логика омогућава представљање знања у вештачкој интелигенцији, било да се ради о једноставним исказима (атомима), или сложеним исказима, као што су правила.

2.1.3. Потреба за аутоматизацијом

Посебан проблем представља ако постоји велики број исказа (стотине или хиљаде). У том случају постоји потреба да се закључивање аутоматизује да би могло да се извршава на рачунару. Један од основних правила ове аутоматизације је да се одреде почетно и циљно стање процеса закључивања.

На примеру из претходне секције (2.1.1) је имплицитно јасно да је почетно стање, или предуслов процеса закључивања појава две чињенице у радној меморији (Табела 2.3: чињенице *P* и *Q*). С друге стране, појава чињенице "*дефект локализован*" (Табела 2.3: атом *S*) представља имплицитни циљ резоновања. У посматраном механизму за закључивање, појавом чињенице *S* процес закључивања је завршен, обзиром да нема других

правила за извршење (изузев *P1* и *P2* која су извршена), нити нових чињеница у радној меморији.

Може се претпоставити да дати механизам за закључивање треба да се унапреди тако да даје прецизнију дијагнозу квара и инструкције за његово отклањање. Да би се наведено омогућило потребно је дефинисање нових чињеница и нових правила. На пример, уколико су у разговору са аутомеханичарем прикупљене нове информације на основу којих се добијају сложени искази у природном језику (Табела 2.4).

1	***ако је*** *"дефект локализован"* ***и*** *"клеме оксидирале"* ***онда је*** *"провера извршена"* ***и*** *порука = „скинути и очистити клеме металном четком, унутрашњи део брусним папиром гранулације 300, вратити, дотегнути и подмазати товарном машћу";*
2	***ако је*** *"дефект локализован"* ***и*** *"кабл прекинут"* ***онда је*** *"провера извршена"* ***и*** *порука = „замени неисправан кабл";*
3	***ако је*** *"дефект локализован"* ***и*** *"осигурач неисправан"* ***онда је*** *"провера извршена"* ***и*** *порука = „замени неисправан осигурач";*
4	***ако је*** *"провера извршена"* ***онда је*** *"циљ остварен"*

Табела 2.4: сложени искази за проширење базе знања дијагностичког система

Најпре треба извући атомске исказе, доделити им симболе и ажурирати табелу атома (Табела 2.5). Постојећи атоми су представљени сивом, док су нови атом представљени црним исписом.

Симбол	Исказ
P	*"напон акумулатора је 12V"*
Q	*"мотор не може да се покрене"*
R	*"дефект у струјном путу"*
S	*"дефект локализован"*
K	*"клеме оксидирале"*
L	*"кабл прекинут"*
M	*"осигурач неисправан"*
N	*"провера извршена"*
O	*"циљ остварен"*

Табела 2.5: проширена таблица атома и додељених симбола

Затим се на основу нових сложених исказа (Табела 2.4), заједно са претходно добијеним атомима (Табела 2.5), дефинишу правила, којима се затим допуњује постојећа база знања (Табела 2.6). Постојећа правила представљена су сивом (*P1* и *P2*), док су нова правила (од *P3* до *P6*) представљена црним исписом.

Ознака правила	Формула која дефинише правило
P1	$P \wedge Q \rightarrow R$
P2	$R \rightarrow S$
P3	$S \wedge K \rightarrow N \wedge$ *порука*
P4	$S \wedge L \rightarrow N \wedge$ *порука*
P5	$S \wedge M \rightarrow N \wedge$ *порука*
P6	$N \rightarrow O$

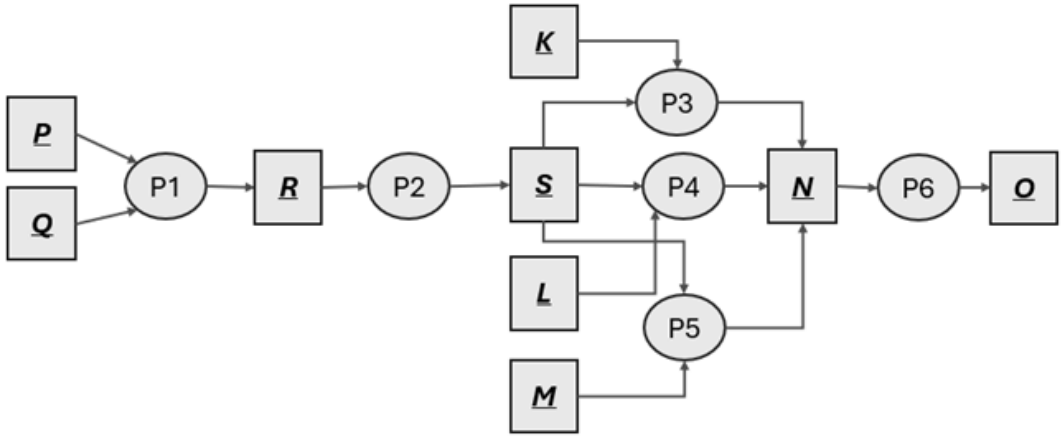
Табела 2.6: Изглед ажуриране базе знања додавањем нових правила

Може се уочити да су, за разлику од правила *P1* и *P2,* код нових дефинисаних правила (од *P3* до *P6*) у закључку две нове чињенице у конјункцији: једна је означена као атомски исказ *N*, а друга је именована као *порука*. Оваквим записом се ставља до знања инжењеру знања да је *порука* намењена искључиво кориснику система и да нема никаквог утицаја на наставак процеса закључивања. С друге стране, чињеница (атом) *N* је неопходна за завршетак процеса закључивања. Она представља услов извршења правила *P6*, које је на неки начин циљно, пошто генерише циљно стање – чињеницу представљену атомом *O* (*"циљ остварен"*).

2.1.4. Потреба за дизајном

Интересантно је да је почетно стање процеса закључивања исто као и пре модификације базе знања. Оно што је промењено је циљно стање, као и чињенице које представљају предуслов доласка до циљног стања. Да би нове чињенице могле да се обраде, уводе се нова правила. Практично, процес закључивања је продужен променама у бази знања.

Други важан закључак је да је од шест дефинисаних правила само једно условљено искључиво чињеницама у радној меморији. То је правило *P1*. Остала правила су зависна не само од чињеница у радној меморији већ и од осталих правила: *P2* је зависно од *P1*, правила *P3*, *P4* и *P5* су зависна од правила *P2* док је правило *P6* зависно од правила *P3*, *P4* и *P5*. Друкчије речено, чињенице које се траже као услов извршења зависних правила генеришу се извршавањем неких других правила. Модификована база знања има комплекснију структуру, а за правила због њихове узајамне зависности може се рећи да су уланчана и могу се представити графички (Слика 2.1).



Слика 2.1: шематски приказ закључивања на бази правила P1-P6

Правила су приказана елипсама, чињенице су приказане квадратима, док усмерене линије приказују смер закључивања и зависности. На пример *P1* зависи од чињеница *P* и *Q*, *P2* зависи од чињенице *R* која настаје искључиво извршењем правила *P1*. Даље, правила *P3*, *P4* и *P5* поред чињеница *K,L* и *M* зависе и од чињенице *S* која настаје искључиво извршењем правила *P2*. На крају, правило *P6* зависи од правила *P3*, *P4* и *P5* обзиром да генеришу чињеницу *N* неопходну за извршење правила *P6*. Коначни исход закључивања је чињеница *O*. Захваљујући замени исказа и правила симболима, добијен је прегледан приказ не само статичке структуре механизма за закључивање, већ и динамике одвијања процеса закључивања.

2.1.5. Закључак о исказној логици у интелигентним системима

Механизми за закључивање на бази исказне логике су једноставног дизајна без обзира на обим чињеница, број и структуру правила које се користе. Чињенице, тј. искази (пропозиције) су најчешће атомски, нису структуирани, имају константне вредности (најчешће *string* типа). Садржај исказа у исказној логици нису битни. Само је битна њихова истинитост – да ли су тачни или не. У представљеном примеру то је још више поједностављено: ако је тражена чињеница у радној меморији механизма за закључивање, онда се узима да је тачна, а у другим случајевима је нетачна.

У случају да су прикупљени сви потребни искази, дизајн механизма за закључивање на бази исказне логике у основи има две фазе. У првој фази

се издвајају прости искази – атомске формуле. У другој фази се од сложених исказа формирају правила за закључивање. У обе фазе се користи формализација симболима и операторима језика математичке логике. Ако постоји велики број атома (чињеница које се користе) и узајамно зависних правила, а у циљу постизања јасности и прегледности структуре, онда се препоручује графичко представљање механизма за закључивање.

2.1.6. Питања и задаци

1. Дати пример простог и сложеног исказа.
2. Како се формирају правила од сложеног исказа?
3. Попунити табеле истинитости за све операторе исказне логике.
4. Проверити да ли је логичка формула (((A ∨ B) ∧ ¬B) → C) таутологија.
5. Проверити да ли је логичка формула ((F ∧ (F → G)) → G) таутологија.
6. Која је разлика између таутологије и контраиндикације?
7. Дати пример за закључивање методом потврђивања (*modus ponens*)
8. Дати пример за закључивање методом оповргавања (*modus tollens*)
9. Проверити да ли је логичка формула ((F ∧ (F → G)) ∧¬G) контраиндикација.
10. На доњој слици (Слика 2.2) је део описа неисправности веш машине који се односи на случај да машина не преузима воду.

Mašina se ne puni vodom.	Slavina za vodu je zatvorena.	♦ Otvorite slavinu.
	Crevo za dovod vode je savijeno.	♦ Ispravite crevo.
	Filter za dovod vode se zapušio.	♦ Očistite filter.
	Možda su vrata mašine otvorena.	♦ Zatvorite vrata.
	Konekcije za vodu su možda pogrešne ili nema vode (Kad nema vode, LED za pranje ili ispiranje trepće).	♦ Proverite konekcije za vodu. Ako nema vode, pritisnite dugme Start/Pauza kad se pojavi voda kako biste nastavili program od pauze.

Слика 2.2: списак неисправности веш машине за задатак 10

Задатак је да се на основу примера у уџбенику (поглавља 2.1.1. – 2.1.3.) изврши дизајн механизма за закључивање на бази исказне логике.

2.2. Предикатска логика и примена у интелигентним системима

Предикатска логика (или логика првог реда) представља проширење исказне логике како би у механизмима за закључивање, поред једноставних исказа, могли да се користе сложени (структурни) типови

података као што су класе (типови објеката) и колекције (низови, листе, скупови, мапе и сл.). Предикатска логика је основа за већину репрезентативних схема (базе знања) у системима вештачке интелигенције.

2.2.1. Синтакса предикатске логике

Скуп елемената над којим се изводи закључивање зове се домен. Елементи за закључивање могу да буду

1. Константе, на пример: *A, X_1, "Мирко", "Марија"*
2. Промењиве, на пример: *a, x, osoba, proizvod*
3. Функције које пресликавају елементе домена у тај исти скуп: *racun, popust*
4. Предикати су специфичне функције које пресликавају елементе домена у једну од логичких вредности (*тачно/нетачно, или T/⊥ или 0/1, или true/false*). Примери предиката су *ozenjen, imaJos, zaposlen*

Синтакса исказне логике обухвата све описане логичке операторе (описане у секцијама 2 и 2.1). Поред наведених елемената, за разлику од исказне логике, предикатска логика уводи квантификаторе – универзални (∀) и егзистенцијални (∃). Тиме се добија већа изражајност формализама написаним језиком предикатске логике. На пример, ако треба да се опише тврдња „*сви воле сладолед*" коришћењем квантификатора то би изгледало као у следећем изразу (Израз 2.6). Ова логичка формула се може интерпретирати да је за свако *x* примењив предикат *voli*. За разлику од природног језика када је предикат најчешће након субјекта, у логичкој формули предикат задржава функционалну нотацију, код које је предикат назив функције, први параметар је субјекат (x), а други параметар је објекат ("sladoled").

$$\forall x \, voli(x, \text{“sladoled”}) \qquad \text{И. 2.6}$$

Језиком предикатске логике интерпретација тврдње из претходног израза може лако да се прошири (Израз 2.7). Тврдња да „*свако воли сладолед*" проширена је еквиваленцијом са тврдњом да "*не постоји ко не воли сладолед*". У другој тврдњи у формализацији је коришћен егзистенцијални квантификатор и двострука негација: негација квантификатора и негација предиката.

$$\forall x\ voli(x, \text{“sladoled”}) \leftrightarrow \neg\exists x\ \neg voli(x, \text{“sladoled“}) \qquad \text{И. 2.7}$$

Ако би предикат *voli(x, “sladoled”)* био представљен симболом *S*, добија се примена Де Морганове теореме у предикатској логици (Израз 2.8).

$$\forall x\ S \leftrightarrow \neg\exists x\ \neg S \qquad \text{И. 2.8}$$

У претходним примерима промењива *x* је квантификована и назива се још *везана* промењива (варијабла). Аналогно претходној тврдњи, промењиве које нису квантификоване називају се *слободним*. Иначе, сва правила за грађење формула у исказној логици важе за предикатску логику. Поред тога су формуле и оне које садрже везане варијабле (квантификаторе). Важна је напомена да квантификатори имају приоритет над свим осталим логичким операторима.

2.2.2. Превођење тврдњи у формуле предикатске логике

У превођењу тврдњи у из природног језика у формуле исказне логике препоручује се интензивно коришћење објеката, предиката и квантификатора. На пример, ако је потребно да се формализује тврдња:

„студенти су особе које студирају на факултетима“

Најпре је потребно препознати објекте, а то су: *студент, особа, факултет*. У следећем кораку треба препознати предикате. Пошто је тврдњом дато да су *студент* и *особа* исти објекат, за њих се дефинише једна промењива (*x*), а , а за објекат *факултет* дефинишемо другу промењиву (*y*). У тврдњи постоји више предиката. Најочигледнији је предикат *студира(jу)*. Како у предикатској логици предикати пресликавају елементе домена (студент, особа) у логичке вредности (тачно/нетачно), због испитивања истинитости треба дефинисати и предикате за објекте *студент* и *особа*. Последњи корак је грађење логичке формуле коришћењем доступних оператора предикатске логике (Израз 2.9).

$$\forall x\ (student(x) \leftrightarrow osoba(x) \wedge (\exists y\ (fakultet(y) \wedge studira(x,y))) \qquad \text{И. 2.9}$$

У датој интерпретацији тврдње коришћена је еквиваленција. Са леве стране се испитује тачност да ли је *x* студент, а с друге стране да ли је *x* особа која студира на факултету *y*. За квантификацију промењиве *x* коришћен је универзални квантификатор (∀), а за квантификацију промењиве *y*

коришћен је егзистенцијални квантификатор (∃). Тиме се изражава да је студент свака особа за коју постоји бар један факултет који студира.

Следећа претпоставка је да је потребно проширити знање о студенту са следећом тврдњом:

„апсолвенти су студенти који су положили све предметне испите“

Да бисмо избегли грешке у формализацији услед великог броја објеката, промењивих, логичких оператора и предиката, потребно је да се примени систематичнији приступ. Најпре се прави табела промењивих и предиката/објеката (Табела 2.7), у коју се уносе додатни објекти / предикати. За задату тврдњу то су *апсолвент* и *предмет (скраћено од предметни испит)*.

промењива	објекат
x	*student*
x	*osoba*
y	*fakultet*
x	*apsolvent*
z	*predmet*

Табела 2.7: табела промењивих у објеката/предиката

Претходна тврдња садржи један експлицитни предикат – *положили*. Обзиром да је апсолвент *сваки* студент који је положио *све* предмете, очигледно је да ће се у формализацији користити само универзални квантификатор. Следи конструкција формуле за задату тврдњу (Израз 2.10).

$$\forall x\,(\text{apsolvent}(x) \leftrightarrow \text{student}(x) \wedge (\forall z\,(\text{predmet}(z) \wedge \text{polozio}(x,z))) \qquad \text{И. 2.10}$$

За разлику од исказне логике, база знања предикатске логике, поред правила може да садржи и еквиваленције као што су дате у примерима (Израз 2.9 и 2.10). На пример, оне омогућавају закључивање да ли је особа у статусу студента, па чак и да ли има статус апсолвента испитујући истинитост одговарајућих чињеница везане за одређену особу.

2.2.3. Универзална специјализација

У претходним примерима уочљиво је интензивно коришћење промењивих у дизајну формула предикатске логике. Промењиве су искоришћене како би се провериле додељене им вредности да ли су елементи домена над којим се закључује. Конкретно, то је провера промењиве *x* да ли се ради о особи

која студира, даље да ли је промењива *y* факултет, а затим да ли особа *x* студира на факултету *y*. Све су то примери још једног новог својства предикатске логике, а то је *универзална сйецијализација*. Формуле (Израз 2.9 и 2.10) указују да ако је формула истинита за сваки елемент домена закључивања, онда је истинита за било који одређени елемент тог домена. Ова тврдња се може представити изразом. Ако је *x* промењива из домена закључивања, F формула предикатске логике и *B* константа из истог домена закључивања, онда важи следећа импликација (Израз 2.11).

$$\forall x\, F(x) \rightarrow F(B) \qquad \text{И. 2.11}$$

На пример, универзалном специјализацијом могло би се извршити закључивање на основу претходних примера (Израз 2.9 и 2.10) да је *B* студент и апсолвент у случају да је *B* особа која студира и која је положила све предметне испите. Обзиром да број елемената у домену није ограничен (нпр. број особа које су студенти је промењив), потпуна провера формуле (било на истинитост, или оповргавање) није могућа, па се за предикатску логику још каже да није комплетна у погледу одлучивости (закључивања).

2.2.4. Закључак о предикатској логици у интелигентним системима

Предикатска логика је омогућила већу изражајност приликом дизајна базе знања у односу на исказну логику. То је постигнуто увођењем квантификатора, предиката и универзалне специјализације. Овим променама је омогућено резоновање над сложеним структурама података. Један изведени податак може да има различите предикате. Тако неки податак може истовремено да буде и особа, и студент и апсолвент, да студира на одређеном факултету и да полаже предметне испите на том истом факултету.

С друге стране не постоји нити један концепт предикатске логике који је у контрадикцији са концептима исказне логике. Практично, уместо узајамне искључивост, постоји узајамна допуњивост (суплементарност). Искази у исказној логици представљају се константама у предикатској логици. Синтакса исказне логике важи и у предикатској логици. То омогућава скалабилност у дизајну база знања и механизама закључивања. Сложенији захтеви се имплементирају у предикатској, а једноставнији у исказној логици.

2.2.5. Питања и задаци

1. Која је разлика између функција и предиката у предикатској логици?
2. Шта постоји у формулама предикатске логике, а не постоји у формулама исказне логике?
3. Која је разлика између везаних и слободних варијабли у предикатској логици?
4. Направити пример (тврдњу) у предикатској логици друге Де Морганове еквиваленције, која гласи: $\exists x\, S \leftrightarrow \neg\forall x \neg S$ (на основу примера И.6)
5. Да ли је могућа потпуна провера истинитости формуле у предикатској логици?
6. Објасните појам универзалне специјализације.
7. На основу логичких формула И.7 и И.8 дефинишите потпуно нову формулу, која је у складу са тврдњом *„айсолвенти су студенти који су йоложили све йредмеше йредвиђене изабраним студијским йрограмом на факулшешу који сшудирају“*

3. Експертски системи

Експертски (експертни) системи су рачунарски програми којима се емулира решавање проблема на начин на који то чине експерти. Одлике експертних система су многобројне: садрже кодирано знање експерта из неког домена, могу се извршавати тамо где су потребни; модуларни су и могу се лако модификовати (у погледу уграђеног знања); имају могућност закључивања (резоновања), објашњавања; врше хеуристичко, а не исцрпно резоновање; обједињују теорију и праксу вештачке интелигенције.

Експертски системи моделирају знање експерта и његов начин решавања проблема. Дакле, заснивају се на моделу реалног света, тако да је једна од одлика и непрецизност и променљивост током времена. Другим речима, експертски системи се ослањају на праксу, односно садрже практично употребљива знања.

Типични проблеми у којима се користе експертски системи представљени су у следећој табели (Табела 3.1).

Врста проблема	Опис
Управљање	Постизање задатог понашања система
Дизајн	Конфигурисање система уз ограничења
Дијагностика	Закључивање о отказима система
Подучавање	Провера и исправка знања студената
Интерпретација	Тумачење ситуације на основу података
Надзор	Поређење измерених и очекиваних вредности
Планирање	Утврђивање редоследа активности
Предикција	Одређивање могућих последица
Препорука	Предлагање решења у случају отказа система
Селекција	Идентификација најбоље од датих могућности
Симулација	Моделовање интеракције компоненти система

Табела 3.1: Типични проблеми у којима се користе експертни системи

Разлика експертских система у односу на конвенционалне програме су представљене следећом табелом (Табела 3.2). Механизми за закључивање у експертским системима, засновани на исказној и предикатској логици, обрађују више симболичке него нумеричке информације. За разлику од тога, код конвенционалних програма је доминантна обрада нумеричких података. Експертски системи омогућавају хеуристичко резоновање, које као резултат може дати *приближна* решења, док конвенционални програми

процесирају информације на бази алгоритама. Рад конвенционалних програма се заснива на изворним подацима, који се обрађују уграђеним алгоритмима. Насупрот томе, експертски системи поред података користе кодирано експертско знање (тзв. база знања), које механизам за закључивање користи за доношење закључака на основу улазних података. Последично, конвенционални програми дају прецизне резултате, док су резултати код експертских система препоруке и аргументације. На крају, модификације код конвенционалних програма захтевају комплексне интервенције софтверских инжењера задужених за њихов развој и одржавање. Код експертских система, унапређења у раду се постижу модификацијама у бази знања, које су у одговорности инжењера знања.

Конвенционални програми	Експертски системи
Нумерички	Симболички
Алгоритамски	Хеуристички
Користе изворне податке	Уместо података користе кодирано знање
Користе прецизне информације	Користе не-егзактне информације
Коначни резултати су прецизни	Исходи су препоруке и објашњења
Често комплексни за модификације	Модификације над базом знања су лаке

Табела 3.2: поређење конвенционалних програма и експертских система

3.1. Архитектура експертског система

Основни концепти сваког експертског система су база знања, радна меморија и механизам за закључивање (Слика 3.1). База знања садржи специјализовано доменско знање најчешће у виду система правила као што је то објашњено у претходним темама (Секције 2.1 и 2.2). База знања се чува у дугорочној меморији (енгл. *Long Term Memory - LTM*). Друга компонента експертског система је радна меморија која је намењена за привремено чување улазних података, чињеница и закључака у току резоновања (енгл. *Short Term Memory – STM*). Најважнија компонента је механизам за закључивање (енгл. *Reasoning engine*), који на основу чињеница и улазних података користи правила из базе знања, извршава их, а затим закључке смешта у радну меморију. Улазне податке и резултате рада експертски систем размењује са корисником преко корисничког интерфејса.

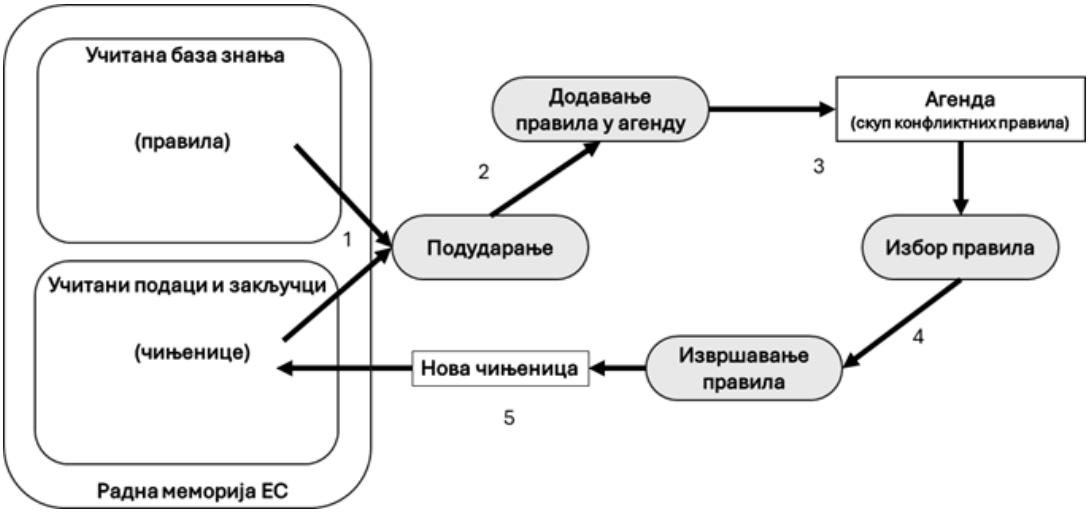
Слика 3.1: Концептуални модел експертског система

Корисник експертског система може да буде физичко лице, али може да буде и други рачунарски програм који користи експертски систем као сервис за подршку у пословним процесима. Експертски системи омогућавају већу агилност пословних система. На пример, свакодневне промене цена, попусти, бонуси, ваучери, поклони и друге акције које трговачки ланци нуде својим потрошачима готово је немогуће имплементирати конвенционалним приступом који подразумева захвате у изворном коду програма. Међутим, ако се за ову врсту функционалности уведе експертски систем било као сервис, или повезана компонента система, онда се све промене измештају у базу знања експертског система. На тај начин је избегнуто интервенисање на изворном коду апликације. Поред тога, уместо да у њихову имплементацију буде укључен тим за развој и одржавање информационог система, бивају укључени само инжењери знања. На тај начин је рационализовано коришћење људских и временских ресурса организације.

3.2. Принцип рада експертског система

Експертски системи се углавном заснивају на правилима, па се неретко називају и системи засновани на правилима. Као што је у поглављу о исказној логици описано (Секција 2.1) правила садрже условни и последични део (Израз 2.1). Ако је условни део задовољен, механизам за закључивање извршава правило. Као последица извршења правила најчешће се стварају нове чињенице у радној меморији експертског система. Правила која стварају нове чињенице називају се продукциона правила, а из тог разлога се експертски системи још називају и продукционим системима.

Принцип рада експертског система се може описати као циклични процес од четири активности (Слика 3.2). Након учитавања базе знања из спољне меморије у меморију експертског система и евентуалног учитавања иницијалних података, експертски систем упоређује чињенице у радној меморији са условима правила (активност названа *Подударање*). Сва правила код којих постоји подударање, додају се у *агенду* (друга активност) – део меморије који садржи скуп конфликтних правила. Правила су конфликтна зато што у једном циклусу закључивања може да се изврши само једно од правила у агенди. Из тог разлога произилази да су правила *у конфликту*. Када је *агенда* попуњена, експертски систем врши избор правила које ће да се изврши (трећа активност). Критеријуми за избор правила су различити. На пример, то могу да буду приоритет правила, правило које је прво или последње додато у агенду. Изабрано правило се извршава (четврта активност). Извршењем се ствара нова чињеница коју експертски систем додаје у радну меморију као закључак. Тиме је завршен један циклус рада експертског система.



Слика 3.2: Принцип рада експертског система

Нове чињенице у радној меморији иницирају нови циклус резоновања, тако да се четири описане активности поново извршавају. Експертски систем тако ради у циклусима, а завршетак процеса закључивања се дешава када се успостави циљно стање. Циљно стање је најчешће случај када нема више нових чињеница у радној меморији (стагнација у резоновању), или се појавила чињеница која сама по себи представља циљно стање, тако да правило које то стање препозна зауставља рад експертског система.

3.2.1. Опис динамике рада експертског система

Динамика рада експертског система биће описана у примеру за сортирање карактера. Полази се од претпоставке да алфабет садржи само три карактера {*a, b, c*} и да ови карактери граде речи дужине пет карактера. Задатак експертског система је да сложи карактере од најмањег до највећег. Базу знања чини 3 правила која користе биграме за сортирање (Табела 3.3). Највећи приоритет је три, најмањи један.

Индекс правила	Приоритет	Правило
P1	3	ba→ab
P2	2	ca→ac
P3	1	cb→bc

Табела 3.3: база знања експертског система за сортирање стрингова

На следећој илустрацији (Слика 3.3) је запис целог процеса закључивања за улазну чињеницу – реч *cbaca*. У првој итерацији на основу подударања условног дела правила са биграмима речи (биграми су редом *cb, ba, ac* и *ca*) у агенди су се нашла сва три правила. Пошто правило *P1* има највећи приоритет, оно је изабрано за извршење. Последица његовог извршења је замена биграма *ba* (у речи *cbaca*) са *ab,* тако да је реч промењена у *cabca*. У другој итерацији постоји подударање премисе само код правила *P2* тако да је ово правило једино додато у агенди (конфликтном скупу). Извршењем правила *P2* мења се реч *cabca* у *acbca*. У тећој итерацији правила *P2* и *P3* су у агенди, пошто имају поклапање премиса (*cb* и *ca*). Правило има већи *P2* приоритет, тако да се поново извршава мењајући реч *acbca* у *acbac*. У четвртој итерацији у агенди су правила правила *P1* и *P3,* извршава се правио *P1* мењајући реч *acbac* у *acabc*. У петој итерацији опет је *P2* једино правило у агенди, извршава се и мења реч *acabc* у *aacbc*. У последњој итерацији *P3* је једино правило у агенди, извршава се и мења реч *aacbc* у *aabcc*. Након тога подударањем није пронађено правило за извршење, агенда је празна и експертски систем зауставља процес резоновања. Као резултат резоновања у радној меморији се налази сортиран стринг.

Итерација	Радна меморија	Подударање	Агенда	Избор правила
1	cbaca		P3, P1, P2	P1
2	cabca		P2	P2
3	acbca		P3, P2	P2
4	acbac		P1,P 3	P1
5	acabc		P2	P2
6	aacbc		P3	P3
	aabcc		∅	крај

Слика 3.3: пример рада експертског система за сортирање стрингова

У описаном примеру у радној меморији експертског система постоји само једна чињеница, стринг који се у итерацијама мења. Ово је карактеристика само чињеница које су представљене изведеним типовима података (класе, листе, низови). Друга специфичност у примеру је да стринг у другој итерацији садржи 2 иста биграма – *ca*: *ca*b*ca*. Међутим, правило се само једном извршава у истој итерацији тако да је промењен само први биграм *ca*b*ca* у *ac*b*ca*.

3.2.2. Уланчавање правила

Уланчавање правила је секвенцијално извршавање правила од почетка до краја процеса закључивања. Уланчавање као термин се користи јер су правила у низу узајамно условљена. Другим речима, следеће правило за извршење је условљено закључцима која производе правила у претходним итерацијама закључивања. Када се правило изврши, оно генерише закључак који, када се дода у радну меморију експертског система, постаје нова чињеница, која се затим испитује за поклапање са премисама правила у бази знања. Експертски систем издваја правила код којих постоји поклапање и додаје их у агенду као кандидате за извршење у наредној итерацији. На пример, ланац правила у закључивању је колона *Избор правила* на претходној слици (Слика 3.3).

Код продукционих система постоји две врсте закључивања – уланчавањем у напред и уланчавањем у назад. Уланчавање у напред је објашњено у претходном поглављу. Код уланчавања унапред полази се од познатих чињеница, а у итерацијама извршавањем правила додају се нове чињенице у радну меморију, све док се не оствари циљ резоновања.

Уланчавање у напред се примењује у случају малог броја иницијалних података (чињеница), а великог броја могућих закључака.

Уланчавање у назад почиње од хипотезе (претпоставке, циља), а затим се проналазе правила која могу да докажу њену ваљаност. У случају да дође до оповргавања хипотезе, систем поставља нове хипотезе и на исти начин покушава да докаже њену ваљаност. Уланчавање у назад се примењује у случају великог броја иницијалних података (чињеница), а малог броја могућих закључака.

3.3. Развој експертских система

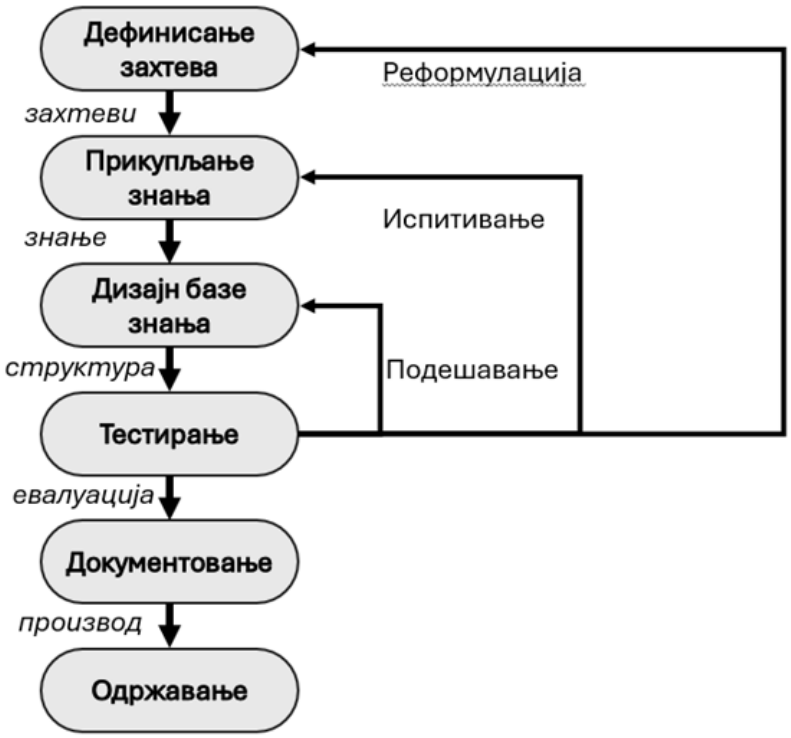
Развој експертских система је сличан развоју било које друге врсте софтверских производа са разликом да је тежиште на развоју базе знања. За остале компоненте система (механизам за закључивање, интерфејси и радна меморија) користе се већ готова окружења. Постоји шест фаза развоја експертских система. Развојни модел подсећа на модел водопада са повратним спрегама (Слика 3.4).

Прва фаза обухвата дефинисање захтева за експертским системом. У овој фази се анализира оправданост увођења експертског система. У ту сврху су корисне табеларно представљене информације на почетку овог поглавља (Табела 3.1 и 3.2). Исход прве фазе су артикулисани захтеви које експертски систем треба да испуни. Они представљају предуслов за улазак у другу фазу.

Друга фаза развоја експертског система подразумева прикупљање знања. Знање се односи на специфични домен употребе (Секција 2.2.1). Експертско знање се у начелу прикупља из документације. Некада то није довољно, већ је за комплетирање потребан интервју са експертом. Исход ове фазе су знања представљена у одговарајућој форми, најчешће табеларно, или у облику стабала одлучивања.

Затим следи фаза дизајна базе података. У овој фази се врши превођење знања у језик формалне логике. Дефинишу се структуре података и правила за закључивање. У случају комплекснијих захтева, да би се олакшало тестирање и одржавање експертског система, врши се дељење базе знања захтеваног домена на мање целине – модуле који су намењени уско специјализованом под-домену. Поред наведеног, фаза дизајна обухвата и

превођење формула базе знања у циљни програмски језик за развој експертских система. Тада је производ спреман за следећу фазу – тестирање.



Слика 3.4: Фазе у дизајну експертског система

Тестирање се разликује од осталих фаза по повратним спрегама са фазама које му претходе. Повратна спрега са фазом дизајна намењена је фином подешавању рада експертског система. Ако се у фази тестирања открију пропусти у прикупљеном знању, потребна су додатна испитивања извора знања како би се недостаци отклонили. Најозбиљније грешке које се могу открити у фази тестирања су погрешно, или непотпуно дефинисани захтеви за експертским системом. Таква интервенција је неопходна у случају да се проблем не може решити на било који други начин и она по правилу обухвата редефинисање захтева. На пример, ако је систем замишљен сувише амбициозно, домен закључивања постаје преобиман и тиме се губи једно од основних својстава експертског система, а то је да је специјализован.

Након што је систем прошао евалуацију тестирањем, потребно га је документовати. То је намена претпоследње фазе развоја. Документација је

неопходна да би се инжењерима знања објаснио у потпуности дизајн базе знања, начин коришћења структура података и система правила како би се база знања могла даље развијати и унапређивати. Ово је такође преко потребно у случају да се прелази на нови језик и на ново окружење за развој експертских система.

Последња фаза развоја експертских система је одржавање. Одржавање омогућава унапређивање перформанси експертског система, модификацију базе знања како би експертски систем одговорио у потпуности новим захтевима корисника и прилагођавање базе знања новим технологијама за развој експертских система. Без одржавања, период експлоатације (животни век) експертског система би био кратак, а исплативост његовог развоја би била упитна.

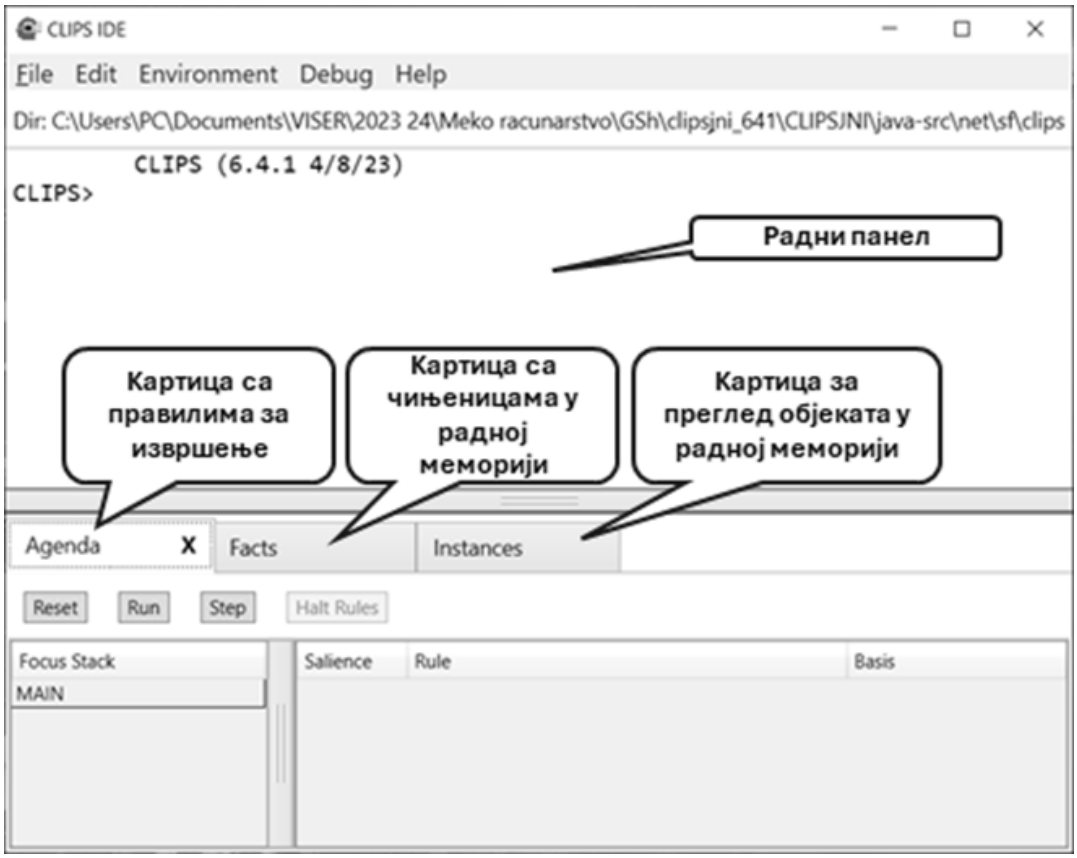
3.4. *CLIPS* – интегрисано окружење за развој експертских система

Као што је већ објашњено, у развоју експертских система тежиште је на развоју базе знања. За остале компоненте система (механизам за закључивање, интерфејси и радна меморија) користе се већ готова окружења. *CLIPS* (енгл. од *C-Language Integrated Production System*) представља једно такво окружење, развијено од стране *NASA Johnson Space Center*. *CLIPS* је данас бесплатан и доступан преко линка https://www.clipsrules.net/.

Поред инсталације, доступна је и комплетна документација. Инсталација се може преузети посредством *SourceForge* странице *CLIPS Rule Based Programming Language Files*. Од верзије 6.4.2, *CLIPS* представља интегрисано развојно окружење за оперативне системе *Windows* и *MacOS*, али исто тако за платформски независне Java/Swing Веб технологије.

Обзиром на актуелност *CLIPS* окружења постоји *online* подршка посредством *stackoverflow* сајта на линку https://stackoverflow.com/questions/tagged/clips, као и дискусионих група на *Google*-у https://groups.google.com/g/CLIPSESG и *SourceForge*-u под називом *CLIPS Rule Based Programming Language Discussion* на линку https://sourceforge.net/p/clipsrules/discussion/.

Након инсталације и стартовања *CLIPS* окружење се појављује на *desktop*-у (Слика 3.5). Испод линије менија налази се радни панел који служи за интерактиван рад инжењера знања са окружењем. У радном панелу је могуће уносити појединачне команде и програмски код писан у *CLIPS* језику.



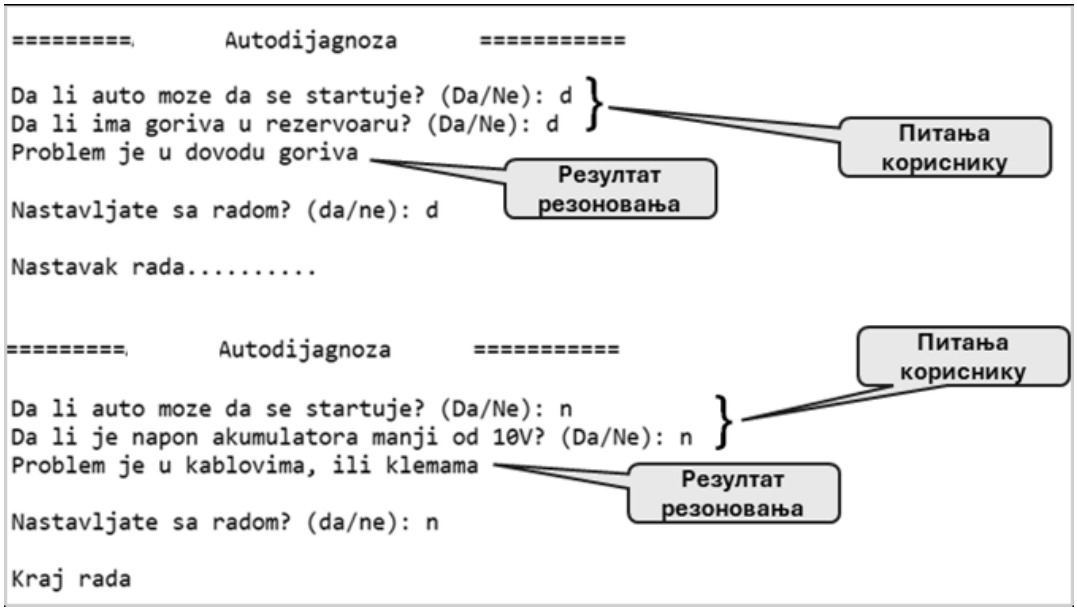
Слика 3.5:Изглед CLIPS радног окружења

У истом панелу окружење даје резултате команди, извршавања програмског кода и друге излазне поруке, као и поруке о грешкама, које омогућавају инжењеру знања развој експертског система. Испод радног панела су три картице. При стартовању окружења оне нису отворене. Отварају се преко *Debug* менија избором истоимених опција. Картица *Agenda* омогућава у току извршења експертског система праћење правила, услова за њихово извршење и помоћу команди *Reset, Run, Stop, Halt Rules* потпуну контролу извршења правила, па и експертског система у целини. Картица *Facts* омогућава праћење чињеница у радној меморији, док картица *Instances* омогућава преглед објеката у радној меморији.

На основу претходног описа може се закључити да је *CLIPS* уједно и програмски језик. Потпуна документација је доступна на линку https://www.clipsrules.net/documentation/, на коме се налази основни и напредни водич за програмирање, корисничко упутство за окружење и упутство за повезивање са другим софтверским производима и технологијама.

3.3.1. Пример за развоја експертског система у *CLIPS* језику

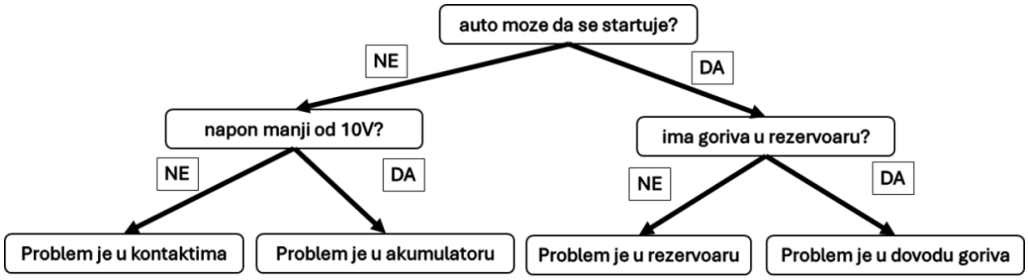
У циљу демонстрације развоја експертског система на бази исказне логике у *CLIPS* језику, искоришћен је пример за дијагностику квара на аутомобилу. У складу са методом развоја експертских система (секција 3.3.), најпре се врши дефинисање захтева. На основу тих захтева врши се осмишљавање интеракције са корисником (Слика 3.6).



Слика 3.6: осмишљавање интеракције са корисником

Питања која експертски систем поставља кориснику морају бити јасна. У циљу једноставније обраде корисничког уноса, одговори корисника морају да буду кратки, у најбољем случају *да/не* одговори. Уз свако питање потребно је да постоји инструкција за давање одговора. Поред тога експертски систем треба да омогући произвољан број сесија са корисником (на слици су приказане две), као и могућност да корисник заустави рад система.

У другој фази се врши прикупљање експертског знања. На основу тих знања је креирано стабло одлучивања за дијагностику квара на аутомобилу (Слика 3.7). На основу садржаја стабла може се уочити да интерни чворови садрже питања, а екстерни чворови дијагнозе. Поред тога, извршено је прилагођавање у коме ће експертски систем постављати питања кориснику на која ће он одговарати искључиво потврдно или одрично.



Слика 3.7: стабло одлучивања за дијагностику квара на аутомобилу

У следећој фази следи дизајн базе знања. Стабло одлучивања одређује минималан број правила која треба да омогуће рад експертског система – за сваку грану одлуке дефинише се посебно правило. Два правила намењена гранама интерних чворова називају се радним правилима (између кореног чвора и његове деце). Четири правила за гране према спољњим чворовима називају се циљним, пошто се њима завршава једна сесија закључивања.

Кориснику се прво питање (које се налази у кореном чвору - *auto moze da se startuje?*) приказује помоћу посебног - *безусловног* правила. То је правило које у условном делу нема никакав захтев за проверу истинитости и извршиће се на самом почетку рада експертског система (Слика 3.8). У *CLIPS* језику сви ентитети (нпр. правила, функције, чињенице, класе) дефинишу се унутар заграда. Прво се одређује врста ентитета службеном речју *def<врста_ентитета>*. У конкретном случају користи се *defrule* за дефинисање правила које се зове *inicijalno-pitanje*. Као ознака за импликацију користи се симбол **=>**, а пошто импликацији не претходи ништа, правило је безусловно. После импликације, у акционом делу правила функцијом *print* приказује се лого (*Autodijagnoza*) и прво питање кориснику са инструкцијом за договор. Прихватање одговора се врши *bind* наредбом која у локалну промењиву *?odgovor* (у *CLIPS* језику *?* је обавезни префикс за локалне промењиве) смешта очитавање са тастатуре наредбом

read. У следећој наредби правило креира чињеницу *auto-startuje* у радној меморији експертског система додељујући јој вредност локалне промењиве *?odgovor*.

```
(defrule inicijalno-pitanje
=>
(print crlf  crlf "=========Autodijagnoza==========")
(print crlf  crlf "Da li auto moze da se startuje? (Da/Ne): ")
(bind ?odgovor (read))
(assert (auto-startuje ?odgovor))
)
```

Слика 3.8: безусловно правило за прво питање кориснику

За разлику од претходног правила, радна и циљна правила су условна. Другим речима, зависе од чињеница које се налазе у радној меморији експертског система. Да би у потпуности сагледали поступак дизајна базе знања, биће представљено радно и циљно правило крајње леве путање стабла одлучивања (Слика 3.7). Радно правило крајње леве гране обрађује случај када је корисник одговорио негативно на прво питање (Слика 3.9). Назив правила треба да осликава његову намену. У конкретном случају правило је названо *auto-ne-startuje,* што значи да обрађује случај да је корисник негативно одговорио на претходно питање. Премиса (условни део правила) правила има два услова у конјункцији (наредба *and*). Први је да постоји чињеница под називом *auto-startuje* у радној меморији експертског система. Други услов је да вредност те чињенице (представљен локалном промењивом *?odgovor*) није потврдан одговор. У ту сврху је коришћена наредба *not*. У другом услову је коришћен користан регуларни израз *CLIPS* језика *?odgovor-read&da|Da|DA|d|D* , који издваја вредност из чињенице и проверава са пет узорака (*da,Da,DA,d,D*). Ако постоји чињеница *auto-startuje*, али не постоји поклапање његове вредности са узорком, израз у премиси правила је тачан и правило се извршава. При том се најпре поставља ново питање кориснику. Затим (као што је код претходног правила објашњено) се прихвата одговор корисника и смешта у локалну промењиву *?odgovor*. На крају се креира нова чињеница у радној меморији експертског система (*napon-akumulatora-nizak*) и додељује јој се вредност промењиве *?odgovor*.

```
(defrule auto-ne-startuje
(and (auto-startuje ?odgovor)(not (auto-startuje ?odgovor-read&da|Da|DA|d|D)))
 =>
(print "Da li je napon akumulatora manji od 10V? (Da/Ne): ")
(bind ?odgovor (read))
(assert (napon-akumulatora-nizak ?odgovor))
)
```

Слика 3.9: ӣример радноӣ ӣравила

Циљна правила су последња у ланцу закључивања. Њихови резултати су циљеви рада експертског система. Циљно правило (*ciljno-napon-12V-auto-ne-startuje*) крајње леве гране обрађује случај да је корисник одговорио негативно на оба претходна питања (Слика 3.10). Да би се ово правило извршило, потребно је да у радној меморији експертског система постоје обе чињенице *auto-startuje* и *napon-akumulatora-nizak*. Ови предуслови се испитују у условном делу правила. У акционом делу ово и друга циљна правила имају само једну *print* наредбу којом кориснику приказују дијагнозу квара на возилу.

```
(defrule ciljno-napon-12V-auto-ne-startuje
(napon-akumulatora-nizak ?odgovor)
(not(auto-startuje ?odgovor-read&da|Da|DA|d|D))
(not(napon-akumulatora-nizak ?odgovor-read&da|Da|DA|d|D))
 =>
(println "Problem je u kablovima, ili klemama")
)
```

Слика 3.10: ӣример циљноӣ ӣравила

Поред описаних, дизајнирана су правила која омогућавају контролу рада експертског система од стране корисника. Прво од правила за контролу тока (*pitanje-za-nastavak-rada*), је безусловно и има мањи приоритет (*salience -10*) од осталих правила (Слика 3.11). Употреба приоритета је била нужна у овом случају зато што већ постоји безусловно правило *inicijalno-pitanje* (Слика 3.8). Да би се избегао конфликт правила, додељен је нижи приоритет безусловном правилу за контролу. Треба напоменути да нула представља подразумевани приоритет правила. Правило *pitanje-za-nastavak-rada* у акционом делу поставља кориснику питање за наставак рада, прихвата одговор у истоимену локалну промењиву, а затим креира под именом *nastavak* нову чињеницу у радној меморији додељујући јој вредност коју је корисник унео.

```
(defrule pitanje-za-nastavak-rada
(declare (salience -10))
 =>
 (print crlf "Nastavljate sa radom? (da/ne): ")
 (bind ?odgovor (read))
 (assert (nastavak ?odgovor))
)
```

Слика 3.11: безусловно правило за контролу тока

Преостала два правила за контролу обрађују кориснички унос и у складу с тим се извршава једно од њих. У случају да је корисник изабрао наставак рада, извршава се правило названо *nastavak-rada* (Слика 3.12). У условном делу правила се испитује постојање чињенице *nastavak* и провера вредности на позитиван одговор регуларним изразом (као што је објашњено код правила *auto-ne-startuje*, Слика 3.10). Ако је израз у премиси истинит у акционом делу правила се исписује порука кориснику, наредбом *reset* врши се повратак система на почетно стање (пражњење радне меморије и агенде), а затим се наредбом *run* поновно покреће експертски систем.

```
(defrule nastavak-rada
 (nastavak ?odgovor-read&da|Da|DA|d|D)
 =>
 (print crlf "Nastavak rada.........." crlf)
 (reset)
 (run)
)
```

Слика 3.12: правило за наставак рада

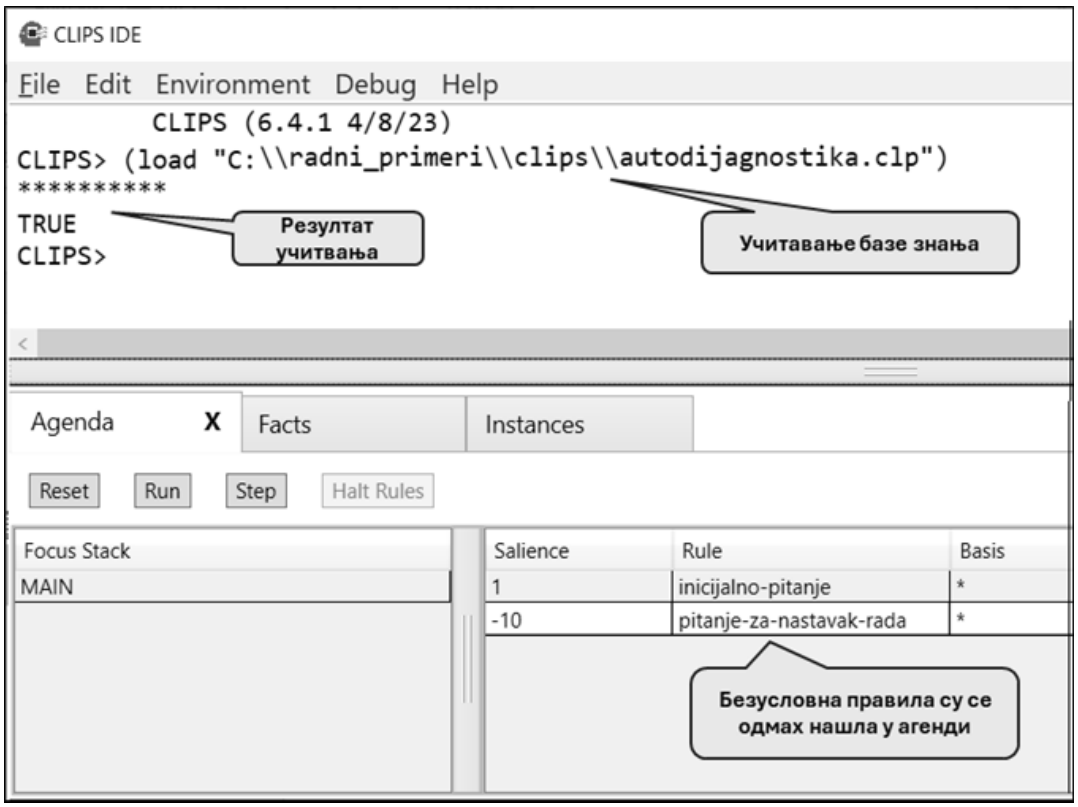
У случају да је корисник изабрао завршетак рада, извршава се правило названо *kraj-rada* (Слика 3.13). Као код претходног правила, у условном делу правила се испитује постојање чињенице *nastavak* и провера вредности на негативан одговор регуларним изразом. Ако је услов правила задовољен, наредбом *retract* избацују се све чињенице из радне меморије и функцијом *print* исписује се порука кориснику да експертски систем завршава с радом.

```
(defrule kraj-rada
 (nastavak ?odgovor-read&ne|Ne|NE|n|N)
 =>
 (retract *)
 (print crlf "Kraj rada" crlf crlf )
)
```

Слика 3.13: правило за крај рада

Креирана база знања се чува као *CLIPS* скрипт у једном или више *CLIPS* фајлова. То су фајлови са **.clp* екстензијом. База знања описана у фази дизајна је запамћена у фајл *autodijagnostika.clp*.

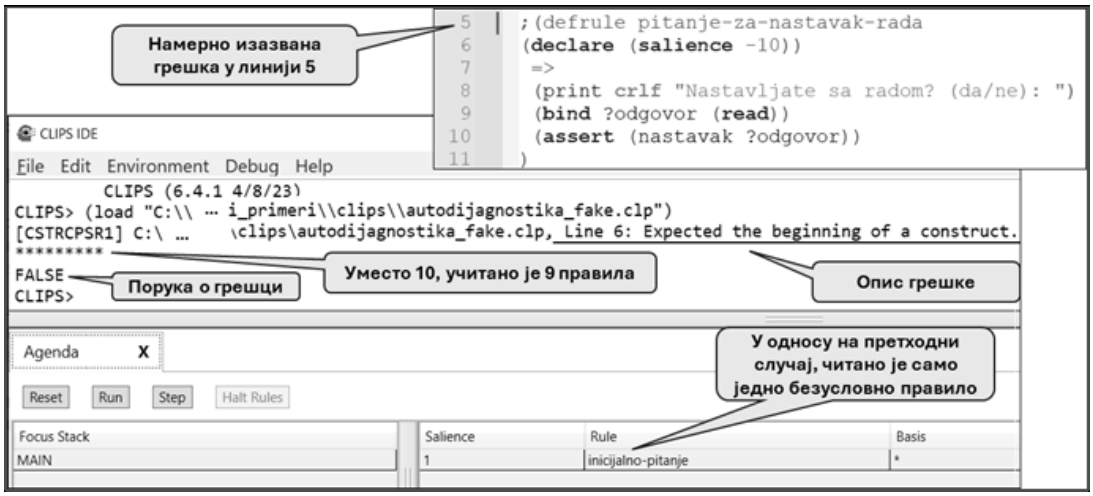
Следећа фаза развоја експертског система је тестирање. Тестирање се такође врши у *CLIPS* интегрисаном развојном окружењу. Најпре је се врши учитавање фајлова који садрже базу знања. У примеру на илустрацији је база знања садржана у фајлу *autodijagnostika.clp* (Слика 3.14). У ту сврху је искоришћена *CLIPS* наредба *load* којој је прослеђен као *string* параметар апсолутна путања и назив фајла.



Слика 3.14: учитавање базе знања и праћење стања у агенди правила

У току учитавања, окружење истовремено испитује синтаксну исправност *CLIPS* скрипта. У случају да је скрипт исправан, у радном панелу *CLIPS* окружења биће приказани симболи *, који представљају број учитаних правила (има их 10), праћени поруком *TRUE* (учитавање је у потпуности успело). Безусловна правила аутоматски су учитана у агенду за извршавање (правила *inicijano-pitanje* и *pitanje-za-nastavak-rada*).

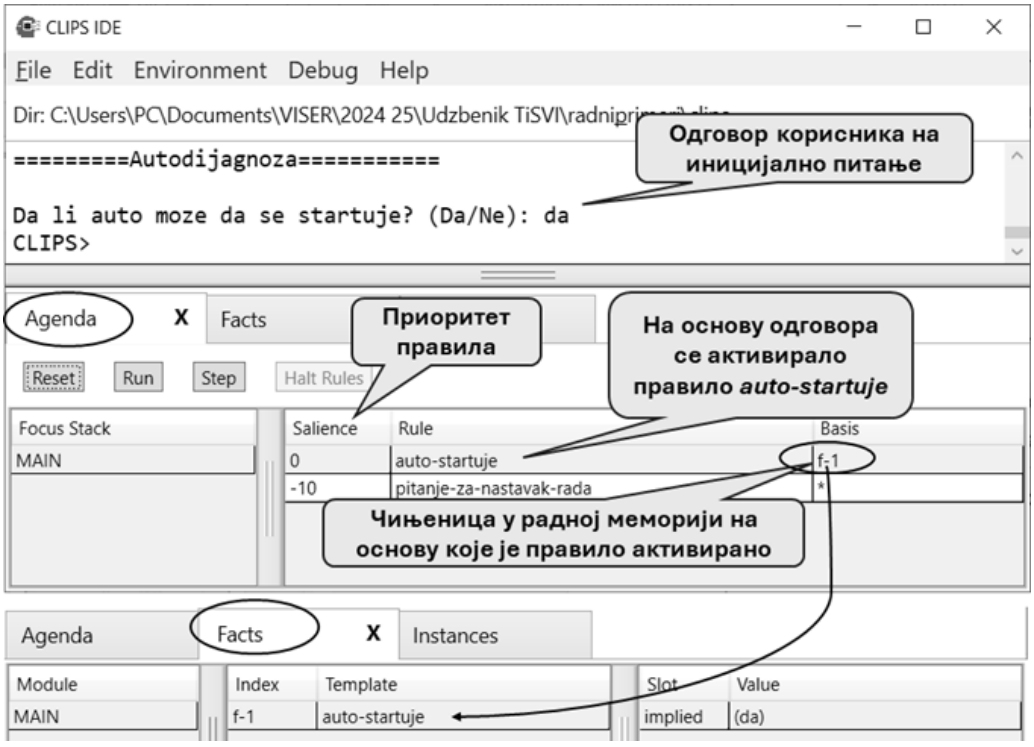
У случају да у *CLIPS* скрипту постоји грешка, окружење даје поруку о грешци, али ипак учитава исправне конструкције у скрипту. На следећем примеру направљена је намерна синтаксна грешка на правилу *pitanje-za-nastavak-rada* у бази знања (Слика 3.15), тако што је коментарисана прва линија декларације правила (у *CLIPS* језику коментар се започиње знаком **;**). При учитавању окружење је открило грешку и дало опис грешке (пошто је грешка у линији 5, порука је да у линији 6 недостаје почетак конструкције правила). Ипак, остала правила су учитана, што потврђује низ * (сада их има 9). У агенди је учитано овог пута само једно безусловно правило, а окружење је дало *FALSE* поруку, којом закључује да постоји синтаксна грешка у учитаном скрипту. У случају грешке код учитавања потребно је отворити скрипт фајл и на основу дате инструкције окружења извршити исправку.



Слика 3.15: пример учитавања скрипта са грешком

Успешно учитан скрипт се затим тестира помоћу картице *Agenda*, на којој постоје контролна дугмад која омогућавају различите врсте извршења експертског система. За тестирање најважније је дугме *Step* које омогућава извршење *корак по корак*. На следећем приказу је други корак резоновања,

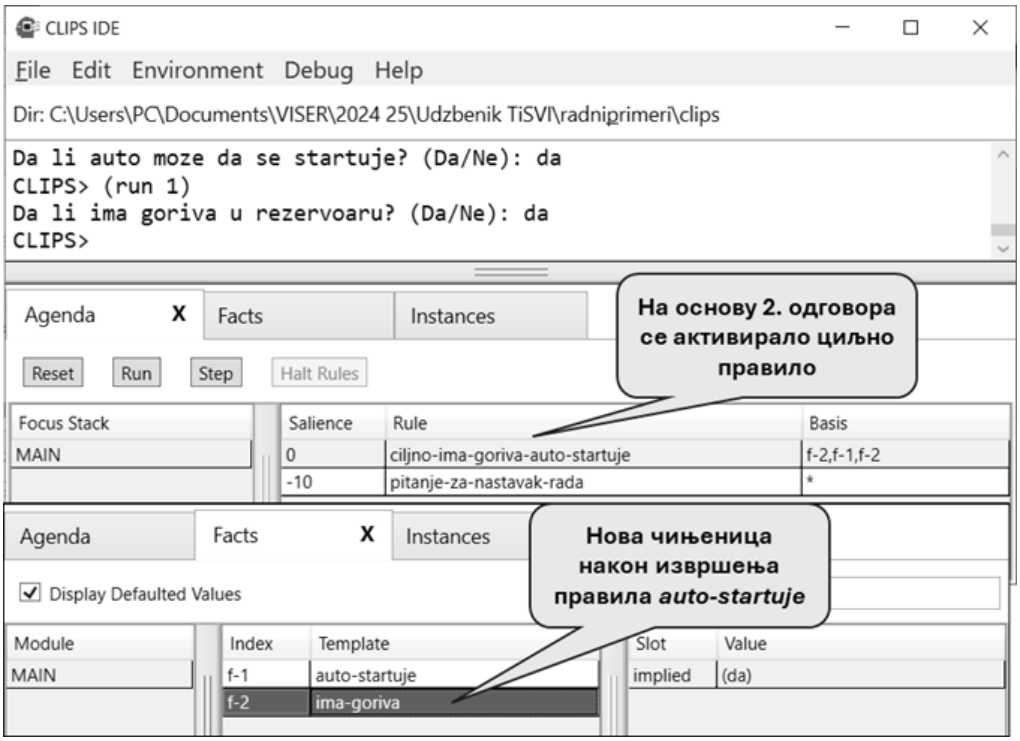
који се дешава након одговора корисника на иницијално питање (Слика 3.16). У агенди више нема правила *inicijalno-pitanje* што значи да је у потпуности извршено. Последица његовог извршења је нова чињеница у радној меморији којој је додељен идентификатор *f-1*. Картица *Facts* даје потпуни увид у детаље ове чињенице. Види се њен идентификатор (колона *Index*), њен назив (колона *Template*) и њена вредност (колона *Value*). Експертски систем је препознао поклапање чињенице *f-1* са условним делом правила *auto-startuje* тако да је ово правило активирано, што се може уочити у колони *Rule* у картици *Agenda*. Поред тога, у агенди је и даље безусловно правило *pitanje-za-nastavak-rada*. Обзиром да ово правило, намењено контроли тока, има мањи приоритет од свих осталих правила, извршиће се само ако не постоји нити једно друго правило за извршење.



Слика 3.16: праћење параметара експертског система у току тестирања

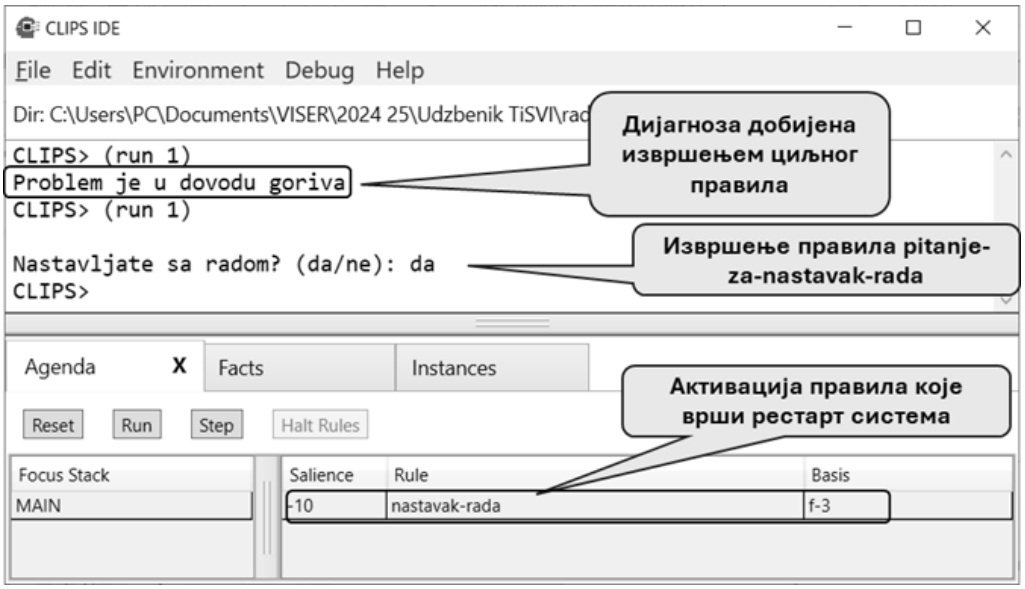
Обзиром да правила у агенди имају различит приоритет, у следећем кораку извршиће се правило већег приоритета. У конкретном случају то је правило *auto-startuje* (Слика 3.17). Ово правило поставља следеће питање корсинику („*Da li ima goriva u rezervoaru?*“), прихвата његов одговор на основу ког креира нову чињеницу у радној меморији названу *ima-goriva*. На картици *Facts* види се да је то чињеница индексирана *f-2,* да има вредност

da и да је омогућила активирање новог правила које се зове *ciljno-ima-goriva-auto-startuje*. Детаљи активације овог правила виде се на картици *Agenda* у колони *Basis* где се може видети да су услов његовог активирања чињенице *f-1* и *f-2*.



Слика 3.17: праћење рада експертског система у наставку интеракције са корисником

Последњи корак у резоновању (Слика 3.18) извршава циљно правило које даје дијагнозу о проблему на возилу. Пошто је након извршења циљног правила у агенди преостало само безусловно правило *pitanje-za-nastavak-rada*, његовим извршењем постављено је питање кориснику „*Nastavljate sa radom?*“, а након добијеног одговора, креирана је чињеница у радној меморији индексирана као *f-3*. Појавом ове чињенице у радној меморији, експертски систем је активирао правило *nastavak-rada*. У наредном кораку ово правило се извршава наредбама *reset* и *run* којима омогућава наставак рада експертског система.



Слика 3.18: Последњи корак у резоновању и контрола тока

Тестирање експертског система треба да се изврши за све случајеве употребе. То за конкретан пример значи да треба извршити уносе података који се очекују или не, коректне и некоректне. Тестирање се врши све док се не постигну очекивани одговори система и за коректне и за некоректне корисничке уносе.

Након тога следи документовање експертског система и то одвојена документација за кориснике и за инжењере знања. За инжењере знања посебно је значајно документовање помоћу коментара у *CLIPS* скрипту. За кориснике се праве стандардна корисничка упутства налик описима датим у овом поглављу.

Последња фаза развоја експертског система је одржавање, која подразумева отклањање грешака у току експлоатације система, прилагођавање система новим верзијама софтвера, унапређење и проширивање функционалности система.

3.5. Закључак о експертским системима

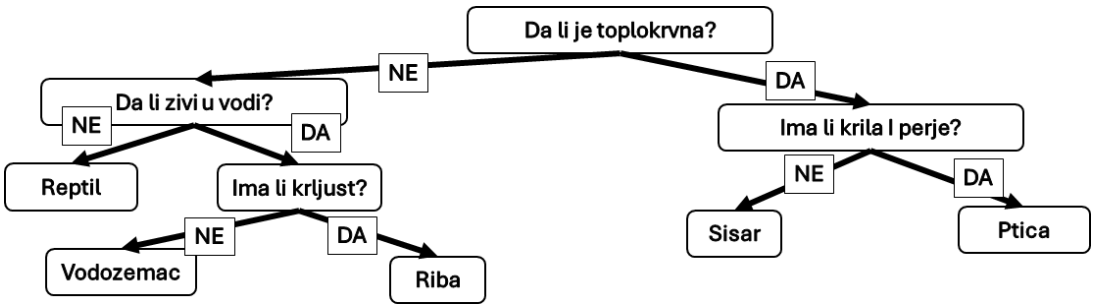
Експертски системи опонашају процес закључивања на начин на који то раде експерти. То су робусни системи који омогућавају брзо доношење одлука, конзистентност у одлучивању, при чему су доступни било када и тамо где људски експерти не могу да буду (било који вид екстремних услова, немогућих за опстанак). С друге стране, експертски системи су ограничени

дефинисаном базом знања. Промене у бази знања могуће су само уз интервенцију инжењера знања. Експертски системи, за разлику од других технологија вештачке интелигенције, омогућавају доношење закључака у врло комплексним и уско специјализованим доменима. Захваљујући томе користе се у медицини (дијагностички системи), у финансијама (управљање ризиком), у индустрији (одржавање система), у контроли и управљању пословним процесима, као и у многим другим областима.

Развој експертских система се састоји од шест фаза. Најважнија фаза је дизајн базе знања. База знања се састоји од правила. У току итеративног процеса закључивања, правила која се извршавају формирају ланац закључивања. У односу на ту чињеницу, начин на који процес закључивања може да се изврши јесте уланчавање у напред и уланчавање у назад. Начин закључивања може експлицитно да се подеси и нема утицаја на дизајн базе знања. У материјалу је представљен пример развоја експертског система за дијагностику квара на возилу. Коришћено је *CLIPS* интегрисано окружење за развој експертских система. Пример је урађен на бази исказне логике. Детаљно је приказан дизајн и тестирање базе знања.

3.6. Питања и задаци

1. Навести фазе развоја експертских система.
2. Које фазе развоја могу да се понављају и у којим случајевима?
3. Нацртати и објаснити принцип рада експертских система.
4. Објаснити како се решава конфликт правила у агенди.
5. На основу базе знања из табеле 10 (секција 3.2.1.) извршити сортирање стринга *cbaca*, ако је највећи приоритет 1, а најмањи приоритет 3 (обрнуто у односу на пример).
6. Ако је дато стабло одлучивања за откривање врсте кичмењака:



   a. Дизајнирати базу знања на бази исказне логике,

b. Имплементирати експертски систем у CLIPS интегрисаном развојном окружењу и
c. Тестирати експертски систем до постизања употребљивости за дате случајеве.

4. Расплинута (fuzzy) логика и примена у интелигентним системима

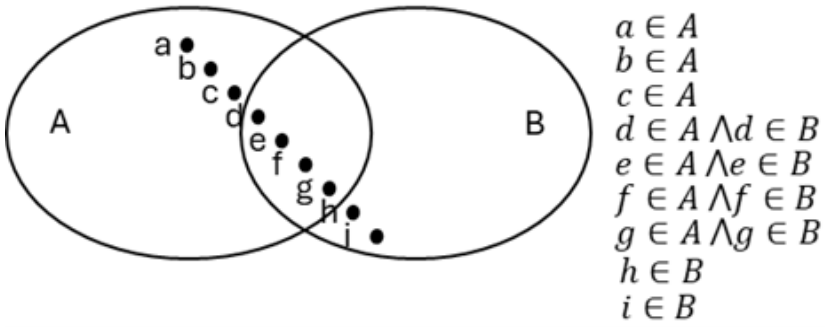
Расплинута (изворно *fuzzy*) логика представља проширење логике која омогућава закључивање на основу података који нису егзактни (подаци који су непрецизни, непотпуни). Као што је у трећој секцији објашњено, код математичке логике постоје две могуће вредности логичких формула: тачно и нетачно. Репрезентација ових константи зависи од нивоа апстракције и конкретног окружења у којима се логичке формуле имплементирају. Тако могу да буду симболи {⊤, ⊥} типични за математичку формулацију, или {1,0}, што је карактеристично за ниже програмске језике (на пример *C*), или {*true, false*} како је уобичајено код виших програмских језика (на пример *C#* и *Java*) и скрипт језика (на пример *JavaScript, Python*).

Недостатак исказне и предикатске логике је немогућност да се егзактно изразе тврдње као што су: *низак притисак*, *врућа вода, велика влажност, висок човек, светла боја коже*. На питање колико Целзијусових степени има *врућа вода*, или колико сантиметара је *висок човек*, добијају се различити и непотпуни одговори. Упркос томе, у свакодневном животу, у комуникацији на послу и ван њега, људи често користе тврдње које нису егзактне, односно тврдње које не могу да се квантификују (да се представе одређеном бројчаном вредношћу). Самим тим постоји и потреба да се развију системи за закључивање који могу да раде на основу података који нису егзактни.

4.1. Основни концепти расплинуте логике

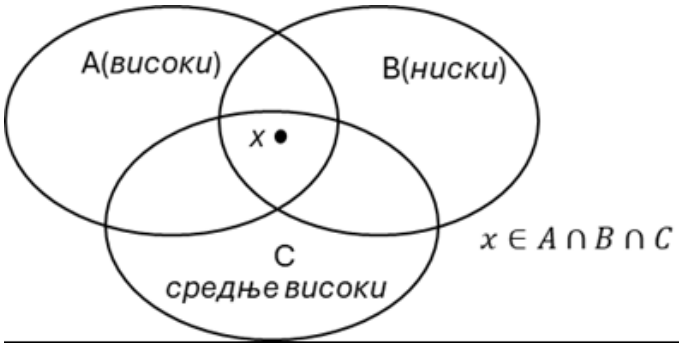
Јаз између могућности логике и потребе закључивања, представљен у уводном разматрању, решио је амерички научник азербејџанског порекла Лотфи Задех увођењем концепта расплинуте (*fuzzy*) логике (1973). Основа концепта је у проширењу теорије скупова расплинутим скуповима. Код класичне теорије скупова, вредност (објекат) припада, или не припада

једном или више скупова. На датом примеру (Слика 4.1) тачне су тврдње да објекти *a, b, c, d, e, f, g* припадају скупу *A* и да објекти *d, e, f, g, h, i* припадају скупу *B*. Са друге стране, нетачне су тврдње да објекти *a, b, c* припадају скупу *B,* као и да и објекти *h* и *i* припадају скупу *A*.



Слика 4.1: пример придавања вредности (објеката) по класичној теорији скупова

На пример, ако би истражитељи питали сведоке у вези починиоца неког противзаконитог дела „Колико је особа *x* висока?“, претпостављајући да особа *x* има 175 цм, највероватније да би људи који су нижи од ~160 цм сматрали особу *x високом*, а људи виши од ~190 цм би је сматрали *ниском*. Они који су приближне висине особе, сматрали би је *средње високом*. То значи да би за исту особу *x* добили три различите интерпретације њене висине. Другим речима, добили би вредности које нису егзактне. Уколико би се покушало то представити класичном теоријом скупова (Слика 4.2), једина информација би била да особа *x* припада пресеку скупова *високих, ниских* и *средње високих* људи. Дакле, не би било могуће сазнати колика је приближна висина човека (упркос томе да људи поседују искуства у процени висине особа).

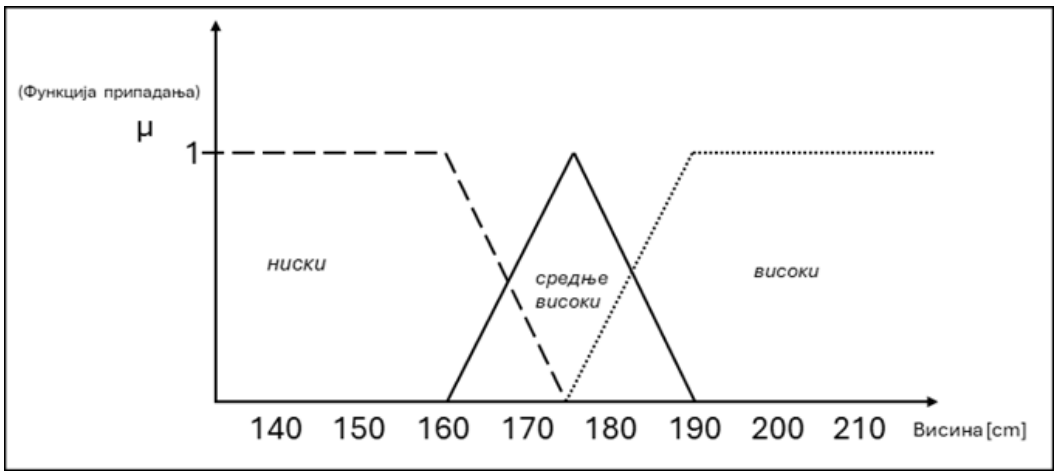


Слика 4.2: интерпретација класичном теоријом скупова придавања особе x скуповима A,B,и C

4.1.1. Расплинути скупови и функција припадања

Лотфи Задех је проширио теорију скупова могућношћу да вредност, или објекат може да припада једном, или више скупова у одређеној мери. Та мера се изражава функцијом припадања (енгл. *Membership function*). Наведено значи да се вредностима, или објектима, поред информације коју носе, придружују и вредности функције припадања скуповима. Скупови које чине објекти са придруженим вредностима функције припадања, називају се расплинутим (*fuzzy*) скуповима.

Дефинисање расплинутог скупа је заправо дефинисање његове функције припадања у односу на величину која се разматра. Примењујући теорију расплинутих скупова на претходном примеру, величина која се разматра је *висина*. Дефинисана су три расплинута скупа (Слика 4.3): *ниских* (приказан испрекиданом линијом), *средње високих* (приказан пуном линијом) и *високих* особа (приказан тачкастом линијом). Величина која се разматра (*висина*) представљена је апсцисом, а функција припадања је представљена ординатом. Вредности представљене на апсциси чине опсег од интереса (енгл. *Universe of Discourse*). Уобичајено је да су границе опсега најмање и највеће вредности посматране величине, које се очекују као улазни параметри за резоновање. То значи да у координатном почетку не мора да буде вредност нула, већ доња граница опсега од интереса. На ординати максимална вредност функције припадања је неименован реални број који може да буде у интервалу [0,1].



Слика 4.3: Графичко представљање расплинутих скупова

Поред графичке интерпретације (која је најочигледнија), функција припадања се може изразити као и свака друга функција табеларно и математичким изразом. На пример, за расплинут скуп *ниски* функција припадања је представљена следећом табелом и изразом (Слика 4.4). Расплинут скуп *ниски* има трапезоидну графичку репрезентацију (претходна слика). На основу табеле лево (Слика 4.4) види се дистрибуција функције припадања само на дискретне вредности висина. Обзиром да у реалности висина има континуалне вредности, најпрецизнија репрезентација је математичка. У математичкој репрезентацији види се јасно да функција припадања има вредност нула за све висине чија је вредност 175 цм или већа, односно да има вредност један за све висине чија је вредност 160 цм или мања. У опсегу у коме је висина већа од 160 цм и мања од 175 цм, функција припадања је опадајућа линеарна дуж, представљена линеарном једначином са негативним коефицијентом који одређује нагиб стрмине и константом која одређује помак у односу на ординату. За сва три расплинута скупа из примера закључује се да су линеарни и састављени од фрагмената (делова). Три фрагмента (вредност 0,1 и стрмина) имају скупови *ниски* и *високи*, док скуп *средње високи* садржи два фрагмента (растући и опадајући линеарни).

висина	μ(ниски)
140	1
150	1
160	1
165	0.666666
170	0.333333
175	0
180	0
190	0
200	0
210	0

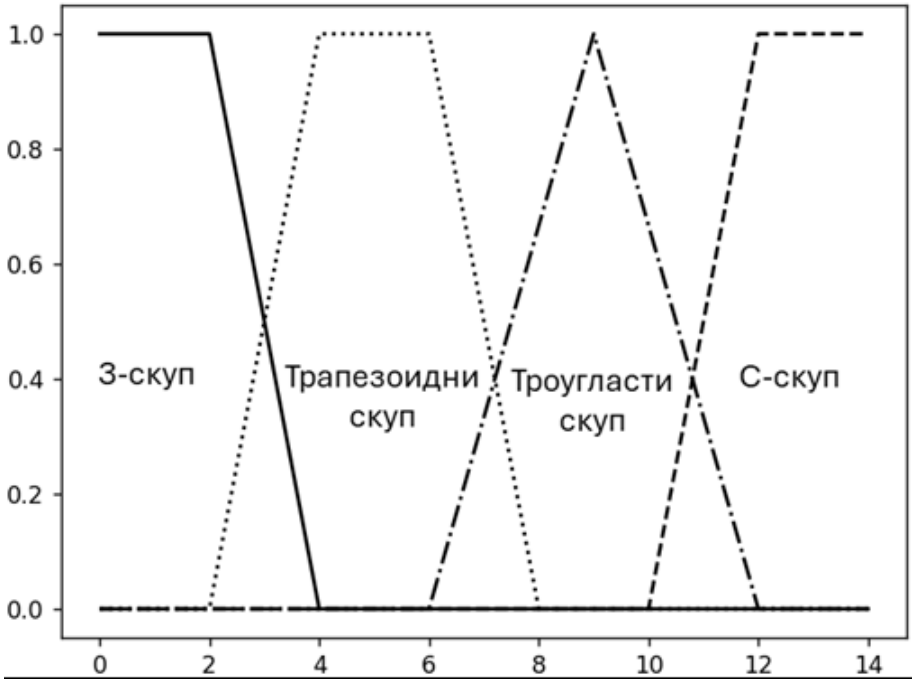
$$\mu(\textit{ниски}) = \begin{cases} 0 & ,x \geq 175 \\ -\frac{1}{15}x + 11\frac{2}{3} & ,160 < x < 175 \\ 1 & ,x \leq 160 \end{cases}$$

Слика 4.4: представљање скупа ниских особа табеларно и математичким изразом

4.1.2. Карактеристичне врсте расплинутих скупова

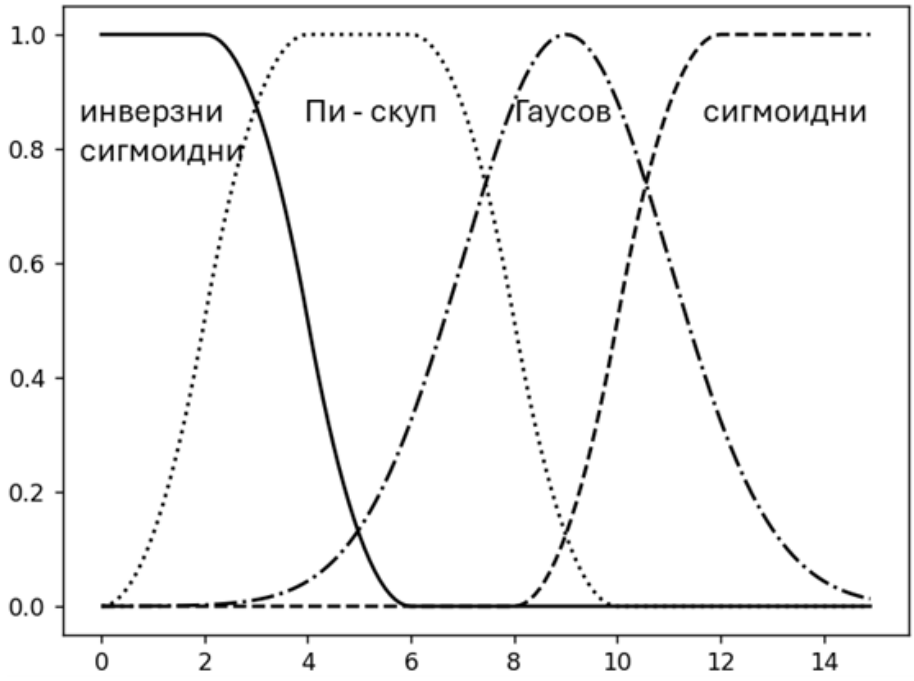
У реалним системима за закључивање на бази расплинуте логике постоје унапред дефинисани расплинути скупови, који омогућавају брзу имплементацију софтверског решења проблема. Најчешће коришћени линеарни скупови су З-скуп, С-скуп, трапезоидни и троугласти скуп (Слика 4.5). З-скуп је тако назван пошто подсећа на латинично слово *Z*. З-скуп се још назива и скуп ограничен са десне стране (енгл. *R Fuzzy Set*). С-скуп је

тако назван пошто подсећа на латинично слово *S*. С-скуп се још назива и скуп ограничен са леве стране (енгл. *L Fuzzy Set*). Трапезоидни скуп се још назива и скуп ограничен с леве и десне стране (енгл. *L R Fuzzy Set*). Троугласти скуп је сличан трапезоидном, с тим да му се горња темена налазе у истој тачки.



Слика 4.5: често коришћени линеарни расплинути скупови

Поред линеарних, постоје и карактеристични нелинеарни расплинути скупови. Најчешће коришћени нелинеарни скупови су сигмоидни и инверзни сигмоидни скупови, *Пи*-скуп, и Гаусов скуп (Слика 4.6). Сигмоидни скуп је тако назван пошто има изглед сигмоидне функције. Инверзни сигмоидни скуп је тако назван пошто има изглед сигмоидне функције. *Пи*-скуп је тако назван пошто има изглед сличан грчком слову π, док је Гаусов скуп тако назван пошто има изглед Гаусове функције.



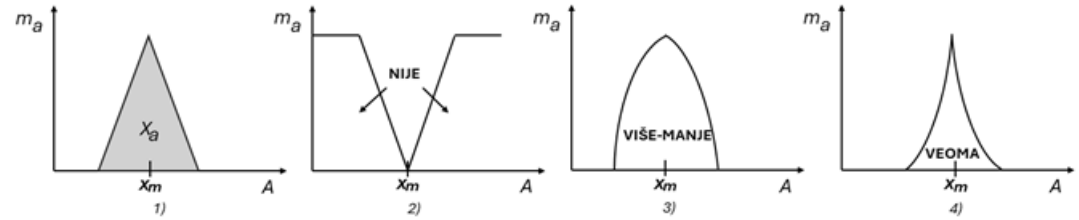
Слика 4.6: често коришћени нелинеарни раслинути скупови

У зависности од окружења називи ових скупова су различити, као и начини конструкције. Поред наведених скупова постоје и други који се изводе као комбинације (најчешће производи две или више функција) у циљу добијања других облика расплинутих скупова који би одговарали специфичним захтевима имплементације.

Поред тога, окружења за развој система на бази расплинуте логике омогућавају и представљање дискретне вредности. Постоји потреба да се на основу очитане (улазне) вредности генерише расплинут скуп који ће моћи да се укључи у процес резоновања. У савременим окружењима постоје два начина да се то омогући. Један начин је имплицитни, који подразумева да само развојно окружење врши *расплињавање* дискретне вредности како би закључивање могло да се изврши. Други начин је да у окружењу постоји адекватни расплинут скуп (нпр. *SingletonFuzzySet* у окружењу *FuzzyJToolkit*), код кога је улазна дискретна вредност представљена усправном линијом висине 1 (један, као вредност функције припадања) у одговарајућој тачки на апсциси.

Поједина окружења за развој система на бази расплинуте логике омогућавају и примену предефинисаних унарних операција над функцијом

припадања расплинутог скупа. На тај начин врши се трансформација расплинутог скупа, што као последицу даје измењено резоновање читавог система. Ови унарни оператори се називају модификатори. На следећој илустрацији (Слика 4.7) представљен је расплинут скуп X_a (1) и пример његових модификација: негација X_a (2), *више-мање* X_a (3) и *веома* X_a (4).



Слика 4.7: Примери модификатора расплинутих скупова

Модификација *више-мање* и *веома* трансформишу расплинут скуп X_a у нелинеарни. Модификација *више-мање* доводи до повећања функције припадања скупу X_a за улазне вредности блиске $\boldsymbol{x_m}$. На тај начин врши се *ублажавање* критеријума припадања улазних вредности скупу X_a, ако су блиске $\boldsymbol{x_m}$. Модификација *веома* доводи до смањивања функције припадања скупу X_a за улазне вредности блиске $\boldsymbol{x_m}$. На тај начин врши се *појачавање* критеријума припадања улазних вредности скупу X_a, ако су блиске $\boldsymbol{x_m}$. Поред наведених постоје и други модификатори, као што је *нормализатор*, који расплинути скуп трансформише тако да су вредности функције припадања увек у [0,1] интервалу, или модификатор за истицање (*интензификатор*) који трансформише троугласти расплинути скуп у скуп који има облик *Гаусовог* звона тако да се у близини граница скупа *појачава* резоновање, а у близини његових максималних вредности резоновање се *ублажава*.

4.1.3. Логички оператори у расплинутој логици

Обзиром да се расплинута логика заснива на теорији расплинутих скупова, логички оператори се интерпретирају на друкчији начин него што је то у дискретној логици. Они се односе на одређивање вредности резултујуће функције припадања, у случају да вредност промењиве (величине) од интереса припада у више расплинутих скупова. Тако конјункција (логички И оператор) представља пресек расплинутих скупова. Она је интерпретирана као функција која враћа најмању од вредности функција припадања скуповима за задату вредност промењиве од интереса (Израз 4.1).

$$\mu_{A\cap B}(x) = min(\mu_A(x), \mu_B(x)\ ) \qquad \text{И.4.1}$$

Расплинути скупови су представљени симболима *A* и *B*, вредност промењиве од интереса је *x,* док је $\mu_{A\cap B}(x)$ вредност функције припадања промењиве *x* пресеку скупова *A* и *B*. Види се да је резултанта мања од две вредности функције припадања $\mu_A(x)$ и $\mu_B(x)$. На следећој илустрацији је пример примене оператора конјункције у расплинутој логици (Слика 4.8). Карактеристично је да ако је једна од функције припадања 0, онда је резултујућа функција припадања такође 0 (нула). Резултујућа функција припадања је 1 само ако су обе функције припадања 1 (један).

$\mu_A(x)$	$\mu_B(x)$	$\mu_{A\cap B}(x)$
0	0.3	0
0.3	0.5	0.3
0.6	0.8	0.6
1	0.8	0.8
1	1	1

Слика 4.8: Примена оператора конјункције у расплинутој логици - пример

Дисјункција представља унију расплинутих скупова, интерпретирану функцијом која враћа највећу од вредности функција припадања скуповима за задату вредност промењиве од интереса (Израз 4.2).

$$\mu_{A\cup B}(x) = max(\mu_A(x), \mu_B(x)\ ) \qquad \text{И.4.2}$$

Као и у претходном случају, расплинути скупови су представљени симболима *A* и *B*, вредност промењиве од интереса је *x,* док је $\mu_{A\cup B}(x)$ вредност функције припадања промењиве *x* унији скупова *A* и *B*. Види се да је резултанта већа од две вредности функције припадања $\mu_A(x)$ и $\mu_B(x)$. На следећој илустрацији је пример примене оператора дисјункције у расплинутој логици (Слика 4.9). Карактеристично је да само у случају да ако су обе функције припадања 0, онда је резултујућа функција припадања такође 0. Резултујућа функција припадања је 1 ако је барем једна функција припадања 1 (један).

$\mu_A(x)$	$\mu_B(x)$	$\mu_{A \cup B}(x)$
0	0.3	0.3
0.3	0.5	0.5
0.6	0.8	0.8
1	0.8	1
1	1	1

Слика 4.9: Примена оператора дисјункције у расплинутој логици - пример

Негација је унарни логички оператор, а његова интерпретација у расплинутој логици слична је као у вероватноћи (Израз 4.3). Пошто су вредности функције припадања у интервалу [0,1], онда је негација функције заправо разлика јединице и вредности функције припадања расплинутом скупу за задату вредност промењиве.

$$\mu_{\neg A}(x) = 1 - \mu_A(x) \qquad \text{И.4.3}$$

Расплинут скуп је представљен симболом *A,* вредност промењиве од интереса је *x,* док је $\mu_{\neg A}(x)$ вредност негације функције припадања промењиве *x* скупу *A*. На следећој илустрацији је пример примене оператора негације у расплинутој логици (Слика 4.10).

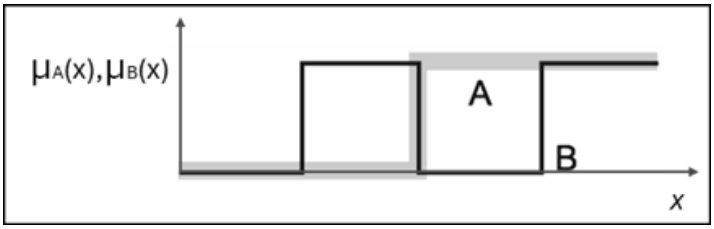
$\mu_A(x)$	$\mu_{\neg A}(x)$
0	1
0.3	0.7
0.6	0.4
0.8	0.2
1	0

Слика 4.10: Примена оператора негације у расплинутој логици - пример

Карактеристични случај је када промењива *x* не припада скупу *A* ($\mu_A(x) = 0$). Тада је негација максимална вредност функције припадања ($\mu_{\neg A}(x) = 1$). Други случај је када промењива *x* има вредност функције припадања скупу *A* један, тако да је негација вредност функције припадања нула ($\mu_{\neg A}(x) = 0$).

На претходним примерима приказане су дискретне вредности добијене применом логичких оператора над функцијама припадања неке варијабле *x* расплинутим скуповима *A* и *B*. Иста правила употребе логичких оператора примењују се и на читаве расплинуте скупове. На пример два расплинута

скупа *A* и *B* су представљени графички (Слика 4.11) – скуп *A* сивом дебљом, а скуп *B* црном тањом линијом. Специфичност ова два скупа је што омогућавају дискретно резоновање у условима расплинуте логике. Скупови су такви да постоје само две дискретне вредности функција припадања скуповима као могуће {0,1}. Скуп *A* је конвексан, али скуп *B* има два одвојена сегмента.

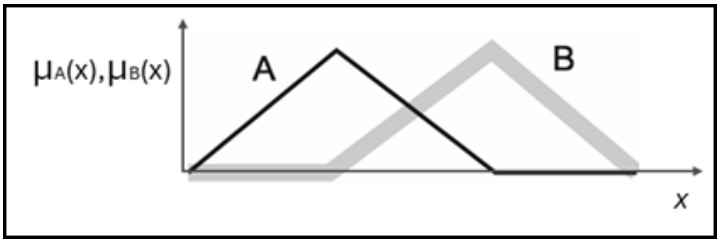


Слика 4.11: расплинути скупови A *и* B

Резултујући расплинути скупови добијени применом логичких оператора дисјункције, конјункције и негације су представљени на следећој илустрацији (Слика 4.12). Разлика у односу на класичну теорију скупова је само у томе што су расплинути скупови представљени у координатном систему промењиве од интереса и функција припадања скуповима.

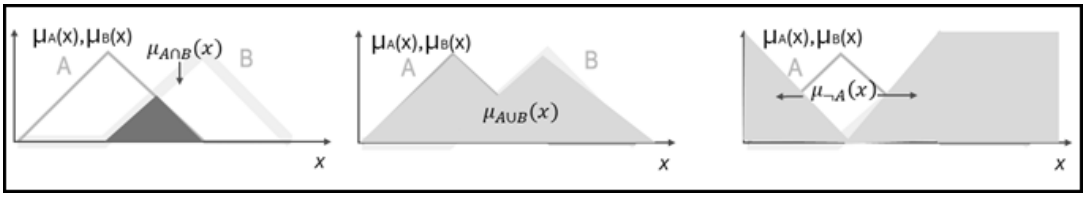
Слика 4.12: примена логичких оператора на читаве расплинуте скупове

Приказани су расплинути скупови (скуп *A* црном тањом, а скуп *B* сивом дебљом линијом) троугластог облика (Слика 4.13). Оба скупа су конвексна, али су вредности функција припадања континуалне у опсегу [0,1].



Слика 4.13: троугласти расплинути скупови

Резултујући расплинути скупови добијени применом логичких оператора дисјункције, конјункције и негације су представљени на следећој илустрацији (Слика 4.14). Применом логичких оператора добијене су резултујући скупови (означене површине). Најмања је конјункција, која обухвата само пресек скупова *A* и *B*. Дисјункција обухвата унију ова два скупа. Негација има највећу површину обзиром да је резултујућа вредност функције припадања овог скупа нула само у једној тачки – у којој је $\mu_A(x) = 1$, иначе $\mu_{\neg A}(x) \geq 0$ у свим осталим сегментима.



Слика 4.14: операције над расплинутим скуповима

Логички оператори и концепти представљени у овом поглављу представљају основ за прављење логичких формула и коришћења правила у системима за закључивање на бази расплинуте логике.

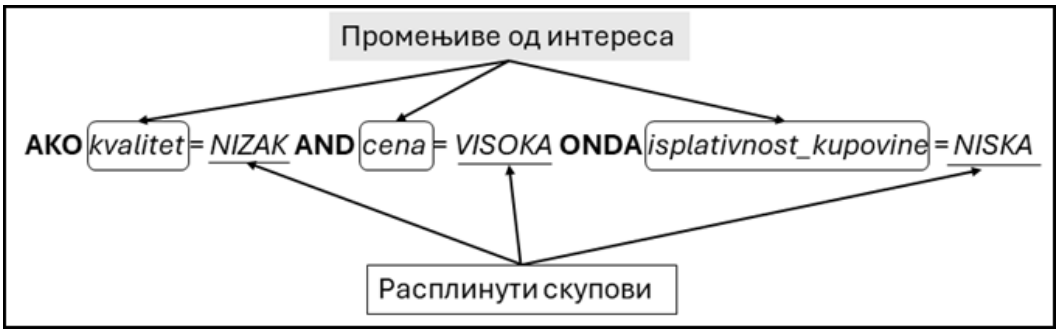
4.2. Системи за закључивање на бази расплинуте логике

Као што је у уводном разматрању о расплинутој логици наведено, постоји потреба да се омогући закључивање на бази непотпуних и нејасних (не-егзактних) података. Основни концепти расплинуте логике (промењиве и придружени расплинути скупови) и логички оператори прилагођени за операције над расплинутим скуповима су неопходни али не и довољни да би се то омогућило. Важни сегменти у решавању овог проблема су правила и механизми за закључивање на бази расплинуте логике.

4.2.1. Правила у расплинутој логици

Правила на бази расплинуте логике у основи имају исту структуру као и правила описана у поглављу о исказној и предикатској логици. Разлика је у томе што правила користе промењиве представљене расплинутим скуповима, као и логичке операторе специфичне за расплинуте скупове. У следећем примеру представљено је правило коришћено у одређивању исплативости куповине на основу квалитета и цене робе (Слика 4.15). У премиси правила налазе се две промењиве чије се вредности испитују у којој мери припадају расплинутим скуповима. Тако се промењива *kvalitet*

испитује у којој мери припада расплинутом скупу *NIZAK* ,а промењива *cena* испитује у којој мери припада расплинутом скупу *VISOKA*. Ова два испитивања су повезана логичком конјункцијом (*AND*), што значи да ће резултат испитивања промењивих у премиси правила дати резултујући расплинут скуп који ће представљати пресек скупова ниске цене и високог квалитета. Као последица извршења правила из примера, промењива *isplativost_kupovine* добија као вредност расплинут скуп *NISKA*.



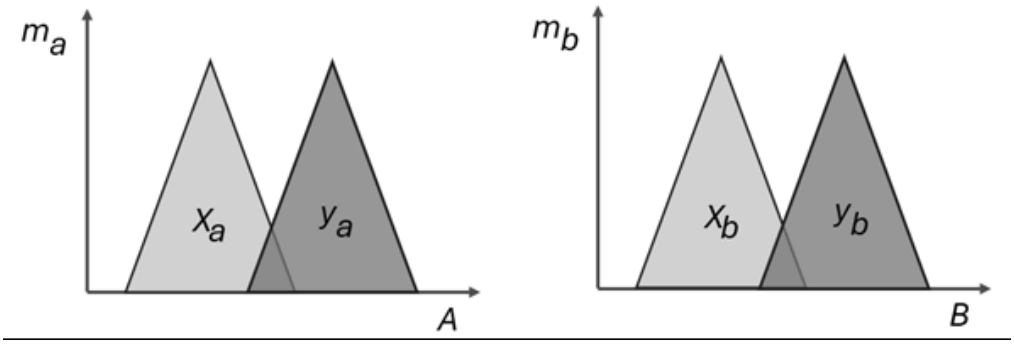
Слика 4.15: структура правила у расплинутој логици

Приликом коришћење правила у процесу закључивања, улазни подаци могу да буду дискретне вредности, или расплинути скупови. Механизам за закључивање испитује која правила могу да се изврше и затим извршава правило код кога су добијене највеће вредности функција припадања за дате улазне вредности. Као последица извршења, промењива у акционом делу правила (излазна промењива) добија као вредност расплинути скуп који ће представљати резултанту операција над расплинутим скуповима у премиси правила.

Важна напомена је да су правила у расплинутој логици вишег нивоа апстракције него што је то са правилима исказне и предикатске логике. Она су прилагођена више говорном језику него што је то случај са правилима предикатске логике. Правила у расплинутој логици одговарају експертима ради уношења (кодирања) знања у систем. Механизми који омогућавају њихово извршење су вишег нивоа комплексности него што је то случај са механизмима за закључивање у исказној и предикатској логици обзиром да се раде операције над скуповима (операције над подацима са више димензија).

4.2.2. Методе резоновања у расплинутој логици

Најчешће коришћене методе резоновања у расплинутој логици су *Mamdani* и *Takagi-Sugeno* метода. На следећем примеру дато је објашњење ове две методе. Дате су две величине означене са *A* и *B*. Свака од њих је дефинисана са по два расплинута скупа, за величину *A* то су x_a *i* x_b, док за величину *B* то су y_a *i* y_b (Слика 4.16).

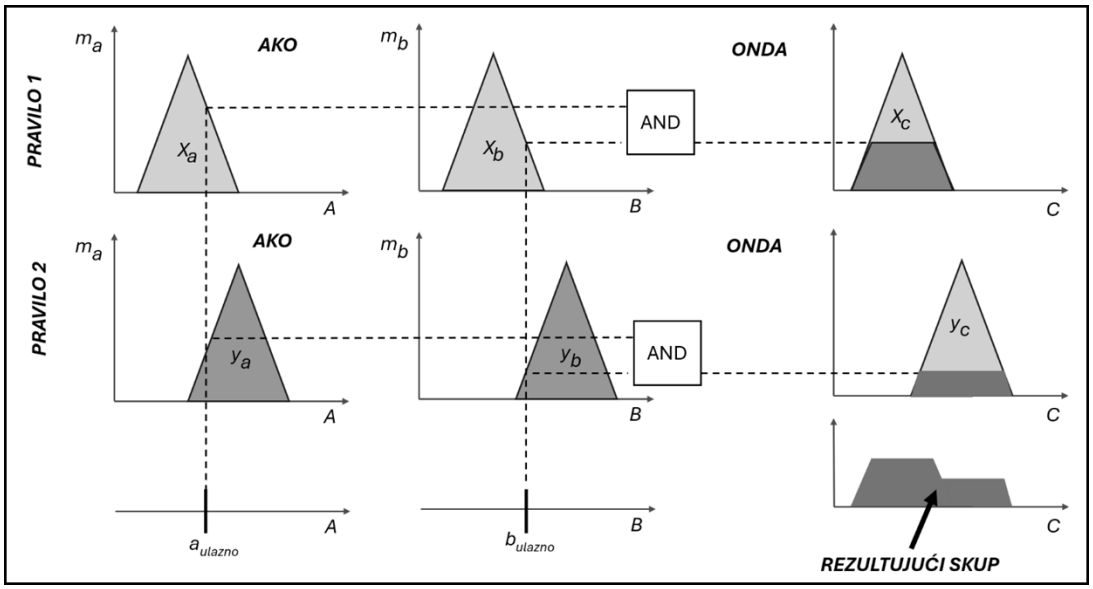


Слика 4.16: Величине означене са A *и* B и њихови расплинути скупови

Затим, дефинисана су два расплинута правила:

PRAVILO 1:	**АКО X_a AND X_b ОНДА X_c**
PRAVILO 2:	**АКО Y_a AND Y_b ОНДА Y_c**

Систем за закључивање *Mamdani* методом даће следећи излаз (Слика 4.17). У доњем делу слике представљене су улазне вредности величина *A* и *B* (a_{ulazno} и b_{ulazno}). Систем одређује функцију припадања за a_{ulazno} и b_{ulazno}, а затим се врши коњункција (*AND* операција), која даје као резултат мању од две улазне вредности (у случају оба правила то су функције припадања за b_{ulazno}).



Слика 4.17: Принцип рада система за закључивање Mamdani методом

Резултујућа вредности се даље користи као функција припадања за расплинуте скупове излазне величине *C*: у правилу 1 то је скуп x_c, док је у правилу 2 то скуп y_c.

Важно је уочити да су оба правила извршена. За разлику од система за закључивање на дискретној логици (код којих се бира једно правило за извршење у једној итерацији закључивања), код система за резоновање на бази расплинуте логике *Mamdani* методом, извршавају се сва правила формирајући резултујући расплинути скуп, који се даље користи како би се добила дискретна вредност као коначан резултат резоновања. Описани процес дискретизације је познат по називу *дефазификација*.

Takagi-Sugeno метода (неретко цитирана као *TS* метода) као и *Mamdani* метода користи расплинута правила. Суштински се разликује од *Mamdani* методе по томе што, да би се дошло до исхода резоновања, уместо расплинутих скупова, у акционом делу правила се налазе дефиниција функција. На следећем примеру дефинисана су два расплинута правила, која у премисама користе конјункцију расплинутих скупова X_a и X_b, односно Y_a и Y_b, али у акционом делу се уместо расплинутих скупова садрже функције f_1 и f_2.

PRAVILO 1:	AKO X_a AND X_b **ONDA** $\boldsymbol{f_1(X_a, X_b)}$
PRAVILO 2:	AKO Y_a AND Y_b **ONDA** $\boldsymbol{f_2(X_a, X_b)}$

Takagi-Sugeno метода се често користи у имплементацији различитих врста контролера на бази расплинуте логике. Другим речима, омогућава примену расплинуте логике у процесној контроли.

4.2.3. Дефазификација

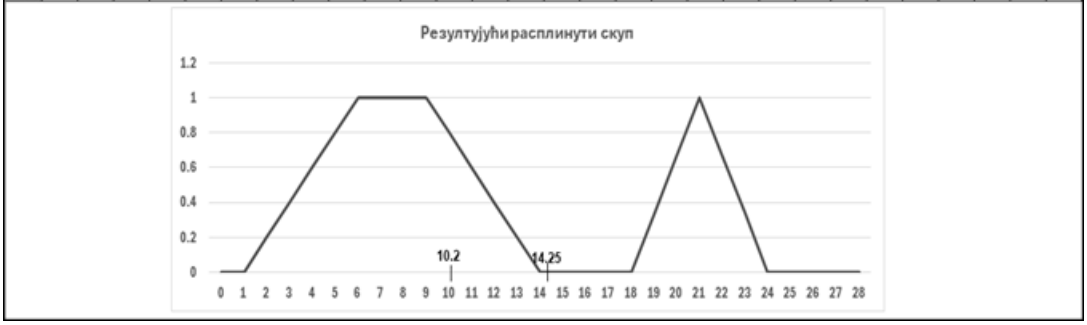
Дефазификацијом се од расплинутог скупа добија дискретна вредност. Она се примењује на крају процеса резоновања, на резултујући расплинут скуп како би се добила вредност примењива за неки конкретан процес. На пример: заокрет вентила чиме се повећава / смањује проток флуида, промена брзине покретне траке ради синхронизације процеса производње, или кретање руке робота ради вршења механичког рада.

Постоји више метода за дефазификацију. Избор методе је врло битан јер је добијена дискретна вредност најчешће крајњи ефекат резоновања система базираног на расплинутој логици. Другом речју, избором погрешне методе дефазификације може да се наруши ефикасност расплинутог резоновања. Избор оптималне (*која највише одговара*) методе дефазификације ради се у току тестирања система, на основу добијених резултујућих расплинутих скупова. У материјалу су обрађене три методе које су најчешће коришћене: средина максимума, средња и момент дефазификација.

4.2.3.1. Дефазификација средином максимума

Дефазификација средином максимума (енгл. *Mean Of Maximum – MOM Defuzzyfication*) представља најчешће коришћену методу, која истиче важност елемената резултујућег расплинутог скупа који имају максималну вредност функције припадања. Примери из добре праксе препоручују примену нормализационог модификатора на резултујући скуп, како би се боље изразили максимуми. У следећем примеру (Слика 4.18), резултујући расплинути скуп приказан је и табеларно и графички.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
m(x)	0	0	0.2	0.4	0.6	0.8	1	1	1	1	0.8	0.6	0.4	0.2	0	0	0	0	0	0.3	0.7	1	0.7	0.3	0	0	0	0	0



Слика 4.18: Пример резултујућег расплинутог скупа и вредности добијене дефазификацијом

Дефазификација средином максимума за дискретну расподелу (као што је то случај у датом примеру) рачуна се као количник суме елемената x_i које имају максималну вредност функције припадања и броја тих елемената k (Израз 4.4).

$$D_{MoM}(x) = \frac{\sum_{i=1}^{k} x_i}{k} \qquad \text{И.4.4}$$

Добијена дискретна вредност на основу података из табеле и датог израза је 10.2 и она представља резултат и коначан исход расплинутог резоновања. За разлику од резултата добијеним дефазификацијом средином максимума, просек вредности x које имају максималне вредности функција припадања дала би дискретну вредност која мање одражава природу резултујућег расплинутог скупа (14.25).

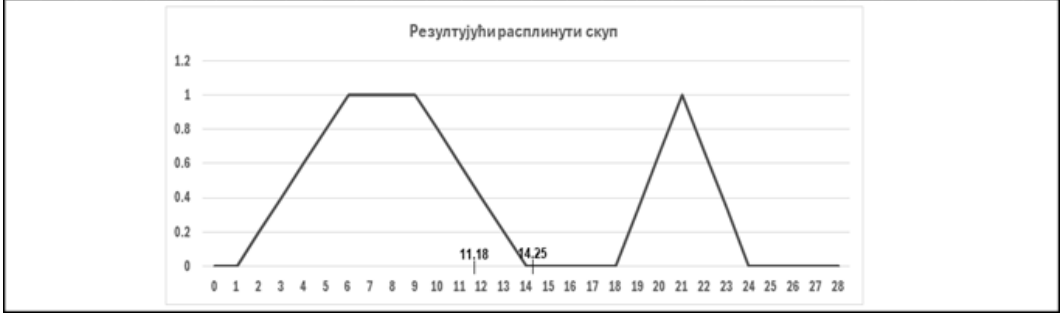
4.2.3.2. Дефазификација тежинкском средином

Дефазификација тежинкском средином (енгл. *Weighted Average Defuzzyfication*) се примењује у случајевима када резултујући расплинути скупови нису балансирани и/или нису конвексни. Овај начин дефазификације с једне стране смањује утицај доминантног сегмента (дела) резултујућег скупа, а са друге стране повећава утицај осталих сегмената (делова). Наведени утицај је промењен захваљујући коришћењу свих датих (n) вредности x_i и вредности њихових функција припадања $m(x_i)$ (Израз 4.5).

$$D_{wa}(x) = \frac{\sum_{i=1}^{n} x_i * m(x_i)}{\sum_{i=1}^{n} m(x_i)} \qquad \text{И.4.5}$$

Добијена дискретна вредност на основу података из табеле и датог израза за исти резултујући расплинути скуп (Слика 4.19) је 11.18, што је веће од вредности добијене дефазификацијом средином максимума (10.2).

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
m(x)	0	0	0.2	0.4	0.6	0.8	1	1	1	1	0.8	0.6	0.4	0.2	0	0	0	0	0	0.3	0.7	1	0.7	0.3	0	0	0	0	0



Слика 4.19: Пример резултујућег расплинутог скупа и вредности добијене дефазификацијом

Резултујућа вредност би била већа од 11.18 уколико би десна компонента резултујућег скупа била већа. На пример, ако би десна компонента уместо једне имала две вредности *xi* са вредношћу функције припадања 1, онда би резултат дефазификације био нешто већи (12.08).

4.2.3.3. Дефазификација центрима гравитације

Дефазификација центрима гравитације (енгл. *Center Of Gravity - CoG Defuzzyfication*) заснива се на декомпоновању (сегментацији) резултујућег расплинутог скупа, а затим рачунањем површина и позиција центроида за сваки сегмент резултујућег скупа. На сваки сегмент се даље примењује формула (Израз 4.6), у којој је p број сегмената, c_i центроид сегмента, а s_i представља површину сегмента. Добијена вредност представља резултат дефазификације центрима гравитације.

$$D_{CoG}(x) = \frac{\sum_{i=1}^{p} c_i * s_i}{\sum_{i=1}^{p} s_i} \qquad \text{И.4.6}$$

У претходном изразу фигуришу x вредности центроида сегмента, које се израчунавају као количници x вредности темена и броја темена сегмента (Израз 4.7).

$$c_x = \frac{\sum_{i=1}^{T} x_i}{T} \qquad \text{И.4.7}$$

У претходном изразу c_x представља вредности центроида посматраног сегмента, T број темена сегмента и x_i представља x координате темена

сегмента. На датом примеру (Слика 4.20) види се да постоји укупно 5 сегмената. Изузев 2. сегмента (који је правоугаоник), остали су правоугли троуглови.

Слика 4.20: сегментација резултујућег расплинутог скупа

На основу вредности сегмената и израза (Израз 4.7) добијене су следеће вредности центроида и површина за сегменте (Табела 4.1).

	Center of Gravity			
	x	m(x)	Centroid x	P segmenta
Segment 1	1	0	4.333333	2.5
	6	1		
Segment 2	6	1	7.5	3
	9	1		
Segment 3	9	1	12.33333	2.5
	14	0		
Segment 4	18	0	20	1.5
	21	1		
Segment 5	21	1	23	1.5
	24	0		

Табела 4.1: вредности x центроида и површина за сегменте 1-5 са слике 43

Применом добијених вредности центроида и површина сегмената резултујућег скупа у изразу за дефазификацију ($D_{CoG}(x)$, Израз 4.6) добија се вредност 11.69, што је у конкретном примеру вредност веома блиска вредности добијеној дефазификацијом тежинском средином (11.18).

Поред наведених постоје и друге методе дефазификације, развијене услед потреба у конкретним случајевима примене расплинуте логике у пословним процесима. Важна је напомена да је дефазификација у неким

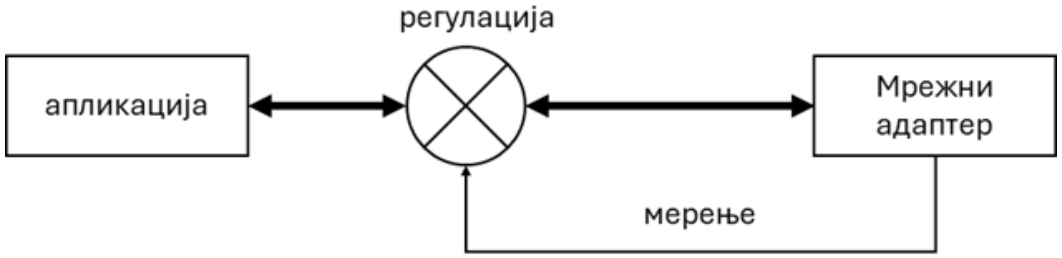
применама неопходна и код система за резоновање који користе *Takagi-Sugeno* методу. Функције у акционим деловима правила која се извршавају дају вредности за излазне величине. У неким случајевима дефинишу се расплинути скупови за излазне величине. У том случају се резултатима функција придружују и вредности функције припадања, тако да постоје сви подаци неопходни за примену неке од метода дефазификације.

4.3. Примери коришћења расплинуте логике

Расплинута логика има веома скалабилну природу. Може се примењивати у системима регулације. Рецимо, као контролери у процесима мале комплексности. Исто тако расплинута логика се може примењивати у експертским системима у циљу подршке у доношењу одлука када су подаци непотпуни и/или непрецизни. У том смислу дати су примери за оба случаја.

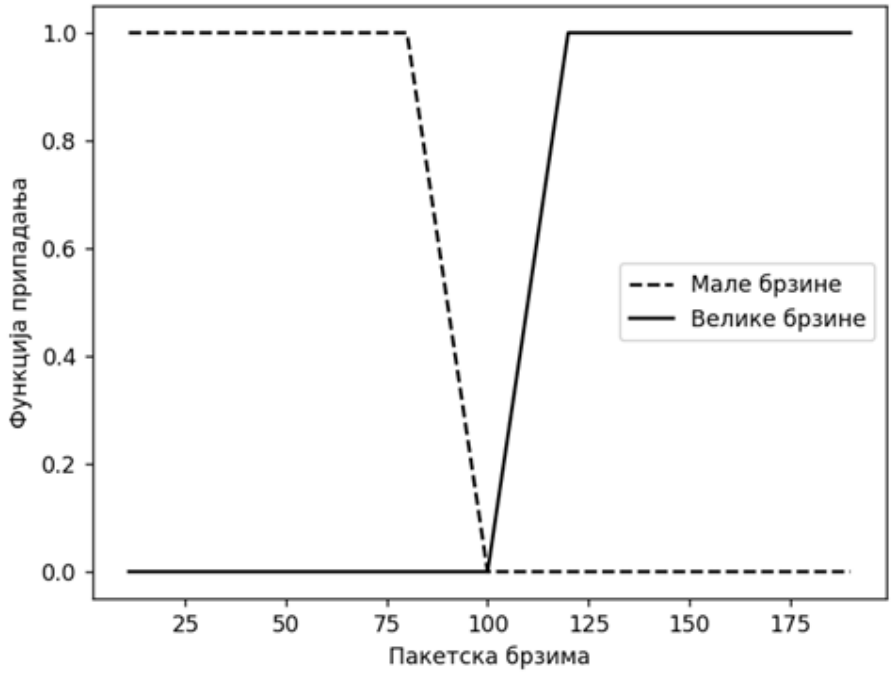
4.3.1. Пример дизајна система за регулацију пакетске брзине на бази расплинуте логике

Регулација пакетске брзине на бази расплинуте логике користи се у контексту омогућавања дељења интернет линка као заједничког ресурса мрежним апликацијама (корисницима), а у циљу одржавања квалитета линка (*QoS – Quality of Service* алгоритам). Модел система (Слика 4.21) се састоји од апликације, линка и регулатора. Поред тога постоје директне везе за размену пакета и повратна веза која омогућава регулацију. Регулатор има функцију стабилизације пакетске брзине око номиналне (пожељне) вредности тако што ће величина промене брзине да буде пропорционална разлици измерене и номиналне вредности. Другим речима, мале разлике измерене и номиналне вредности систем ће да компензује малим променама вредности брзине. За велике разлике измерене и номиналне вредности систем ће да врши велике промене пакетске брзине.



Слика 4.21: модел система за регулацију пакетске брзине

Пакетска брзина представља величину чије вредности се регулишу применом расплинуте логике. У циљу решавања проблема, над том величином направљена су два расплинута скупа – скуп *малих* и скуп *великих брзина* (Слика 4.22). У примеру је вредност 100 дата као номинална брзина. У тој тачки оба расплинута скупа имају вредност функције припадања 0 (нула). Удаљавајући се од номиналне пакетске брзине, вредност функције припадања за оба расплинута скупа расте, до брзине 80 за малу, односно брзине 120 за велику пакетску брзину. Ван тог опсега, вредност функције припадања за оба расплинута скупа је 1 (један). На основу описа проблема је очигледно да је вредност функције припадања заправо величина промене пакетске брзине у процесу њене регулације. Ако је измерена вредност пакетске брзине једнака номиналној, вредност функције припадања је 0, тако да нема промене брзине. Ако је пакетска брзина мања од 80, вредност функције припадања је 1, тако да је промена брзине позитивна и највећа могућа. Аналогно томе, ако је пакетска брзина већа од 120, вредност функције припадања је такође 1, али је тада промена брзине негативна и највећа могућа. Вредност функције припадања опада у опсегу [80,100], тако да у том случају опада и величина промене пакетске брзине. Аналогно томе, када вредност функције припадања расте у опсегу [100,120], у том случају расте и величина промене пакетске брзине.



Слика 4.22: расплинути скупови

Оба расплинута скупа су линеарна и сваки има три сегмента. Математички израз за функцију припадања за расплинут скуп малих брзина представљен је следећом илустрацијом (Слика 4.23). За брзине 100 и веће вредност функције припадања је 0, за брзине 80 и мање вредност функције припадања је 1, а у средњем сегменту, за брзине између 80 и 100 вредности функције припадања је линеарно опадајућа.

$$\mu(\textit{мале брзине}) = \begin{cases} 0 & ,x \geq 100 \\ -\frac{1}{20}x + 5 & ,80 < x < 100 \\ 1 & ,x \leq 80 \end{cases}$$

Слика 4.23: математичка репрезентација функције припадања за мале пакетске брзине

Математички израз за функцију припадања за расплинут скуп великих брзина представљен је следећом илустрацијом (Слика 4.24). За брзине 100 и мање вредност функције припадања је 0, за брзине 120 и веће вредност функције припадања је 1, а у средњем сегменту, за брзине између 100 и 120 вредности функције припадања је линеарно растућа.

$$\mu(\textit{велике брзине}) = \begin{cases} 0 & ,x \leq 100 \\ \frac{1}{20}x - 5 & ,100 < x < 120 \\ 1 & ,x \geq 120 \end{cases}$$

Слика 4.24: математичка репрезентација функције припадања за велике пакетске брзине

Овако дефинисан систем од два расплинута скупа се користи тако што се врши очитавање стварне брзине, а затим на основу њене вредности се израчунавају вредности функције припадања за скуп малих и великих брзина, које се користе у изразу за израчунавање вредности кориговане брзине (Израз 4.8).

$$v_{t+1} = v_t + \mu_m * v_t * \Delta v - \mu_v * v_t * \Delta v \qquad \text{И.4.8}$$

У изразу v_{t+1} представља вредност кориговане брзине, v_t представља вредност измерене брзине, μ_m је вредност функције припадања скупу малих брзина, Δv је корак промене брзине и μ_v је вредност функције припадања скупу великих брзина. Динамика рада система дата је у следећем примеру. Ако је измерена брзина $v_t = 110$, а корак промене брзине $\Delta v = 10$, заменом (по Израз 4.8) добија се да је вредности кориговане брзине 105:

$$v_{t+1} = 110 + 0 * 10 - 0.5 * 10 = 105$$

У следећем мерењу брзина је 105, заменом (по Израз 4.8) добија се да је вредност кориговане брзине 102.5:

$$v_{t+1} = 105 + 0 * 10 - 0.25 * 10 = 102.5$$

Промена брзине је мања него у претходном случају (опада са 110 на 105) захваљујући смањеној вредности функције припадања (опада са 105 на 102.5). Због сметњи на комуникационом линку, следећа измерена брзина је 80 и систем реагује променом понашања – уместо смањивања, повећавање брзине. Заменом (по Израз 4.8) добија се да је вредности кориговане брзине 90:

$$v_{t+1} = 80 + 1 * 10 - 0 * 10 = 90$$

У следећем мерењу брзина је 90, заменом (по Израз 4.8) добија се да је вредност кориговане брзине 95:

$$v_{t+1} = 90 + 0.5 * 10 - 0 * 10 = 95$$

Након тестирања математичког модела, врши се имплементација система у одговарајућем развојном окружењу. Представљена је имплементација у *Python* окружењу коришћењем *sckit fuzzy* библиотеке (*skfuzzy*). Најпре се дефинише промењива која ће одговарати пакетској брзини као физичкој величини од интереса (Слика 4.25). Да би се добио опсег вредности од интереса (енгл. *Universe of Discourse - UoD*), промењива *paketska_brzina* је дефинисана као низ чији су елементи целобројне вредности. За најмању брзину узета је вредност 11, за највећу 191. Иако се промењива *paketska_brzina* састоји од 180 дискретних вредности, *skfuzzy* врши интерполацију у случају да се појаве реални бројеви као улазне вредности за регулацију. У следећем кораку се дефинишу расплинути скупови за ниску и високу брзину.

Библиотека *skfuzzy* коришћена је са алијасом *fuzz*, а скупови су формирани тако што се функционалном позиву *trapmf* проследила промењива *paketska_brzina* и низ вредности које означавају пројекције темена трапезоида на апсциси. На овај начин да скуп за малу брзину (представљен промењивом *paketska_brzina_mala*) има сегмент од 11 до 80 у коме је вредности функције припадања 1 и сегмент од 80 до 100 у коме је функција припадања опадајућа од 1 до 0. За вредности промењиве *paketska_brzina* веће од 100 вредност функције припадања је 0 (као подразумевана). На сличан начин је креиран скуп за велику брзину (представљен промењивом *paketska_brzina_velika*), само што за вредности промењиве *paketska_brzina* мање од 100 вредност функције припадања је 0 (нула). Следи сегмент у коме је функције припадања растућа [100,120], а затим сегмент у коме је вредност функције припадања један [120,191].

```
paketska_brzina=np.arange(11,191,1)
paketska_brzina_mala=fuzz.trapmf(paketska_brzina, [11,11,80,100])
paketska_brzina_velika=fuzz.trapmf(paketska_brzina, [100,120,191,191])
```

Слика 4.25: дефинисање промењиве и расплинутих скупова

У следећем кораку се врши дефинисање функције за израчунавање вредности кориговане брзине (Израз 4.8). Функција *korekcija* има три

параметра: измерену брзину, вредности функције припадања скупу малих и скупу великих брзина (Слика 4.26).

```
def korekcija(brzina, mf_mala_brzina, mf_velika_brzima):
    brzina = brzina + mf_mala_brzina * 10 - mf_velika_brzima * 10
    return brzina
```

Слика 4.26: функција за израчунавање вредности кориговане брзине

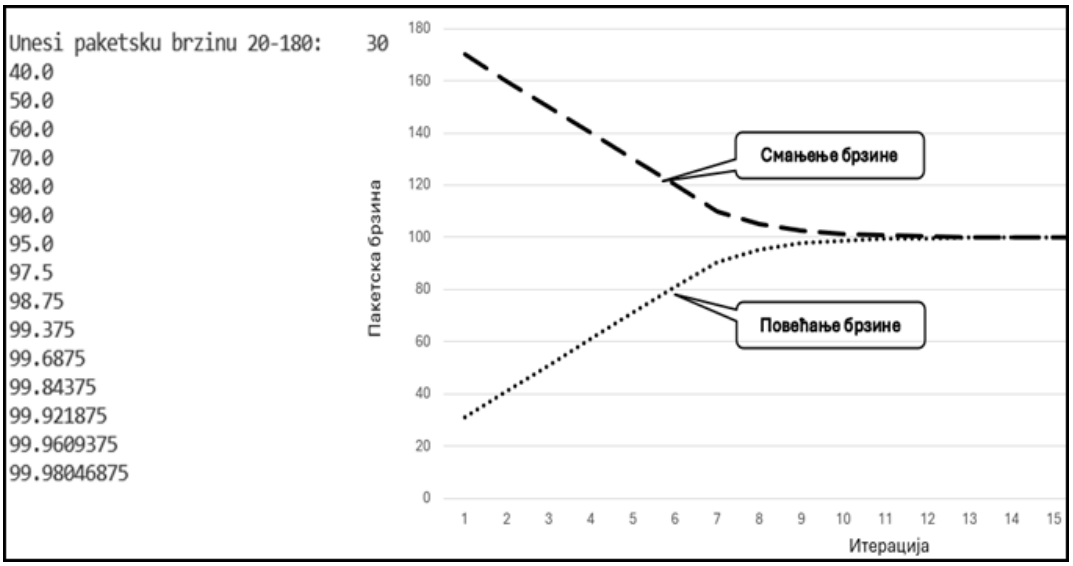
На крају имплементације, следи дефинисање функционалности за тестирање (Слика 4.27). Код омогућава кориснику да унесе произвољну пакетску брзину, а затим у петљи врши низ узастопних корекција брзина. Број итерација је ограничен, обзиром да се ради о тестирању. У првој итерацији прихвата унос у промењиву *nova_brzina*, а затим користи библиотечку функцију *interp_membership*, како би добио вредности функције припадања расплинутим скуповима *paketska_brzina_mala* и *paketska_brzina_velika* за вредност промењиве *nova_brzina*. Добијене вредности прихвата у промењиве *mf_ubrzanje* и *mf_usporenje*, које затим, заједно са брзином прослеђује функцији *korekcija* како би се добила коригована брзина.

```
unos = input("Unesi paketsku brzinu 20-180:    ")
prvi_prolaz = True
brojIteracija = 0
while(True):

    if(brojIteracija == 15):
        break
    if(prvi_prolaz):
        nova_brzina = float(unos)
    prvi_prolaz = False
    brojIteracija = brojIteracija+1
    mf_ubrzanje = fuzz.interp_membership(paketska_brzina,paketska_brzina_mala,nova_brzina)
    mf_usporenje = fuzz.interp_membership(paketska_brzina,paketska_brzina_velika,nova_brzina)
    nova_brzina = korekcija(nova_brzina,mf_ubrzanje,mf_usporenje)
    print(f'{nova_brzina}')
```

Слика 4.27: функција за тестирање рада система за корекцију пакетске брзине

Резултат рада функције за тестирање приказан је на следећој илустрацији (Слика 4.28). На левој страни је приказ интеракције корисника и окружења, а на десној страни су визуализовани подаци у случајевима када су кориснички уноси били 30 и 170 (за пакетску брзину).



Слика 4.28: тестирање система

Након тестирања, систем се уводи у експлоатацију. Најчешћи случај је да систем улазне податке прима периодично из удаљеног извора, а корективе вредности шаље на посебан склоп, или уређај за контролу брзине.

4.3.2. Систем за подршку у одлучивању на бази расплинуте логике

Поред контролно – регулационе примене представљене на претходном примеру, расплинута логика може се користити и у системима за подршку у одлучивању. Следи приказ дизајна и имплементације система за процену исплативости куповине. На бази два улазна критеријума (*цена* и *квалитет* робе), даје препоруку потенцијалном купцу у којој мери је исплатива куповина. Као и у претходном случају, за имплементацију је коришћена Python *sckit fuzzy* библиотека (*skfuzzy*). Поред тога, за креирање структура података коришћена је библиотека *numpy*, док је за приказ дијаграма (приказ расплинутих скупова) коришћена библиотека *matplotlib.pyplot* (Слика 4.29).

```
import skfuzzy as fuzz
import numpy as np
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt
```

Слика 4.29: Python библиотеке коришћене у примеру

Као у претходном примеру, најпре се дефинишу величине *цена, квалитет* и *исплативост*. Ове величине су представљене независним промењивама *цена* и *квалитет* робе, и зависном промењивом *исплативост* куповине. Све величине су представљене неименованим вредностима у опсегу од 0 до 10. У конкретној имплементацији за све три величине су конструисани низови реалних бројева од 100 елемената (Слика 4.30).

```
kvalitet_robe=np.arange(0,10.1,0.1)
cena_robe=np.arange(0,10.1,0.1)
isplativost=np.arange(0,10.1,0.1)
```

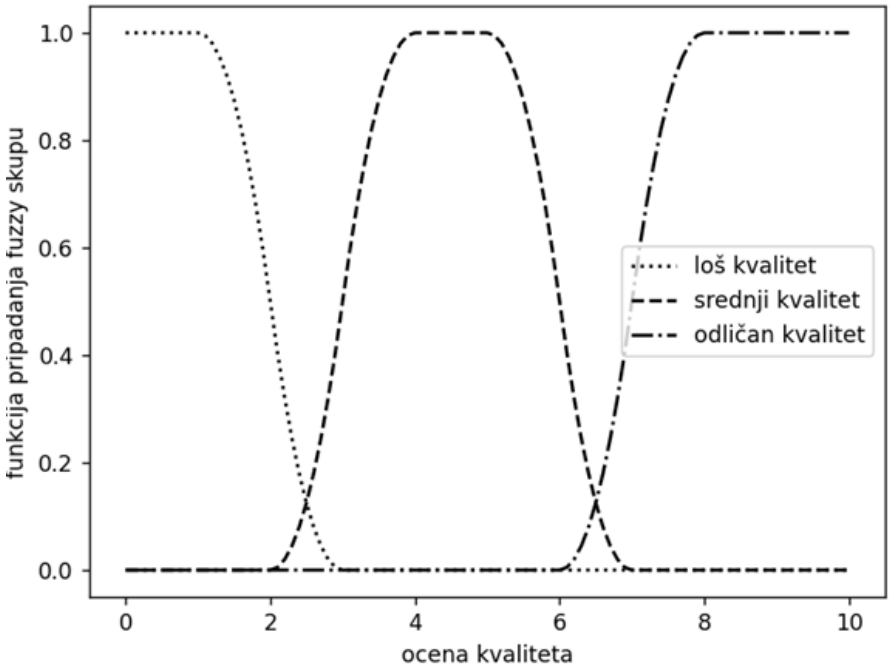
Слика 4.30: Дефинисање расплинутих промењивих

Тако је *квалитет* робе представљен помоћу три расплинута скупа – *лош, средњи* и *одличан* (Слика 4.31). Скупови су дефинисани помоћу предефинисаних типова из *skfuzzy* библиотеке, а то су *z, s* и *pi* скупови (објашњени су у секцији 4.1.2).

```
srednji_kvalitet=fuzz.pimf(kvalitet_robe,2,4,5,7)
los_kvalitet=fuzz.zmf(kvalitet_robe, 1, 3)
odlican_kvalitet=fuzz.smf(kvalitet_robe, 6, 8)
```

Слика 4.31: Расплинути скупови за квалитет робе

Изглед три расплинута скупа дефинисана за *квалитет* робе представљен је на следећем дијаграму (Слика 4.32).



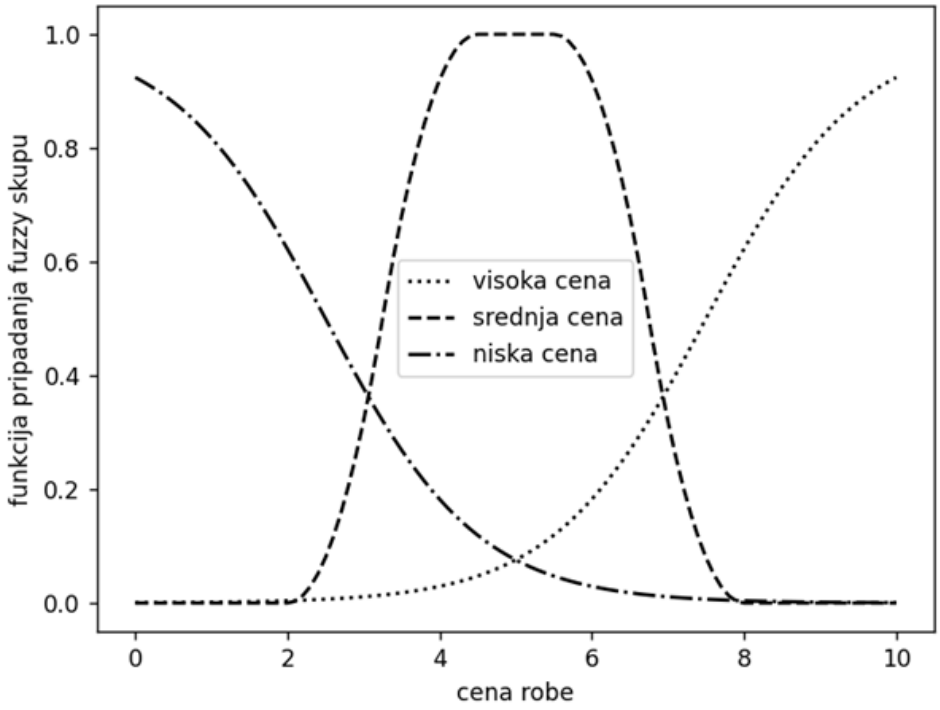
Слика 4.32: Изглед расплинутих скупова за квалитет робе

Варијабла *цена* робе представљена је такође помоћу три расплинута скупа – *лош, средњи* и *одличан* (Слика 4.33). Скупови су дефинисани помоћу предефинисаних типова из *skfuzzy* библиотеке, а то су *pi* и сигмоидни (*sigmf*) скупови. Као код *квалитета* робе, *pi* функција је искоришћена за дефинисање средњег расплинутог скупа. Сигмоидна функција је искоришћена за оба преостала расплинута скупа. Трећи параметар сигмоидне функције омогућава да се увођењем негативне вредности сигмоида „окрене“ према координатном почетку.

```
visoka_cena=fuzz.sigmf(cena_robe, 7.5, 1.0)
srednja_cena=fuzz.pimf(cena_robe, 2,4.5,5.5,8)
niska_cena=fuzz.sigmf(cena_robe, 2.5, -1.0)
```

Слика 4.33: Расплинути скупови за цену робе

Изглед три расплинута скупа дефинисана за промењиву *цена робе* представљен је на следећем дијаграму (Слика 4.34).



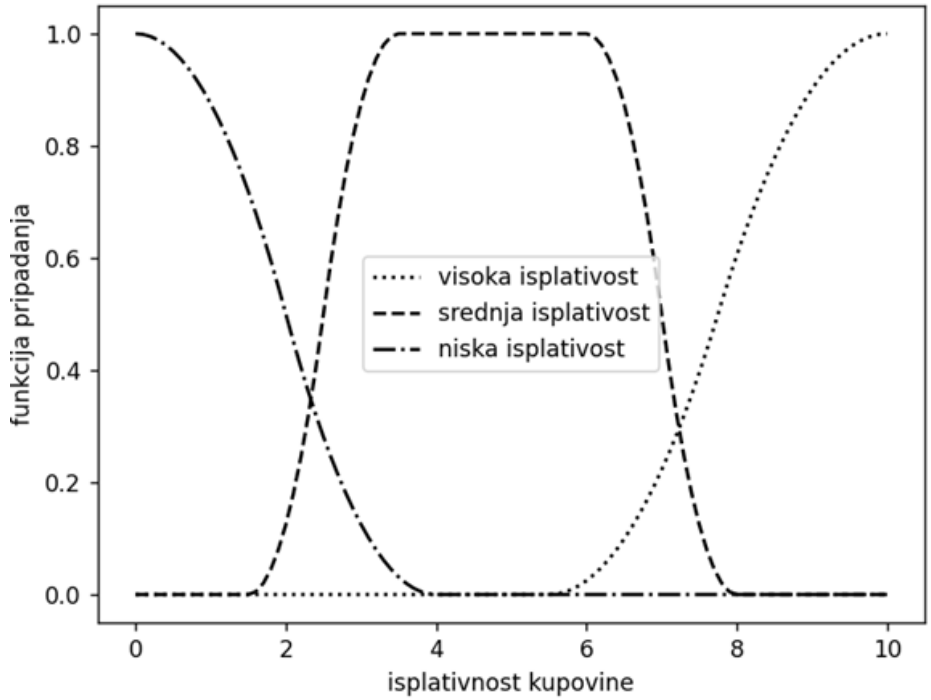
Слика 4.34: Изглед расплинутих скупова за цену робе

Варијабла *исплативост* куповине представљена је такође помоћу три расплинута скупа – *ниска, средња* и *висока* (Слика 4.35). Као и код промењиве *квалитет*, скупови су дефинисани помоћу предефинисаних типова из *skfuzzy* библиотеке, а то су *z, s* и *pi* скупови (објашњени су у секцији 4.1.2).

```
srednja_isplativost=fuzz.pimf(isplativost,1.5,3.5,6,8)
niska_isplativost=fuzz.zmf(isplativost, 0, 4)
visoka_isplativost=fuzz.smf(isplativost, 5.5, 10)
```

Слика 4.35: Расплинути скупови за исплативост куповине

Изглед три расплинута скупа дефинисана за промењиву *исплативост* куповине представљен је на следећем дијаграму (Слика 4.36).



Слика 4.36: Изглед расплинутих скупова за исплативост куповине

Након дефинисања промењивих и њима припадајућих расплинутих скупова, следећи корак је креирање базе знања. База знања код система за подршку у одлучивању на бази расплинуте логике се такође састоји од правила. Међутим, у премисама правила налазе се расплинути скупови независних промењивих, док су у последицама правила расплинути скупови зависних промењивих. Да би се наведено омогућило, искоришћени су предефинисани типови модула *control* из *skfuzzy* библиотеке: *Antecedent* за дефинисање садржаја премисе правила и *Consequent* за дефинисање садржаја последичног дела правила. Најпре се врши креирање промењивих које ће представљати премису и последицу правила (Слика 4.37). Конструктори класа *Antecedent* и *Consequent* садрже расплинуте промењиве и њихове текстуалне ознаке (лабеле). На пример, у премисама правила наћи ће се вредности обе расплинуте промењиве *квалитет* и *цена* робе, представљене промењивама *akvalitet_robe* и *acena_robe*. У последици правила биће укључена само расплинута промењива *isplativost*, представљена промењивом *cisplativost_kupovine*.

```
akvalitet_robe = ctrl.Antecedent(kvalitet_robe,"Kvalitet robe")
acena_robe = ctrl.Antecedent(cena_robe,"Cena robe")
cisplativost_kupovine = ctrl.Consequent(isplativost,"Isplativost kupovine")
```

Слика 4.37: Дефинисање садржаја премисе и последице правила

У следећем кораку се у конструисане промењиве додају расплинути скупови (Слика 4.38). Промењиве *akvalitet_robe, acena_robe* и *cisplativost_kupovine* омогућавају формирање асоцијативних низова код којих се расплинути скупови додају као елементи тих низова коришћењем ознала (лабела) уместо индекса. На примеру, расплинути скуп *los_kvalitet* је додат као елементи премисе *akvalitet_robe* који има лабелу „*Лош kvalitet*“.

```
akvalitet_robe['Loš kvalitet'] = los_kvalitet
akvalitet_robe['Dobar kvalitet'] = srednji_kvalitet
akvalitet_robe['Odličan kvalitet'] = odlican_kvalitet

acena_robe['Niska cena'] = niska_cena
acena_robe['Visoka cena'] = visoka_cena
acena_robe['Srednja cena'] = srednja_cena

cisplativost_kupovine['Niska isplativost'] = niska_isplativost
cisplativost_kupovine['Srednja isplativost'] = srednja_isplativost
cisplativost_kupovine['Visoka isplativost'] = visoka_isplativost
```

Слика 4.38: додавање расплинутих скупова у промењиве премиса и последица правила

Након дефинисања промењивих за премисе и консеквенце правила, следи дефинисање правила. Број правила који мора да се дефинише зависи од броја расплинутих промењивих у премиси и од броја расплинутих скупова које те промењиве садрже. Практично се ради о производу који се може представити следећим изразом (Израз 4.9), у коме N_r представља број расплинутих правила за n промењивих укључених у премисе правила, док F_i представља број расплинутих скупова дефинисаних за расплинуту промењиву i.

$$N_r = \prod_{i=1}^{n} F_i \qquad \text{И.4.9}$$

Како је у датом примеру предвиђено да премисе имају 2 промењиве са по 3 расплинута скупа, инжењер знања мора да дефинише 3*3=9 расплинутих правила. На следећем фрагменту *Python* кода представљена су три

расплинута правила (од девет). За дефинисање расплинутих правила користи се предефинисани тип *Rule* (енгл. правило) модула *control* из *skfuzzy* библиотеке (Слика 4.39). Конструктор типа *Rule* има два параметра: премису и последицу. Први параметар садржи вредности промењивих премисе *akvalitet_robe, acena_robe*, повезане конјункцијом. У линији 69 то су расплинути скупови за квалитет робе *akvalitet_robe['Loš kvalitet']* и за цену робе *acena_robe['Visoka cena']*. Други параметар (линија 70) садржи вредност зависне промењиве (последице) - *cisplativost_kupovine* која је представљена расплинутим скупом за исплативост куповине *cisplativost_kupovine['Niska isplativost']*.

```
69  pravilo_1 = ctrl.Rule(akvalitet_robe['Loš kvalitet'] & acena_robe['Visoka cena'],
70                        cisplativost_kupovine['Niska isplativost'])
71  pravilo_2 = ctrl.Rule(akvalitet_robe['Dobar kvalitet'] & acena_robe['Visoka cena'],
72                        cisplativost_kupovine['Niska isplativost'])
73  pravilo_3 = ctrl.Rule(akvalitet_robe['Odličan kvalitet'] & acena_robe['Visoka cena'],
74                        cisplativost_kupovine['Srednja isplativost'])
```

Слика 4.39: дефинисање расплинутих правила

Након дефинисања неопходних правила, следи успостављање експертског система - механизма за закључивање за бази расплинуте логике. За ту сврху искоришћена је класа *ControlSystem* уграђена у модул *control* из *skfuzzy* библиотеке (Слика 4.40). Њеном конструктору се прослеђује листа претходно дефинисаних расплинутих правила.

```
isplativost_ctrl = ctrl.ControlSystem([pravilo_1,pravilo_2,pravilo_3,
                                       pravilo_4,pravilo_5,pravilo_6,
                                       pravilo_7,pravilo_8,pravilo_9])
```

Слика 4.40: додавање правила у експертски систем на бази расплинутог резоновања

За тестирање експертског система на бази расплинуте логике, представљеног промењивом *isplativost_ctrl,* користи се симулатор представљен класом *ControlSystemSimulation* уграђен у модул *control* из *skfuzzy* библиотеке (Слика 4.41). Конструктору класе се прослеђује промењива *isplativost_ctrl* и систем може да се користи.

```
isplativost_simulacija = ctrl.ControlSystemSimulation(isplativost_ctrl)
```

Слика 4.41: симулатор за тестирање експертског система на бази расплинутог резоновања

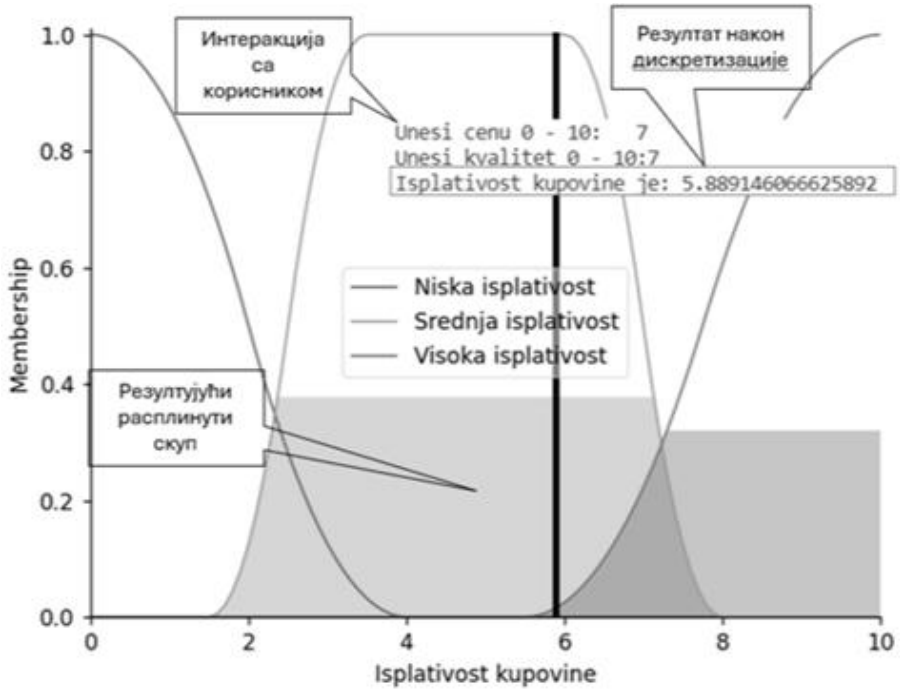
Следећи кодни фрагмент приказује начин коришћења система посредством симулатора (Слика 4.42). Прве четири наредбе представљају имплементацију интеракције са корисником, коме се постављају захтеви и прихватају кориснички уноси који се конвертују у реалане бројеве. Након тога, наредбом *compute* извршава се закључивање, а последње две наредбе служе за преузимање и приказ резултата.

```
unos_cena = input("Unesi cenu 0 - 10:   ")
isplativost_simulacija.input["Cena robe"] = float(unos_cena)
unos_kvalitet = input("Unesi kvalitet 0 - 10:")
isplativost_simulacija.input["Kvalitet robe"] = float(unos_kvalitet)

isplativost_simulacija.compute()
print(f"Isplativost kupovine je: {isplativost_simulacija.output["Isplativost kupovine"]}")
cisplativost_kupovine.view(sim = isplativost_simulacija)
```

Слика 4.42: код за коришћење система посредством симулатора

Приказ рада система и визуелизација података представљени су на следећој илустрацији (Слика 4.43). Након интеракције с корисником и завршетка процеса закључивања, као исход добија се резултујући расплинути скуп (на слици приказан осенчено). Применом одговарајућег алгоритма овај резултат се дискретизује у једну нумеричку вредност (тзв. *дефазификација*) која се захвата *output* методом примењеног симулатора. Међутим, за комплетну интерпретацију резултата закључивања често је потребно и сагледавање резултујућег расплинутог скупа. У представљеном примеру кориснички уноси су за цену – 7, а што би представљало умерено високу цену и за квалитет – 7, што би представљало одличан квалитет робе. Добијена је излазна вредност ~5.89, што би представљало незнатно повишену средњу исплативост куповине.



Слика 4.43: приказ интеркације са корисником и резултата резоновања

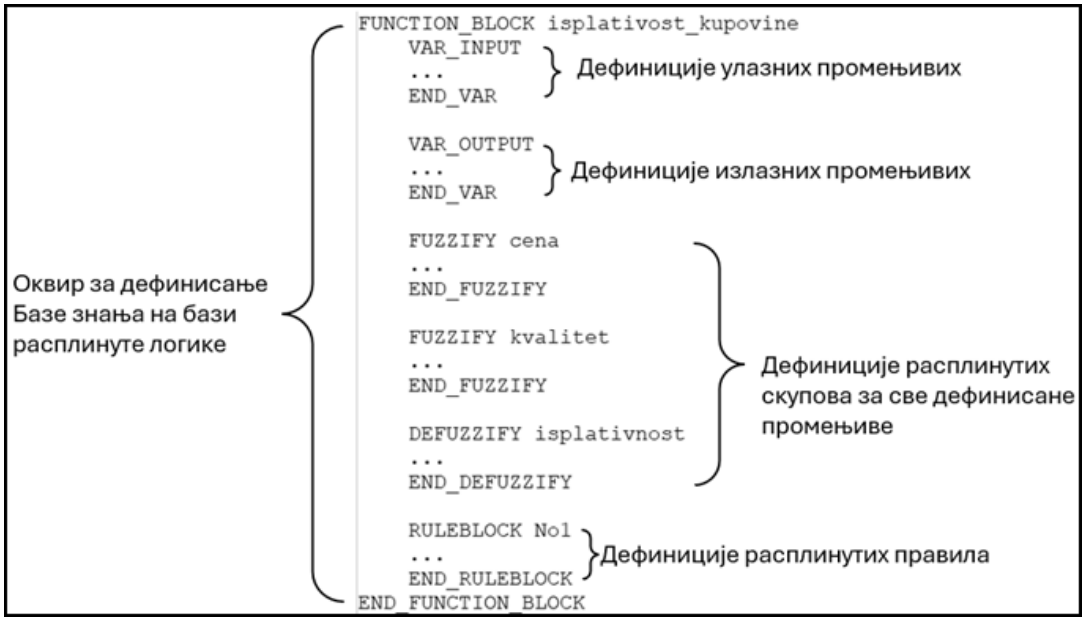
4.4. Пример коришћења система на бази расплинуте логике у Јава апликацији

Постоје многе бесплатне библиотеке које омогућавају коришћење система на бази расплинуте логике у другим апликацијама. У следећем садржају је представљена Јава апликација која користи библиотеку *jFuzzyLogic* ради имплементације система за подршку у одлучивању у вези процене исплативости куповине. Искоришћен је пример из претходне секције (Секција 4.3.2) у коме се на бази два улазна критеријума (*цена* и *квалитет* робе), даје препорука потенцијалном купцу у којој мери је исплатива куповина.

4.4.1. FCL скрипт језик

Слично као код *CLIPS* скрипт језика (Секција 3.3.1), за дизајн базе знања користи се FCL (од енгл. *Fuzzy Control Language*) скрипт језик. База знања се чува у датотекама са *fcl* екстензијом и има врло униформну структуру (Слика 4.44), која се састоји од једног или више функционалних блокова који имају назив. У примеру је назив функционалног блока *isplativost_kupovine*. Унутар

функционалног блока најпре се дефинишу расплинуте промењиве (*VAR_INPUT* и *VAR_OUTPUT* блокови), затим расплинути скупови за све промењиве (*FUZZIFY* блокови). Дефиниција и начин дискретизације резултујућег расплинутог скупа се налази у *DEFUZZIFY* блоку. На крају следи један, или више блокова у којима се дефинишу правила на бази расплинуте логике.



Слика 4.44: Структура базе знања на бази расплинуте логике

Дефиниција расплинутих промењивих садржи назив и тип промењиве (Слика 4.45). На представљеном примеру дефинисане су две независне (улазне) промењиве *cena* и *kvalitet* и зависна (излазна) промењива *isplativost*. Све три промењиве су типа реални бројеви.

```
VAR_INPUT
    cena : REAL;
    kvalitet : REAL;
END_VAR

VAR_OUTPUT
    isplativnost : REAL;
END_VAR
```

Слика 4.45: Дефинисање расплинутих промењивих

Дефиниција расплинутих скупова врше се у *FUZZIFY* и *DEFUZZIFY* блоковима, чији називи указују којој промењивој припадају расплинути

скупови унутар блока (Слика 4.46). За независне (улазне) промењиве користе се *FUZZIFY* блокови. Поред назива промењиве којој припадају, садрже под-блокове означене речју TERM. Сваки TERM под-блок садржи назив термина (на пример *niska_cena*), тип расплинутог скупа (на пример *SIGM*) и неопходне димензије за формирање расплинутог скупа. Димензије се разликују по броју и предзнаку у зависности од типа расплинутог скупа. Тип *SIGM* је сигмоидна функција чија је орјентација и стрмина одређена предзнаком и вредношћу првог броја, а централна тачка вредношћу другог броја. Тип *GBELL* је Гаусова функција чији параметри одређују стрмину, ширину расплинутог скупа и његову централну тачку. За зависну (излазну) промењиву користи се *DEFUZZIFY* блок, који поред TERM под-блока садржи и начин дискретизације (под-блок *METHOD*). У примеру је коришћена *COG* (енгл. *Center of Gravity*) дискретизација. На крају, *DEFUZZIFY* блок садржи и подразумевану вредност у случају да се при резоновању не изврши нити једно правило (под-блок *DEFAULT*). У примеру је 0 искоришћена као подразумевана вредност.

```
FUZZIFY cena
    TERM niska_cena := SIGM -4.0 3.0;
    TERM srednja_cena := GBELL 5.5 2.0 4.0;
    TERM visoka_cena := SIGM 3.0 8.5;
END_FUZZIFY

FUZZIFY kvalitet
    TERM los_kvalitet := SIGM -3.0 2.5;
    TERM odlican_kvalitet := SIGM 3.0 8.5;
    TERM srednji_kvalitet := GBELL 5.5 2.0 4.0;
END_FUZZIFY

DEFUZZIFY isplativnost
    TERM niska_isplativnost := SIGM -3.0 3.0;
    TERM srednja_isplativnost := GBELL 5.0 2.0 2.0;
    TERM visoka_isplativnost := SIGM 3.0 8.5;
    METHOD : COG;
    DEFAULT := 0.0;
END_DEFUZZIFY
```

Слика 4.46: Дефинисање расплинутих скупова

За дефинисање расплинутих правила користе се *RULEBLOCK* блокови (Слика 4.47). У примеру је дефинисан један такав блок назван *Br1*. Овај блок се састоји од различитих врста под-блокова. Прва три под-блока намењена су спецификацији рада механизма за резоновање. Под-блок *ACT* (од енгл. *Activation*) намењен је за дефинисање начина активације и извршења

правила. Под-блок *ACCU* (од енгл. *Accumulation*) намењен је за дефинисање начина акумулације резултата – исхода извршења више расплинутих правила. Под-блок *AND* намењен је за дефинисање логичке конјункције, за коју се користи функција минимума. Имплицитно је да ће се у том случају за логичку дисјункцију користити функција максимума.

```
RULEBLOCK Br1
    ACT : MIN;
    ACCU : MAX;
    AND : MIN;
    RULE 1 : IF (cena IS niska_cena) AND (kvalitet IS los_kvalitet)
        THEN isplativnost IS niska_isplativnost;
    RULE 2 : IF (cena IS niska_cena) AND (kvalitet IS srednji_kvalitet)
        THEN isplativnost IS visoka_isplativnost;
    .....
    RULE 9 : IF (cena IS visoka_cena) AND (kvalitet IS odlican_kvalitet)
        THEN isplativnost IS srednja_isplativnost;
END_RULEBLOCK
```

Слика 4.47: Дефинисање расплинутих правила

Затим следе *RULE* под-блокови који садрже дефиниције расплинутих правила. На горњем примеру правила у премиси имају испитивање конјункција вредности улазних промењивих (*cena* и *kvalitet*), а у акционом делу додељују вредност излазној промењивој *isplativnost*. За број правила које треба да се дефинише важи иста формула као и у примеру у секцији 4.3.2 (Израз 4.9). Обзиром да обе улазне промењиве имају по три расплинута скупа, онда је потребан број правила девет. У случају да има мање дефинисаних правила, односно да излазне вредности нису дефинисане за све случајеве употребе, као резултат резоновања појавиће се подразумевана вредност, која је у представљеном примеру једнака нули.

4.4.2. Примена базе знања расплинуте логике у Јава апликацији коришћењем *jFuzzyLogic* окружења

Примена расплинуте логике у Јава апликацији најпре укључује учитавање базе знања из *fcl* скрипт датотеке (Слика 4.48). За ту намену користи се статичка метода *load* класе *FIS*. Класа *FIS* представља фасадну класу система за резоновање на бази расплинуте логике. Њена метода *load* поред учитавања скрипта, конструише сопствену инстанцу у радној меморији. У примеру то је објекат *fuzzy_sistem*. Овај објекат преузима сваку даљу комуникацију са окружењем, а у конкретном случају са Јава апликацијом. Унос улазних вредности врши се методом *setVariable*, којој се

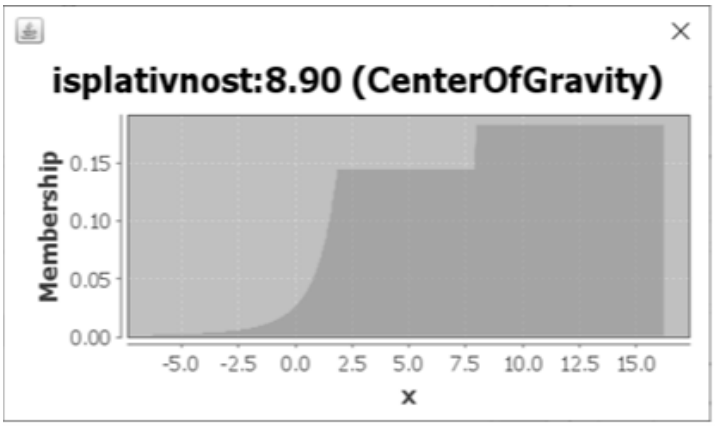
као параметри прослеђују назив улазне промењиве и њена вредност. У примеру су постављене за цену вредност 3 и за квалитет вредност 8 (линије 23 и 24).

```
13  nazivFajla += "isplativost_kupovine.fcl";
14  FIS fuzzy_sistem = FIS.load(fileName: nazivFajla,verbose: true);
15  if( fuzzy_sistem == null ) {
16      System.err.println("Fajl se ne može učitati: '" + nazivFajla + "'");
17      return;
18  }
19  FunctionBlock functionBlock =
20          fuzzy_sistem.getFunctionBlock(name: "isplativost_kupovine");
21  JFuzzyChart.get().chart(fb: functionBlock);
22
23  fuzzy_sistem.setVariable(varName: "cena", value: 3);
24  fuzzy_sistem.setVariable(varName: "kvalitet", value: 8);
25
26  fuzzy_sistem.evaluate();
27
28  Variable isplativost = functionBlock.getVariable(name: "isplativnost");
29  JFuzzyChart.get().chart(var:isplativost,
30          defuzzifier: isplativost.getDefuzzifier(), showIt:true);
```

Слика 4.48: Примена базе знања расплинуте логике у Јава апликацији

Извршење резоновања се покреће методом *evaluate* (линија 26). Узимање резултата се врши методом *getVariable*, којој се као параметар прослеђује назив излазне промењиве (линија 28). Преостале наредбе у коду намењене су приказу расплинутих скупова промењивих и резултата резоновања (линије 21 и 29). За ту сврху коришћена је класа *FunctionBlock* која енкапсулира учитан скрипт базе знања. Обзиорм да је у систем за резоновање могуће истовремено учитавање више функционалних блокова, одређени блок се захвата на објекту система за резоновање (*fuzzy_sistem*), позивом функције *getFunctionBlock* којој се прослеђује назив блока (линије 19 и 20).

Као исход резоновања, добијен је резултујући расплинути скуп промењиве *isplativnost* (Слика 4.49). Иако је скуп у великом распону вредности по *x* оси, вредности функције припадања су мање од 0.2, а вредност добијена дискретизацијом је 8.9, што указује на високу исплативост куповине за дате вредности цене и квалитета.



Слика 4.49: Резултат резоновања приказан у jFuzzyLogic окружењу

4.5. Закључак о интелигентним системима на бази расплинуте логике

Расплинута логика је омогућила да интелигентни системи могу да резонују на основу нејасних (не-егзактних) података. Системи расплинуте логике имају широку примену како у контроли и регулацији једноставних процеса, тако и у сложеним системима за закључивање са базама знања, који су имплементирани као системи продукционих правила. Системи на бази расплинуте логике су унели додатни квалитет у машинско резоновање, које је постало још приближније доношењу закључака од стране људи, обзиром на то да људи у бројним животним ситуацијама морају да доносе одлуке на бази делимично познатих и не-егзактних информација.

4.6. Питања и задаци

1. Какав је однос промењиве и расплинутог скупа у системима расплинуте логике?

2. Које су врсте расплинутих скупова?

3. Која је разлика између извршавања правила у системима за дискретно резоновање (експертним системима) и системима са правилима на бази расплинуте логике?

4. Која је намена процеса дискретизације (*defuzzification*) у резоновању на бази расплинуте логике?

5. Koja je razlika između *Mamdani* и *Takagi-Sugeno* методе резоновања на бази расплинуте логике?

6. Чему служи FCL скрипт језик?

7. Који су елементи структуре FCL скрипте?

8. Које су врсте дискретизације (*defuzzification*)?

9. Ако у бази знања постоје 2 промењиве са по 3 расплинута скупа и 1 промењива са 2 расплинута скупа, колики је број расплинутих правила неопходан за рад система за резоновање на бази расплинуте логике?

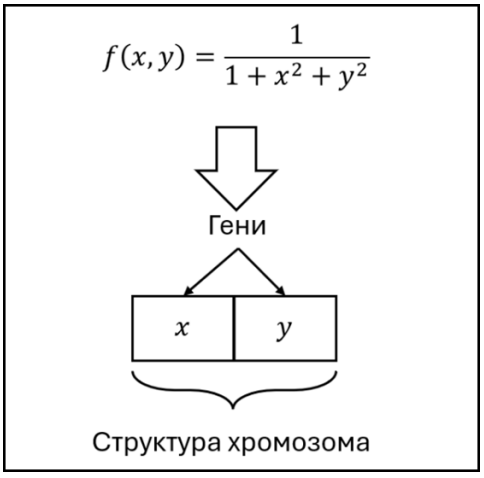
10. Које су разлике између логичких оператора дискретне и расплинуте логике?

5. Генетски алгоритам

У вештачкој интелигенцији генетски алгоритам (у даљем тексту ГА) је техника претраживања која користи хеуристику за налажење тачног или приближног (апроксимативног) / прихватљивог решења, односно резултата. ГА је врста тзв. еволуционих алгоритама, у којима се користе технике – генетички оператори као што су наслеђивање, селекција, мутација и укрштање. Основни принцип ГА је да се, уместо побољшавања (оптимизације) једног решења, ради са више потенцијалних решења. Концепт хромозома се уводи као структура података која представља потенцијално решење. Језиком програмирања, хромозом је изведен тип података који садржи податке названа генима.

ГА користи популацију хромозома као скуп потенцијалних решења. У контексту генетике, фактори (својства) који утичу на решење представљени су генима. Хромозоми садрже гене, који практично представљају варијабле (параметре) битне за решавање проблема.

Метода за решавање проблема коришћењем ГА најпре захтева одређивање (дефинисање) параметара релевантних за проблем. Параметри су обично основног типа (стрингови, цели или реални бројеви). У другом кораку, од уређених скупова тих параметара (н-торке) се конструишу хромозоми. На пример, ако је задатак да се генетским алгоритмом максимизира вредност функције $f(x,y) = 1/(1+x^2+y^2)$ (Слика 5.1) представљен је хромозом који садржи два гена, тј. два параметра означена као x и y.



Слика 5.1: Приказ хромозома који има два гена

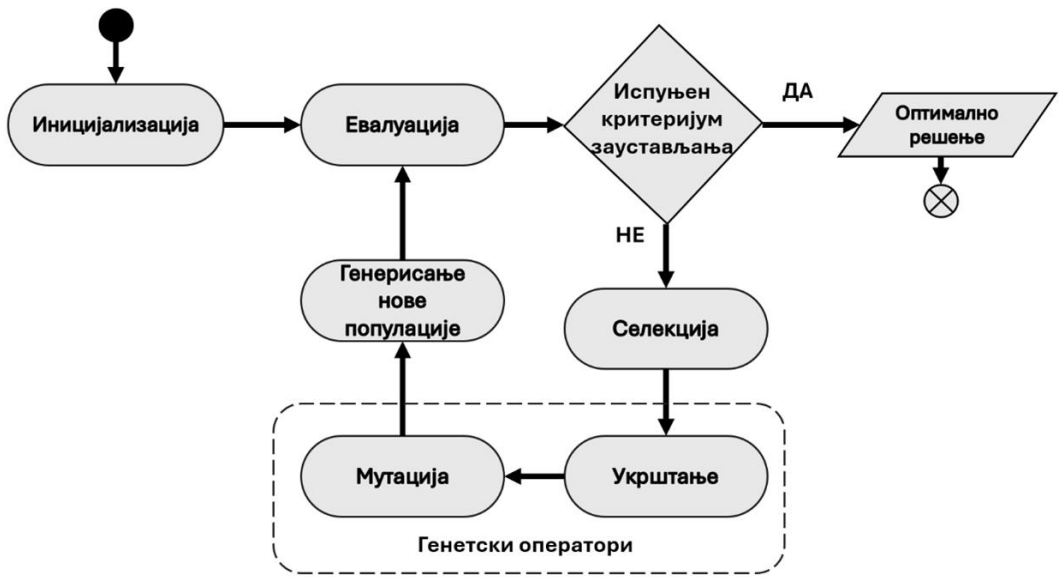
Начин функционисања ГА формализован је још 70-их година прошлог века и описан је такозваном теоремом Холандове шеме (по америчком научнику Џону Холанду). Ова теорема уводи функцију прилагођености (*fitness*).

Генетички оператори се примењују на изабране хромозоме из популације што резултира појављивање бољих/погоднијих хромозома. У овој активности користи се теорема Холандове шеме.

5.1. Фазе генетског алгоритма

Генетски алгоритам има четири фазе од којих се три понављају кроз еволутивни процес. То су иницијализација, евалуација, селекција и примена генетских оператора (Слика 5.2). Сам еволутивни процес је промена (унапређивање) популације хромозома из генерације у генерацију.

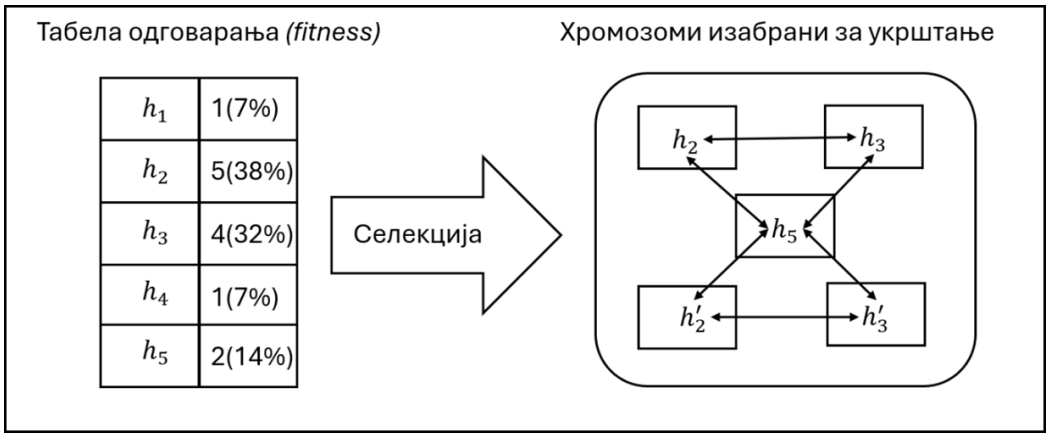
Иницијализација, представља генерисање иницијалне популације хромозома. Величина популације зависи од домена проблема (уобичајено реда од 10^2 до 10^3). Популација практично представља простор претраживања. Иницијална популација хромозома (кандидата решења) може се генерисати генератором случајних бројева. Иницијализација се може извршити коришћењем већ познатих хромозома. ГА ће бити ефикаснији уколико популација садржи различитије хромозоме.



Слика 5.2: Фазе генетског алгоритма

Евалуација представља процену појединачних хромозома у популацији, односно процену у којој мери сваки од њих одговара као решење проблема. За сваки хромозом, врши се провера потенцијалног решења проблема (резултата функције) са датим вредностима гена тог хромозома – практично вредности параметара.

Селекција представља избор одређеног броја хромозома из популације како би се на њима извршили генетичке операције и добили нови хромозоми за следећу генерацију. На следећем примеру (Слика 5.3) приказана је селекција над популацијом од пет хромозома означених као h_1, h_2 ,..., h_5. На левој страни је табела у којој су дате вредности функције прилагођености (у даљем тексту *fitness* функција) за поједине хромозоме. На десној страни представљени су хромозоми одабрани за формирање популације у следећој генерацији. Очигледно је да су одабрани хромозоми са већом вредношћу ове функције - h_2 , h_3 и h_5, док су хромозоми h_1 и h_4 одбачени. Како популација мора да одржи константан број хромозома, што је у овом примеру пет, онда су хромозоми који највише одговарају - h_2 и h_3 практично *клонирани* h_2' и h_3', односно по два пута изабрани како би се попунио број од пет хромозома у популацији. Циљ клонирања је да би се омогућио генетски оператор укрштања.

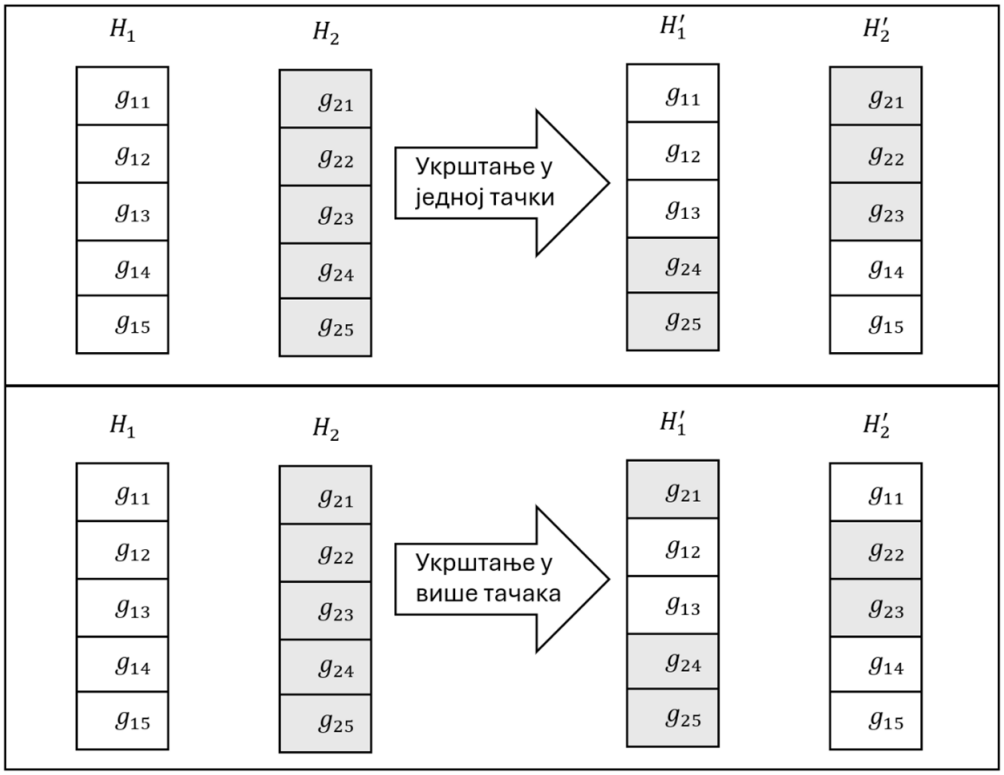


Слика 5.3: Селекција хромозома

Обзиром да избор хромозома директно утиче на резултат ГА, за селекцију се каже да је најзначајнија фаза ГА.

Примена генетских оператора на хромозомима изабраним у фази селекције је последња фаза једне итерације. Постоје две врсте генетских

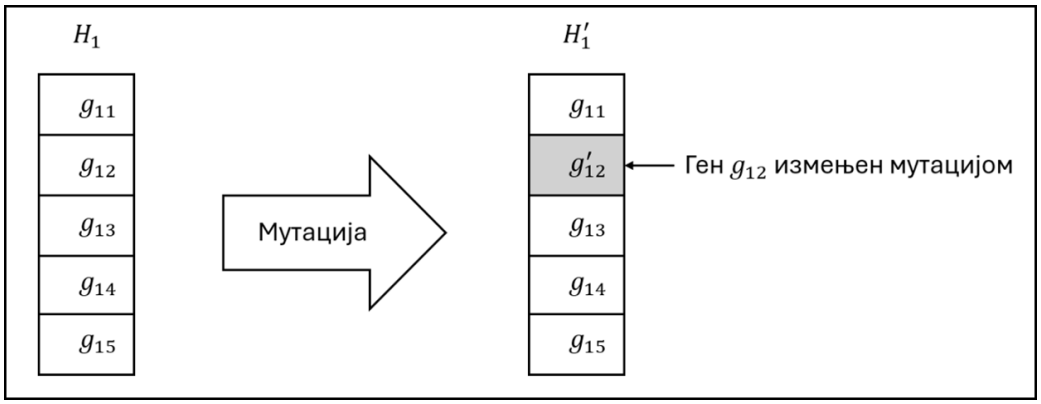
оператора: укрштање (*crossover*) и мутација. **Укрштање** је основни генетски оператор којим парови хромозома у истој популацији размењују гене. Начелно (према Холандовој теореми) овај процес из генерације у генерацију доводи до повећања просечне вредности *fitness* функције, односно до напретка у решавању проблема. Постоје две врсте укрштања: у једној тачки (*single point crossover*) и у више тачака (*multiple point crossover*) (Слика 5.4). Укрштање у једној тачки је случај када се гени у сваком од хромозома за укрштање поделе у две секвенцијалне групе гена (гени се налазе један до другог). Након тога два хромозома који се укрштају размењују секвенцијалне групе гена.



Слика 5.4: Укрштање хромозома

Код укрштања у више тачака, гени у сваком од хромозома за укрштање се деле у три, или више секвенцијалних група гена, а затим их хромозоми размењују међу собом. Укрштање не уноси нови генетски материјал, али се у популацији појављују нови хромозоми (деца) као нови кандидати за решења.

С друге стране, **мутација** представља оператор који може и да угрози функционисање ГА. Ова операција се уводи само у специфичним случајевима. Један од њих се дешава када није добијено прихватљиво решење, али нема више ни напретка из генерације у генерацију у повећању просечне вредности *fitness* функције (познат још и као проблем локалног максимума). То значи да су могућности укрштања гена међу хромозома у истој популацији исцрпљене и неопходно је применити мутацију. Овај оператор уводи промену вредности изабраног гена (или више гена) у једном, или више хромозома, тако да се практично тиме уводи нови генетски материјал за рад ГА (Слика 5.5). Ова промена може да доведе до побољшања рада ГА, али може и да га погорша. Да би били сигурни да неће доћи до погоршавања, основно правило код мутације је да се уносе мале промене вредности гена и то само на једном хромозому у једном тренутку (једној генерацији). Увођењем потпуно нове вредности гена, мутацијом се повећава и простор решења.

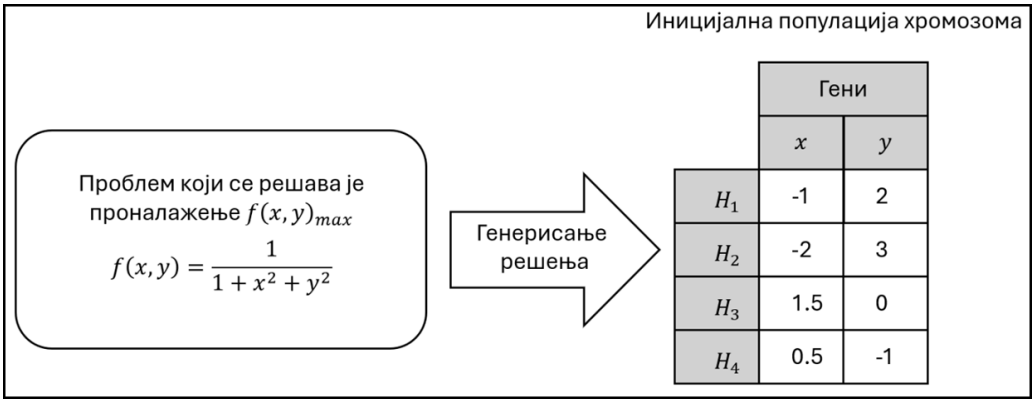


Слика 5.5: мутација хромозома на једном гену

Применом генетских оператора имплицитно су уведена још два концепта – *родитељи* и *деца*. Родитељи су парови хромозома који су одабрани за укрштање. Деца су нова два хромозома добијена укрштањем гена родитељских хромозома. Деца су хромозоми од којих се формира нова генерација, односно нова популација хромозома, а која ће се даље процењивати на основу вредности *fitness* функције (евалуација), на којој ће се вршити селекција хромозома за укрштање и на којој ће се изводити генетски оператори, тако из генерације у генерацију, све док се не дође до задовољавајућег решења проблема. Ова циклична (инкрементално

итеративна) природа ГА представља имплементацију еволутивног процеса, а и сам ГА се још зове еволутивни алгоритам.

Динамика еволуције се може приказати на једноставном примеру употребе ГА за решавање максимума задате функције (Слика 5.6). Пошто функција (леви део слике) има две промењиве које су реални бројеви, да би се применио ГА, потребно је да хромозом садрже два гена – за сваку промењиву по један. Формирана је иницијална популација која садржи четири таква хромозома означених H_1, H_2, H_3, и H_4 (табела десно на слици). Гени означени са *x* и *y* имају вредности дате у табели.



Слика 5.6: примена ГА у решавању максимума задате функције

Следећа фаза ГА је евалуација хромозома иницијалне популације (Слика 5.7). Функција чија се максимална вредност тражи заправо представља *fitness* функцију. Резултати показују да хромозоми H_3 и H_4 имају највеће, а хромозом H_2 има најмању вредност *fitness* функције.

Иницијална популација хромозома

	Гени		Функција одговарања *(fitness)*
	x	y	$f(x,y)$
H_1	-1	2	0.167
H_2	-2	3	0.007
H_3	1.5	0	0.31
H_4	0.5	-1	0.44

Слика 5.7: евалуација хромозома иницијалне йойулације

Следи поступак селекције који хромозом H_2 искључује, а хромозоме H_1, H_3 и H_4 бира за укрштање. Укрштањем се најпре формирају парови хромозома родитеља и то су (H_1, H_4) и (H_3, H_4), а затим долази до размене гена родитеља. Тиме се добија нова генерација хромозома – деца, која имају *x* ген од једног, а *y* ген од другог родитеља (Слика 5.8). На пример, дете хромозом H_1' садржи *x* ген од родитеља H_4, а *y* ген од родитеља H_3, или дете хромозом H_4' садржи *x* ген од родитеља H_{10}, а *y* ген од родитеља H_1. У десној колони се може видети унапређене вредности *fitness* функције у новој генерацији. Највећа вредност је скоро дупло већа него у претходној генерацији (0.8), док је најмања вредност већа од оне код хромозома који је био изабран за укрштање у претходној генерацији.

Популација хромозома Након укрштања

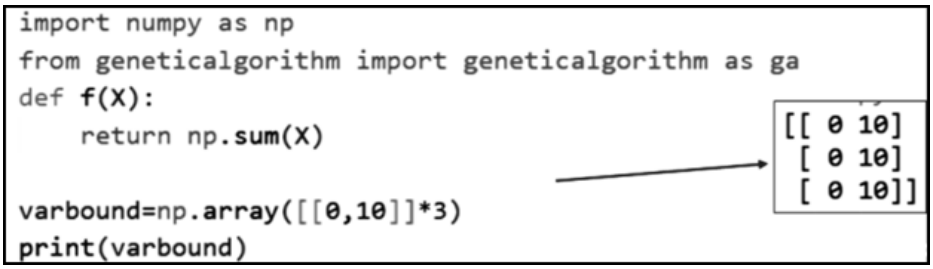
	Гени		Функција одговарања *(fitness)*
	x	y	$f(x,y)$
$H_1'(x_{H4}, Y_{H3})$	0.5	0	0.8
$H_2'(x_{H3}, Y_{H4})$	1.5	-1	0.24
$H_3'(x_{H4}, Y_{H1})$	0.5	2	0.19
$H_4'(x_{H1}, Y_{H4})$	-1	-1	0.33

Слика 5.8: Хромозоми деца и њихова евалуација

На описан начин настала је нова генерација (популација) хромозома означених H'_x. Ако је вредности *fitness* функције од 0.8 задовољавајућа, онда хромозом H'_1 представља апроксимативно решење задатог проблема, тако да се извршење ГА прекида. У случају да вредности *fitness* функције од 0.8 не задовољава, ГА се наставља селекцијом хромозома са највећим вредностима *fitness* функције на начин који је описан. У случају да нема даљег напредовања (вредност *fitness* функције се не побољшава), потребно је применити мутацију на начин објашњен за овај генетски оператор. Ако ни примена мутација не доводи до напредовања, потребно је поновно размотрити садржај иницијалне популације у смислу промене броја хромозома и/или промене вредности гена у хромозомима.

5.2. Пример примене генетског алгоритма

Постоји велики број *Python* библиотека које се баве ГА. У примеру је коришћена *pypi* библиотека *genticalgorithm*. Након инсталације, за тестирање библиотеке потребни су тест подаци и једноставна *fitness* функција. На пример (Слика 5.9), следећи *Python* код дефинише *fitness* функцију *f* која као параметар *X* прихвата структуру података, а као резултат враћа суму вредности елемената те структуре. Након тога *Python* код генерише матрицу 3x2 која дефинише опсеге унутар којих се могу генерисати вредности за гене хромозома. Оваква дефиниција указује да ће се иницијална популација хромозома генерисати тако да ће сваки од њих имати по три гена, а гени ће имати као вредности реалне бројеве из [0,10] интервала.



Слика 5.9: припрема података и fitness функције за тестирање библиотеке genticalgorithm

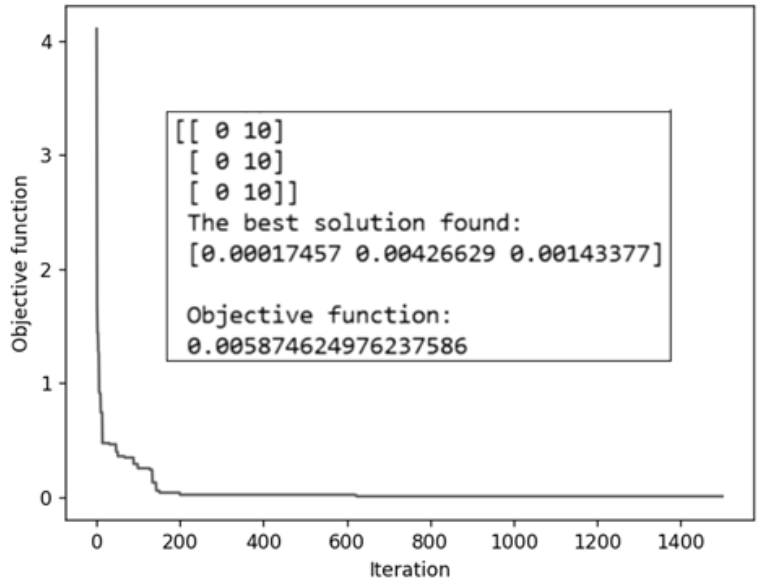
У даљој имплементацији, библиотека *genticalgorithm* конструише модел ГА на основу прослеђених параметара (Слика 5.10). Ових параметара може бити промењиви број. На примеру има 4 параметра (највише их може бити 14). Параметри су именовани и препознатљиви: *f* као *fitness* функција,

dimension је величина популације (број хромозома), следећа дефинише тип података (реални број) и четврта је иницијална популација, коју представља варијабла *varbound*. Друга наредба је извршна – њом се покреће ГА.

```
model=ga(function=f,dimension=3,variable_type='real',variable_boundaries=varbound)
model.run()
```

Слика 5.10: конструкција модела ГА у Python библиотеци genticalgorithm

Резултат рада ГА представљен је на илустрацији (Слика 5.11). Може се уочити да је логика рада модела да се добије што мања вредност *fitness* функције, која је у овом случају сума вредности гена у хромозомима. Ако није експлицитно подешено, модел ће радити десет секунди, што се и догодило на представљеном примеру. Друго опажање је да је ГА имао приближно 1500 итерација, дакле радило се са 1500 генерација хромозома. Напредовање је практично престало након нешто више од 600 генерација. Хромозом који представља решење проблема је такође на слици (*the best solution found*), а вредности његових гена су јако блиске 0 (вредности су реда 10^{-3} – 10^{-4}). Вредност *fitness* функције за овај хромозом је такође блиска 0 (~$5.87*10^{-3}$).



Слика 5.11: резултат добијен за 10 секунди рада ГА

Увид у остале параметре овог модела могуће је остварити позивом наредбе *param* на моделу ГА (Слика 5.12). У конкретном примеру није ограничен број итерација ГА (број генерација). Величина популације је 100 хромозома.

Вероватноћа мутације је 0.1, што значи да ће у свакој генерацији 10 хромозома бити изложено мутацији. Параметар *elit_ratio* означава проценат хромозома са најбољом вредношћу *fitness* функције који неће бити под дејством генетских оператора, већ ће само бити пренешен у следећу генерацију хромозома. Вредност 0.01 значи да ће само један *елитни* хромозом бити без модификација изабран у следећу генерацију. Вероватноћа укрштања (*crossover_probability*) је параметар који дефинише вероватноћу да ће доћи до укрштања. Вредност 0.5 значи да ће од предефинисаног броја хромозома за укрштање само 50% њих бити изабрано за укрштање. Параметар *parents_portion* означава колико ће родитељских хромозома да учествује у генетским операцијама да би се добила деца следеће генерације хромозома. Вредност 0.3 указује да ће 30 родитељских хромозома учествовати својим генима у формирању хромозома деце. Параметар *crossover_type* одређује начин избора гена од родитељских хромозома за децу хромозоме. Вредност *uniform* се користи кад се подједнак број гена користи од оба родитеља код оба хромозома детета. Последњи параметар (*max_iteration_without_improv*) има вредности *None*, што указује да не постоји ограничење у броју итерација у којима не постоји напредак у вредности *fitness* функције.

```
model=ga(function=f,dimension=3,variable_type='real',variable_boundaries=varbound)
print(model.param)
```

```
{
    'max_num_iteration': None,
    'population_size': 100,
    'mutation_probability': 0.1,
    'elit_ratio': 0.01,
    'crossover_probability': 0.5,
    'parents_portion': 0.3,
    'crossover_type': 'uniform',
    'max_iteration_without_improv': None
}
```

Слика 5.12: Преглед параметара модела ГА

Поред наведене библиотеке *genticalgorithm* постоји новија верзија *genticalgorithm2* развијена на основу искустава у коришћењу претходне верзије, са незнатним променама у погледу коришћења, редукованом скупу параметара (избачени су параметри који су се показали као сувишни). За више детаља погледати *github* линк (https://github.com/PasaOpasen/geneticalgorithm2-6.8.7?tab=readme-ov-file).

5.3. Закључак о генетском алгоритму

Генетски алгоритам је технологија вештачке интелигенције заснована на хеуристици којом је омогућено решавање проблема код којих су прихватљива приближна решења. Коришћењем концепата хромозома, као потенцијалних решења и променом њихове садржине коришћењем генетских оператора, генетски алгоритам напредује кроз итерације, мерећи квалитет апроксимативног решења величином функције губитка. Генетски алгоритам заправо учи у покушају проналажења решења. Примене генетског алгоритма су актуелне у индустријским процесима, саобраћају, процесима планирања и у науци.

5.4. Питања

1. Које су фазе генетског алгоритма?

2. У којим случајевима и како се користи мутације као генетски оператор?

3. Који су услови завршетка рада генетског алгоритма?

4. Шта представљају хромозоми а шта гени код генетског алгоритма?

5. Објасни намену параметара модела генетског алгоритма *genticalgorithm* коришћеном у примеру из лекције.

6. Представљен је пример проналажења максималног решења функције $\frac{1}{x^2+y^2+1}$ помоћу примене генетског алгоритма на популацији од 4 хромозома у две генерације. Шта ће се десити у трећој генерацији даљим укрштањем - напредак или назадовање?

7. Шта треба променити у коду на слици 78 како би ГА тражио као решење хромозом који има највећу суму?

6. Бајесова теорема и Бајесове мреже

6.1. Бајесова теорема и примена

Бајесова теорема представља начин за израчунавање вероватноће хипотезе (претпоставке) у зависности од евиденције (посматраног догађаја). На пример, служи за израчунавање вероватноће да ће да *ūага киша* (хипотеза) у случају да је облачно (евиденција). Овако дефинисана вероватноћа се записује као *P(H|E)* - постериорна вероватноћа тачности хипотезе *H* у случају појаве евиденције (догађаја) *E*. Математички, Бајесова теорема је формула (Израз 6.1) којом се израчунава постериорна вероватноћа $P(H_0|E)$ као количник производа условне вероватноће $P(E|H_0)$ и вероватноћа *a priori* $P(H_0)$ у бројиоцу и маргиналне вероватноће $P(E)$ у имениоцу.

$$P(H_0|E) = \frac{P(E|H_0) * P(H_0)}{P(E)} \qquad \text{И.6.1}$$

На основу претходног примера, ако вероватноћу кише изразимо као однос кишних дана и укупног броја дана у току године – онда је то вероватноћа *a priori* $P(H_0)$. Даље, може се изразити и вероватноћа да је облачно у случају да пада киша. Онда се ради о *условној* вероватноћи (енгл. *Likelihood*) $P(E|H_0)$. То је вероватноћа појаве евиденције (догађаја) E у случају тачности хипотезе H. Маргинална вероватноћа $P(E)$, или вероватноћа догађаја (евиденције) E, служи као нормализатор, који обезбеђује резултујућу вредност у интервалу [0,1]. Вредност маргиналне вероватноће се добија сумирањем производа условних вероватноћа евиденције од могућих хипотеза у посматраном домену (Израз 6.2). На основу претходног примера, постоје две могуће хипотезе (киша пада, или не пада) у случају да је облачно (*E*), тако да ће сума имати свега два сабирка.

$$P(E) = \sum_{i}^{n} P(E|H_i) * P(H_i) \qquad \text{И.6.2}$$

Бајесовом теоремом се практично врши ажурирање поверења у хипотезу након дејства евиденције на доменски контекст.

6.1.1. Примена Бајесове теореме за евалуацију теста на заразну болест

Примена Бајесове теореме биће представљена на примеру испитивања вероватноће тачне дијагнозе о заразној болести на основу теста. За решавање овог проблема значајни су статистички подаци како о самој болести у популацији, тако и о искуствима у коришћењу теста у откривању болести:

- 0.01% је проценат заступљености болести у популацији;
- у 95% случајева резултат теста је позитиван ако болест постоји.

Проблем који се решава може да се постави као израчунавање постериорне вероватноће хипотезе да болест постоји (H_0=⊤) у случају да је тест на болест позитиван (E=+). На основу претходних сазнања може да се изврши препознавање чинилаца Бајесове теореме:

- проценат заступљености болести у популацији представља приорну вероватноћу $P_{(H_0=\top)}$= 1% = 0.01;
- вероватноћа да је резултат теста позитиван ако болест постоји представља условну вероватноћу $P_{(E=+|H_0=\top)}$ = 95% = 0.95

Следи формулација Бајесовом теоремом Израз 6.3:

$$P_{(H_0=\top|E=+)} = \frac{P_{(E=+|H_0=\top)} * P_{(H_0)}}{P_{(E=+)}} \qquad \text{И.6.3}$$

У имениоцу количника је маргинална вероватноћа која се може изразити преко суме производа вероватноће евиденције са условним вероватноћама евиденције од могућих хипотеза у посматраном домену Израз 6.4:

$$P_{(E=+)} = P_{(E=+|H_0=\top)} * P_{(H_0=\top)} + P_{(E=+|H_0=\bot)} * P_{(H_0=\bot)} \qquad \text{И.6.4}$$

У горњем изразу први сабирак је производ вероватноће евиденције са условном вероватноћом евиденције (тест је позитиван) када је хипотеза тачна (болест постоји). Други сабирак је производ вероватноће евиденције (тест је позитиван) са условном вероватноћом евиденције када хипотеза није тачна (болест не постоји). Израчунавањем маргиналне вероватноће (Израз 6.4) добија се $P_{(E=+)} = \sim 0.06$ која ће послужити као корективни фактор за постериорну вероватноћу:

$$P_{(E=+)} = 0.95 * 0.01 + 0.05 * 0.99 = 0.0095 + 0.0495 = 0.059$$

На основу Бајесове теореме, (Израз 6.3) добија се постериорна вероватноћа:

$$P_{(H_0=\text{Т}|E=+)} = \frac{0.95 * 0.01}{0.059} = \frac{0.0095}{0.059} = 0.161$$

Закључак испитивања у овом примеру би био да је вероватноћа да ће болест бити потврђена тестом само 0.161, иако је вероватноћа да је тест позитиван, а болест постоји у 95% случајева ($P_{(E=+|H_0=\text{Т})}$ = 95%).

Другачијом интерпретацијом резултата може да се закључи да је вероватноћа да је тест негативан, а болест постоји у 5% случајева ($P_{(E=-|H_0=\text{Т})}$ = 5%). Пошто је заступљеност болести у популацији $P_{(H_0)}$ = 1%, онда се имплицитно може закључити да на 100 људи, поред једног стварно оболелог са позитивним тестом, постоји још 5 људи који имају позитиван тест али нису оболели.

Даље, претпоставимо да је дошло до епидемије и уместо једног на сто људи оболи педесет $P_{(H_0)}$ = 0.5, онда заменом и прорачуном добијамо већу маргиналну вероватноћу.

$$P_{(E=+)} = 0.95 * 0.5 + 0.05 * 0.5 = 0.5$$

, али и већу постериорну вероватноћу, пошто се иста вредност приорне вероватноће нашла у бројиоцу:

$$P_{(H_0=\text{Т}|E=+)} = \frac{0.95 * 0.5}{0.5} = 0.95$$

Интерпретација ове промене је да на основу Бајесове теореме, тест бива знанто поузданији у случају епидемија.

Ако се настави са експлоатацијом овог примера, питање које се намеће је: да ли може да се побољша ефикасност теста променом методе коришћења. Одговор је ДА. Рецимо да се уместо једног обави два тестирања, поставка израза на основу Бајесове теореме се онда мења (Израз 6.5).

$$P_{(H_0=\text{Т}|E_1=+,\, E_2=+)} = \frac{P_{(E_1=+|H_0=\text{Т})} * P_{(E_2=+|H_0=\text{Т})} * P_{(H_0)}}{P_{(E_1,E_2)}} \qquad \text{И.6.5}$$

Исто тако се мења поставка израза за маргиналну вероватноћу (Израз 6.6).

$$P_{(E_1=+,E_2=+)} = P_{(E_1=+|H_0=\top)} * P_{(E_2=+|H_0=\top)} * P_{(H_0=\top)} + P_{(E_1=+|H_0=\perp)} * P_{(E_2=+|H_0=\perp)} * P_{(H_0=\perp)}$$ И.6.6

Заменом и израчунавањем израза И.76 добија се мања маргинална вероватноћа него што је то у случају једног тестирања:

$$P_{(E_1=+,E_2=+)} = 0.95 * 0.95 * 0.01 + 0.05 * 0.05 * 0.99 = 0.0115$$

Заменом у изразу за постериорну вероватноћу (И.75), добија се нова вредност која је значајније бољa него у случају једног тестирања (~0.785 у односу на 0.161).

$$P_{(H_0=\top|E_1=+,\,E_2=+)} = \frac{0.95 * 0.95 * 0.01}{0.0115} = \frac{0.009025}{0.0115} = 0.784782$$

6.1.2. Примена Бајесове теореме за детекцију *spam*-а

Бајесова теорема се може успешно користити и за детекцију непожељне електронске поште (*spam*-а). На пример, потребно је детектовати електронска писма која садрже фразу „кликни овде“. При томе постоје статистички подаци да је 10% свих електронских писама заправо *spam*, а од њих 60% садржи фразу „кликни овде“. Поред тога ова фраза се појављује у 5% исправне електронске поште. Задатак је да се одреди вероватноћа да је е-писмо *spam* у случају да садржи фразу „кликни овде“.

На основу горњих података, вероватноћа *spam* од 10% је уствари вероватноћа а приори (нулта хипотеза):

$$P(S) = P_{(H_{0=\top})} = 0.1$$

Чињеница да 60% *spam* – ова садржи фразу „кликни овде“ је условна вероватноћа:

$$P_{(E=+|H_0=\top)} = 0.6$$

Вероватноћа која се тражи (да је е-писмо *spam* у случају да садржи фразу „кликни овде“) је према Бајесовој теореми постериорна вероватноћа $P_{(H_0=\top|E=+)}$.

Поред тога дат је податак да се фраза „кликни овде“ налази и у 5% исправне електронске поште. Ако је вероватноћа *spam*-а $P_{(H_{0=\top})} = 0.1$, онда је вероватноћа исправне поште $P_{(H_{0=\perp})} = 0.9$, а податак да се фраза налази у исправној пошти се онда може изразити као:

$$P_{(E=+|H_0=\perp)} = 0.05$$

Даље, маргинална вероватноћа може да се изрази као:

$$P_{(E=+)} = P_{(E=+|H_0=\top)} * P_{(H_{0=\top})} + P_{(E=+|H_0=\perp)} * P_{(H_{0=\perp})}$$
$$= 0.6 * 0.1 + 0.05 * 0.9 = 0.105$$

Сада постоје сви елементи за израчунавање тражене постериорне вероватноће:

$$P_{(H_0=\top|E=+)} = \frac{P_{(E=+|H_0=\top)} * P_{(H_0)}}{P_{(E=+)}} = \frac{0.6 * 0.1}{0.105} = \frac{0.06}{0.105} = 0.5714$$

Значи, 57% је вероватноћа да је е-писмо *spam* у случају да садржи фразу „кликни овде“.

Настављајући експлоатацију овог примера, обавља се проширење додавањем две нове фразе у *spam* филтер: “бесплатна услуга” и “награда” а њихове условне вероватноће су (табела 6.1).

Fraza	Oznaka evidencije	$P_{(E=+\|H_0=\top)}$	$P_{(E=+\|H_0=\perp)}$
„klikni ovde“	E_1	0.6	0.05
“besplatna usluga”	E_2	0.4	0.04
“nagrada”	E_3	0.7	0.07

Tabela 6.1: условне вероватноће фраза за детекцију спама

Вероватноћа *spam*-а је подсећамо $P_{(H_{0=\top})} = 0.1$ (остале поруке нису *spam* - $P_{(H_{0=\perp})} = 0.9$). При томе се подразумева да су фразе узајамно независне. Задатак је да се нађе вероватноћа да је електронска пошта *spam* у ако садржи појављивања три наведене фразе.

Најпре се израчунава производ вероватноће а приори и условних вероватноћа за поједине речи (дељеник Бајесове формуле):

$$P_{(E_1=+,E_2=+,E_3=+|H_0=\top)}$$
$$= P_{(H_{0=\top})} * P_{(E_1=+|H_{0=\top})} * P_{(E_2=+|H_{0=\top})} * P_{(E_3=+|H_{0=\top})}$$
$$= 0.1 * 0.6 * 0.4 * 0.7 = 0.0168$$

У циљу израчунавања маргиналне вероватноће затим се тражи производ појављивања фраза у исправној е-пошти ($H_0 = \perp$):

$$P_{(E_1=+,E_2=+,E_3=+|H_0=\perp)} = P_{(H_0=\perp)} * P_{(E_1=+|H_0=\perp)} * P_{(E_2=+|H_0=\perp)} * P_{(E_3=+|H_0=\perp)}$$
$$= 0.9 * 0.05 * 0.04 * 0.07 = 0.000126$$

Маргинална вероватноћа (делилац Бајесове формуле) је сума претходне две:

$$P_{(E_1=+,E_2=+,E_3=+)} = P_{(H_0=\top|E_1=+,E_2=+,E_3=+)} + P_{(H_0=\perp|E_1=+,E_2=+,E_3=+)} = 0.016926$$

Тражена вероватноћа да електронска пошта јесте *spam* (постериорна вероватноћа) је онда:

$$P_{(H_0=\top|E_1=+,E_2=+,E_3=+)} = \frac{0.0168}{0.016926} = 0.9925$$

Закључак је да је вероватноћа већа од 99% да је е-писмо *spam* у случају да садржи све три фразе, што је неупоредиво више од 57% када е-писмо садржи само фразу „кликни овде“.

6.1.3. Бајесова теорема и биномна расподела

Биномна расподела представља дискретну дистрибуцију вероватноће и користан алат за ажурирање вероватноћа – чинилаца у Бајесовој формули. Биномна расподела се може представити изразом (Израз 6.7):

$$P_{(x)} = \frac{N!}{x!\,(N-x)!}\Pi^x(1-\Pi)^{N-x} \qquad \text{И.6.7}$$

где је $\frac{N!}{x!(N-x)!}$ биномни коефицијент $\binom{N}{x}$, N је укупан број покушаја, x је број успешних покушаја, Π^xје вероватноћа успеха, $(1-\Pi)^{N-x}$је вероватноћа неуспеха.

Следи пример коришћења биномне расподеле и Бајесове теореме. На основу статистике које води компанија – малопродајни ланац, да би отварање продавница у новој регији било профитабилно, потребно је да се постигне 25%, или више учешћа на тржишту регије. Компанија ангажује агенцију за истраживање тржишта да би проверила услове за потенцијално инвестирање у новој регији. Агенција је реализовала анкету на узорку од 20 корисника и добила резултат да је пет корисника веома заинтересовано за куповину производа тог малопродајног ланца. На основу анкете, агенција претпоставља да је њен резултат довољан за инвестирање у новој регији обзиром да је резултат 25% (пет од двадесет) заинтересованости.

Компанија – малопродајни ланац међутим процењује да је узорак мали, те даје недовољан да би резултати истраживања били прихватљиви.

Компанија међутим има следеће пословне резултате (историјске податке) у другим регијама (Табела 6.2). Види се да 70% продавница у малопродајном ланцу постиже тржишно учешће од 25% и више.

Učešće na tržištu (%)	Broj prodavnica (%)
10	5
15	5
20	20
25	**20**
30	40
35	10
	Σ=100%

Табела 6.2: Преглед учешћа на тржишту које је постигла компанија

На основу анкете и горње табеле, продавнице у новој регији би ушле у групу 20% продавница које постижу 25% тржишног учешћа, што значи да је нулта хипотеза $H_0 = 0.25$, док је вероватноћа *a priori* $P_{(H_{0=\text{T}})} = 0.2$ (случај тачне хипотезе).

Заменом, биномна расподела се користи за ажурирање условне вероватноће:

$$P_{(E|H_0)} = P_{(x=5|N=20)} = \frac{20!}{5!\,(20-5)!}0.25^5(1-0.25)^{20-5} = 0.20233$$

Дељеник Бајесове теореме је онда:

$$P_{(E|H_{0=\text{T}})} * P_{(H_{0=\text{T}})} = 0.20233 * 0.2 = 0.040466$$

Маргинална вероватноћа се израчунава као сума производа свих условних вероватноћа за задате хипотезе (колона в, Табела 12) и њихових априорних вероватноћа (колона δ, табела 12) (колона г, Табела 12). Исто као у случају нулте хипотезе, биномска расподела се користи за ажурирање условне вероватноће и за све остале хипотезе. На пример, за случај 30% учешћа на тржишту то би било:

$$P_{(E|H_0)} = P_{(x=5|N=20)} = \frac{20!}{5!\,(20-5)!}0.3^5(1-0.7)^{20-5} = 0.071544$$

То значи да се у свим случајевима користи иста вредност биномног коефицијента $\binom{20}{5}$. Након ажурирања условних вероватноћа биномном расподелом и за све остале хипотезе, добија се вредност маргиналне вероватноће:

$$P_{(E=+)} = \sum_{H_i} P_{(E=+|H_i)} P_{(H_i)} = 0.16638$$

На крају, тражена постериорна вероватноћа да ће нова инвестиција одговарати случају од 20% продавница које имају 25% тржишног учешћа је:

$$P_{(H_{0=\text{T}}|E=+)} = \frac{P_{(E|H_{0=\text{T}})} * P_{(H_{0=\text{T}})}}{P_{(E=+)}} = \frac{0.040466}{0.16638} = 0.243$$

Међутим ако би се израчунале постериорне вероватноће и за остале пожељне исходе (пошто је услов >=25%), а то су продавнице које остварују 25%, 30% и 35% тржишног учешћа (табела 6.3), онда је укупна постериорна вероватноћа за сва 3 случаја 0.749 (Табела 6.3, колона д). То значи да је, на основу података анкете и статистичких података пословања компаније у другим регијама, добијена вероватноћа успешног инвестирања у нову регију ~75%.

Учешће на тржишту (Hi)	Учешће продавница компаније P(Hi)	Условна вероватноћа P(E|Hi) ажурирана биномном расподелом	Производи условне и приорне вероватноће P(E|Hi)* P(Hi)	Постериорна вероватноћа
а	б	в	г	д
10%	0.05	0.03192	0.001596	0.00959
15%	0.05	0.10285	0.005142	0.00309
20%	0.2	0.17456	0.034912	0.20983
25%	**0.2**	**0.20233**	**0.040466**	**0.24321**
30%	0.4	0.17886	0.071544	**0.43**
35%	0.1	0.1272	0.01272	**0.07645**
Маргинална вероватноћа			**0.16638**	

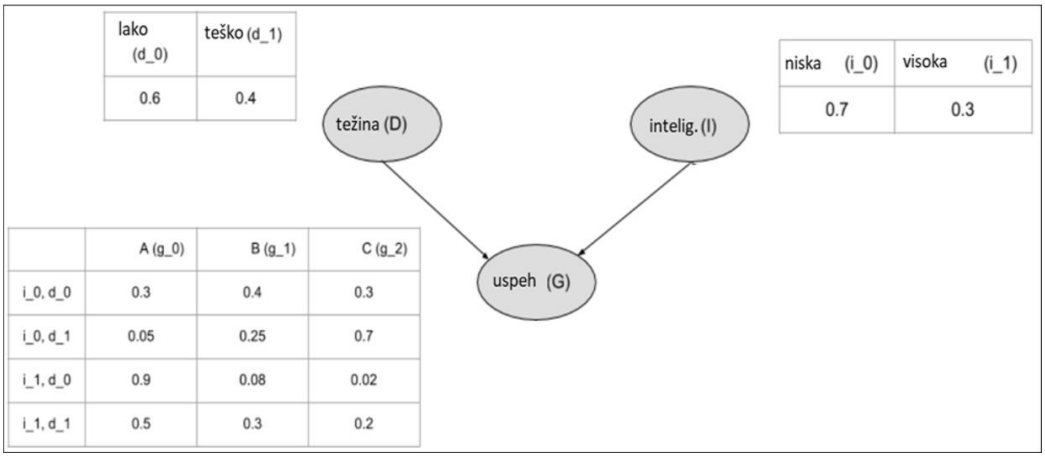
Табела 6.3: израчунавање маргиналне вероватноће преко биномне дистрибуције

6.2. Бајесове мреже

Бајесове мреже представљају графички модел пробабилистичких веза – условних вероватноћа између промењивих од интереса. Бајесове мреже се заснивају на Бајесовој теореми. У системима са малим бројем промењивих Бајесова теорема нуди технике контроле условних вероватноћа. Међутим, за комплексне системе, који имају велики број промењивих, при чему промењиве могу имати велики број различитих вредности које су вишеструко условљене вредностима других промењивих, Бајесове мреже представљају решење. Зову се још и мреже поверења.

Бајесова мрежа је усмерен ациклични граф. На следећем примеру (Слика 6.1) је Бајесова мрежа која је намењена предикцији резултата на испиту, представљеног промењивом *G* у зависности од интелигенције (промењива

I) и тежине испита (промењива *D*). Промењиве су представљене као чворови у графу, док усмерени лукови показују зависност између промењивих. Промењива *D* је безусловна (независна) и има само две могуће вредности: лако и тешко. За чвор је везана дистрибуција вероватноће могућих вредности промењиве (представљена табеларно). На датом примеру за промењиву *D* (тежине испита) дата је вероватноћа 0.6 да је испит лак, односно вероватноћа 0.4 да је испит тежак. Слична ситуација је и са интелигенцијом – дистрибуција вероватноћа за поједине вредности промењиве *I* је 0.7 za nisku i 0.3 za visoku inteligenciju. Дистрибуција вероватноћа за промењиву *G* (успех) је комплекснија из два разлога. Први је тај да је условљена промењивама *D* и *I*, а други је да има 3 могуће вредности (A, B i C). Множењем могућих стања све три промењиве добија се укупан број дистрибуција за промењиву *G*, а то је дванаест ($N_D * N_I * N_G = 2 * 2 * 3 = 12$). На слици се такође може уочити да сума дистрибуција вероватноћа по редовима мора увек да буде један.



Слика 6.1: Bajesova mreža kao aciklični graf

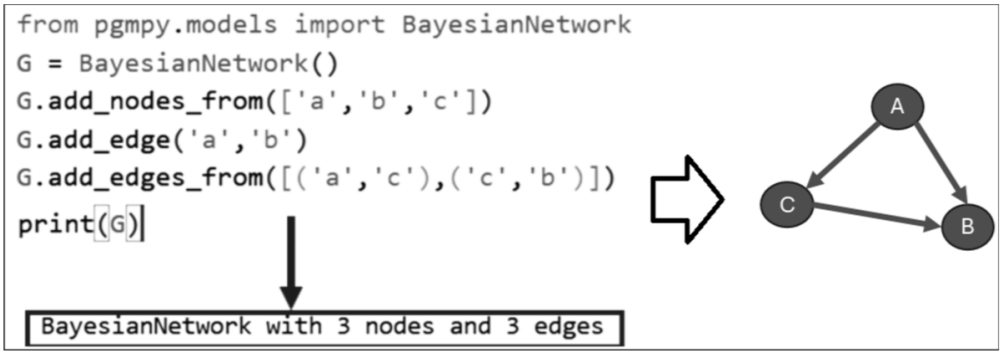
Бајесова теорема омогућава промену постојећег ‘поверења’ под утицајем нових чињеница. На тај начин омогућено је да се новим подацима омогући кориговање постојећих знања и експертиза. Систем се тако динамички ‘мења’ и ‘учи’.

6.2.1. Примери конструкције Бајесових мрежа у Python-у

У циљу представљања конструкције Бајесових мрежа у Python-у коришћен је модул *pgmpy*. Он се ослања на друге модуле које *pip installer* аутоматски преузима и инсталира ако већ не постоје на рачунару: *networkx, numpy,*

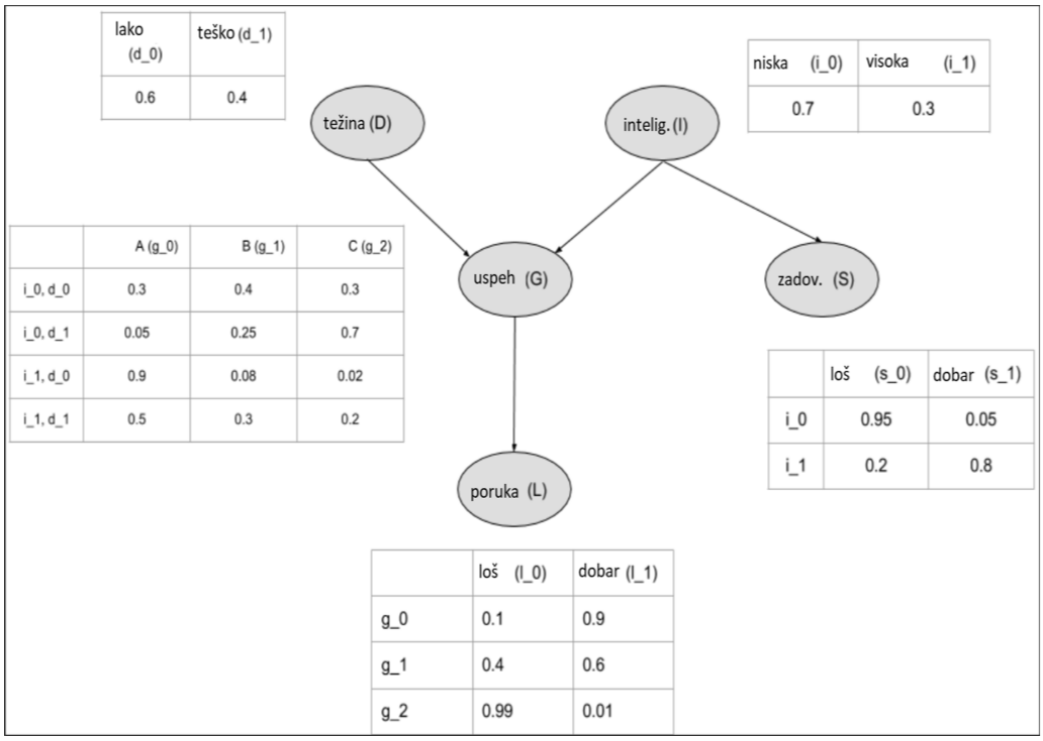
scipy, scikit-learn, pandas, pyparsing, torch, statsmodels, tqdm, sympy, jinja, colorama, fsspec (напомена је да је *torch* инсталација од ~200MB).

У циљу тестирања инсталације користи се модел *BayesianNetwork*. Поступак почиње тиме да се најпре конструише објекат модела у радној меморији (Слика 6.2), а затим да се додају чворови и лукови. На примеру је представљено додавање више чворова прослеђивањем низа њихових ознака као параметра функције. Чворови се могу додавати и појединачно, као што је то учињено са додавањем лука између чворова *a* и *b*. Такође је представљено и групно додавање лукова између чворова *a* и *c*, као и чворова *c* и *b*.



Слика 6.2: Конструкција једноставне Бајесове мреже

Након успешног тестирања, следи пример конструкције Бајесове мреже за процену успеха на основу интелигенције и тежине испита. Модел коришћен у уводном разматрању о Бајесовим мрежама је проширен са још две промењиве (Слика 6.3). Прва од њих је условљена интелигенцијом (промењивом *I*). Ова промењива представља кумулативну процену студента на пријемном испиту (у погледу поседовања математичких и језичких вештина и знања) и означена је симболом *S*. Пошто промењива *S* има две могуће вредности (*лош* и *добар*), а условљена је промењивом *I* која такође има две могуће вредности, онда њена табела дистрибуција условних вероватноћа садржи 4 могуће вредности. Друга промењива, означена симболом *L* представља процену општег успеха студента. Ова промењива такође има две могуће вредности (*лош* и *добар*), има две могуће вредности (*лош* и *добар*), а њена табела дистрибуција условних вероватноћа садржи 6 могућих вредности, обзиром да промењива *G* има 3 могуће вредности (*A*,*B* и *C*) ($N_G * N_L = 3 * 2 = 6$).



Слика 6.3: Бајесова мрежа за предикцију успеха студената

У овом примеру је коришћен нови модел Бајесове мреже из *pgmpy* библиотеке зван *BayesianModel*. Метод намеће конструкцију Бајесове мреже. Овог пута конструктору се прослеђује листа уређених парова чворова. На тај начин се дефинишу не само чворови, већ и лукови у мрежи. Полазишни чвор је први, а одредишни чвор други у пару (Слика 6.4).

```
from pgmpy.models import BayesianModel
from pgmpy.factors.discrete import TabularCPD

model = BayesianModel([('D', 'G'), ('I', 'G'), ('G', 'L'), ('I', 'S')])
```

Слика 6.4: Конструкција комплетне Бајесова мреже

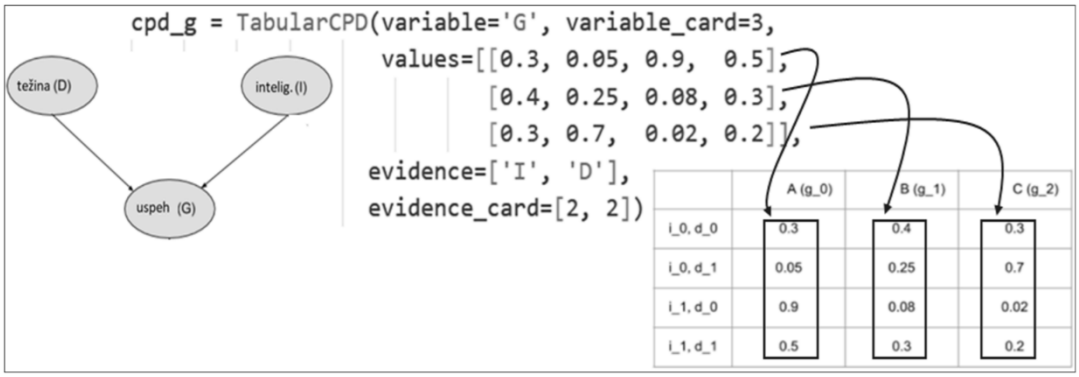
За разлику од претходног примера (Слика 6.4), за чворове се дефинише дистрибуција вероватноћа. У ту сврху је коришћена класа *TabularCPD* (od eng. *Tabular Conditional Probability Distribution*) чији је задатак да омогући дистрибуцију условних вероватноћа чворовима у мрежи. Прво се дефинишу табеле дистрибуције вероватноћа безусловним чворовима (независне промењиве), а затим условним. Следећи код (Слика 6.5) представља додавање табела дистрибуције вероватноћа безусловним

чворовима, промењивама *D* и *I*. За ту намену се користи конструктор *TabularCPD* који има као први параметар назив чвора за који се табела креира (*variable*), затим број могућих вредности (*variable_card*) и вектор дистрибуција вероватноћа које табела садржи (*values*). Ово су основни параметри, а поред њих још се користе *state_names* називе стања (на пример, за веријаблу *D* то би било *state_names={'D': ['lako, 'teško']}*).

```
cpd_d = TabularCPD(variable='D', variable_card=2, values=[[0.6], [0.4]])
cpd_i = TabularCPD(variable='I', variable_card=2, values=[[0.7], [0.3]])
```

Слика 6.5: Додавање табела дистрибуције вероватноћа безусловним чворовима

Пример дефинисања табеле дистрибуције вероватноћа у условном чвору дат је следећим кодом (Слика 6.6). У питању је промењива *G,* која има 3 могуће вредности, али је условљена са два независна чвора - промењиве *I* и *D* (параметар *evidence*), које имају по две могуће вредности (параметар *evidence_card*). У овом случају дистрибуције вероватноћа су представљене матрицом, код које редови представљају дистрибуције по вредностима (*A,B* и *C*), а колоне су контролне, пошто збирови по колонама морају да буду један.



Слика 6.6: Додавање табеле дистрибуције вероватноћа безусловном чвору

Иако су табеле дистрибуције вероватноћа додаване чворовима, морају и експлицитно да се додају у модел.

Следећи код представља додавање свих табела дистрибуције вероватноћа моделу Бајесове мреже (Слика 6.7). Као што се може уочити из претходног кода, табеле дистрибуција су означене идентификаторима *cpd_** , који су прослеђени моделу као параметри методе *add_cpds*.

```
model.add_cpds(cpd_d, cpd_i, cpd_g, cpd_l, cpd_s)
```

Слика 6.7: Додавање свих табела дистрибуције вероватноћа моделу Бајесове мреже

Експлицитно додавање табела дистрибуције вероватноћа моделу Бајесове мреже омогућава његову брзу модификацију. На пример, да би се додали називи стања промењивама, метод је да се направи нова табела дистрибуције, а затим да се новом замени стара табела дистрибуције. Следећи код (Слика 6.8) представља креирање две нове табеле дистрибуције вероватноћа (намењене промењивама *D* и *I*), а које за разлику од претходних, имају именована стања (вредности промењивих).

```
cpd_d_sn = TabularCPD(variable='D', variable_card=2, values=[[0.6], [0.4]],
                      state_names={'D': ['Lako', 'Teško']})
cpd_i_sn = TabularCPD(variable='I', variable_card=2, values=[[0.7], [0.3]],
                      state_names={'I': ['Niska', 'Visoka']})
```

Слика 6.8: Први корак модификације Бајесове мреже – пример именовања стања

Замена старих новим табелама дистрибуција условних вероватноћа врши се додавањем нових табела методом *add_cpds* (Слика 6.9). Модел ће на основу идентификатора промењивих да изврши аутоматску замену.

```
model.add_cpds(cpd_d_sn,cpd_g_sn,cpd_i_sn,cpd_l_sn, cpd_s_sn)
```

Слика 6.9: Ажурирање модела Бајесове мреже новим табелама дистрибуција условних вероватноћа

На потпуно идентичан начин могу се мењати и вредности у табелама дистрибуција условних вероватноћа. Провере података у табелама се врше преко објеката *TabularCPD*. Модел на бази класе *BayesianModel* има уграђену методу *get_cpds* која враћа листу свих табела са основним подацима (Слика 6.10). На пример, за чвор *D* је табела означена *P(D)* што указује да се ради о безусловној промењивој, која има два стања. У случају чвора *G* представљена је табела дистрибуције условних вероватноћа *P(G|I, D)* у зависности од чворова *I* и *D*. Испис такође указује да промењива *G* има 3 стања, а промењиве *I* и *D* имају по 2 стања.

```
print(model.get_cpds())
```

```
[<TabularCPD representing P(D:2) at 0x115ae39cd70>, <TabularCPD representing
P(G:3 | I:2, D:2) at 0x115e795d970>, <TabularCPD representing P(I:2) at 0x115
e786f620>, <TabularCPD representing P(L:2 | G:3) at 0x115e795d940>, <TabularC
PD representing P(S:2 | I:2) at 0x115e795dc40>]
```

Слика 6.10: Приказ основних података у табелама дистрибуција условних вероватноћа

Детаљан извештај о одређеној табели дистрибуција условних вероватноћа може се добити *print* наредбом чији је параметар идентификатор табеле. На следећем примеру је детаљан приказ табеле дистрибуција условних вероватноћа за промењиву - чвор *G* (Слика 6.11).

```
print(cpd_g_sn)
```

I	I(Niska)	I(Niska)	I(Visoka)	I(Visoka)
D	D(Lako)	D(Teško)	D(Lako)	D(Teško)
G(A)	0.3	0.05	0.9	0.5
G(B)	0.4	0.25	0.08	0.3
G(C)	0.3	0.7	0.02	0.2

Слика 6.11: Приказ детаљних података у табели дистрибуција условних вероватноћа

На сличан начин могу се добити детаљни извештаји о зависностима у мрежи (метода *local_independances*), као и могућим путањама (метода *active_trail_nodes*).

За коришћење овако дефинисане Бајесове мреже, поред саме мреже, потребно је дефинисати и алгоритам за закључивање. Постоје два алгоритма: *VariableElimination* и *BeliefPropagation* (модул *inference* библиотеке *pgmpy*). У примеру је коришћен алгоритам *VariableElimination* (Слика 6.12). Метода коришћења је потпуно идентична за оба алгоритма. Најпре се конструктору алгоритма прослеђује модел, а затим се врши упит – захтев за закључивање на основу прослеђених параметара. За постављање упита користи се метода *query* алгоритма за закључивање. Први параметар је чвор над којим се врши упит, остали параметри су евиденције – конкретни подаци на основу којих ће Бајесова мрежа произвести закључак. За улазне податке – тежина испита лака, а студент

има високу интелигенцију, модел даје закључак да се од студента очекује најбоља оцена (са постериорном вероватноћом $P(G = 'A'|D = 'Lako', I = 'Visoka') = 0.9$).

```
infer = VariableElimination(model)
print(infer.query(['G'], evidence={'D': 'Lako', 'I': 'Visoka'}))
```

```
+------+----------+
| G    |   phi(G) |
+======+==========+
| G(A) |   0.9000 |
+------+----------+
| G(B) |   0.0800 |
+------+----------+
| G(C) |   0.0200 |
+------+----------+
```

Слика 6.12: коришћење Бајесове мреже – упит на интерном чвору

Може се поставити упит за било који чвор у графу (тј. промењиву у Бајесовој мрежи). На основу претходно датог графичког приказа Бајесове мреже, види се да је чвор *G* интерни чвор графа. У наставку ће се извршити упит на спољњем чвору – промењивој *L* (Слика 6.13). За улазне податке – испит је тежак, а студент има ниску интелигенцију, модел даје закључак да се очекује лош општи успех са постериорном вероватноћом $P(L =' Loš'|D =' Teško', I =' Niska') = 0.798$, односно добар успех са значајно мањом постериорном вероватноћом $P(L = 'Dobar'|D =' Teško', I =' Niska') = 0.202$.

```
print(infer.query(['L'], evidence={'D': 'Teško', 'I': 'Niska'}))
```

```
+----------+----------+
| L        |   phi(L) |
+==========+==========+
| L(Loš)   |   0.7980 |
+----------+----------+
| L(Dobar) |   0.2020 |
+----------+----------+
```

Слика 6.13: коришћење Бајесове мреже – упит на екстерном чвору

6.3. Закључак о Бајесовој теореми и Бајесовим мрежама

Важност Бајесове теореме се састоји у томе што представља начин за ажурирање вероватноћа на основу нових евиденција (података). На тај

начин Бајесова теорема умањује неизвесност у домену одлучивања и класификације. Бајесова теорема је основ за закључивање у многим системима у којима постоји потреба за перманентним ажурирањем вероватноћа неке хипотезе (претпоставке) на основу нових података који се прикупљају и увећавају временом.

Бајесове мреже омогућавају примену Бајесове теореме у условима великог броја својстава од утицаја, при чему ова својства узајамно зависе у одређеној мери. Наведена зависност изражена је дистрибуцијом условних вероватноћа у Бајесовим мрежама. Захваљујући овом својству, Бајесовим мрежама се моделују системи вештачке интелигенције, намењени за предвиђање, дијагностику и закључивање у комплексним доменима.

6.4. Питања

1. Која је функција маргиналне вероватноће код Бајесове теореме?
2. Зашто се користи биномска расподела код примене Бајесове теореме?
3. У систему доставе хране 0.1% од свих достава залута, а 15% су пице. Поред тога пице се налазе у 50% исправних достава. Која је вероватноћа да ће достава да залута у случају да садржи пицу?
4. Претпоставимо да је вероватноћа да особа има туберкулозу (ТБ) 0,0005, и да је тест за ТБ 99% тачан. Која је вероватноћа да особа има ТБ ако тест покаже позитиван резултат?
5. У неком региону на основу статистичких података, 51% одраслих чине мушкарци. Такође, 9,5% мушкараца и 1,7% жена су пушачи. Која је вероватноћа да је случајни пролазник који пуши цигарету мушког пола?
6. Особа је преузела посао. Вероватноћа да ће посао бити завршен на време ако пада киша је 0,44, а вероватноћа да ће посао бити завршен на време ако не пада киша је 0,95. Ако је вероватноћа да ће падати киша 0,45, одредите вероватноћу да ће посао бити завршен на време.
7. Када се користи Бајесова теорема, а када Бајесове мреже при моделовању система?

7. Припрема података за коришћење у системима ВИ

У вештачкој интелигенцији подаци се користе у фази обучавања модела система вештачке интелигенције (машинско учење) и у фази коришћења у циљу кластеровања, класификације, предикције, за закључивање, доношење одлука и решавање различити врста проблема. Подаци могу да буду текстуални и бинарни (бајтовски).

Текстуални подаци су на пример, текстуални записи у неком језику и нумерички подаци. Нумерички подаци се најчешће представљају такође као текстуални записи због интероперабилности између система вештачке интелигенције и других система. У том случају се користе цифре (0 1 2 3 4 5 6 7 8 9), децимални сепаратори (тачка, или зарез, на пример, 0,123 i 23.353) и префикс који означава негативне бројеве (на пример, негативан -23.432 и позитиван 23.432). Поред тога, текстуални записи нумеричких података имају и ознаку у ком бројном систему је нумерички запис (имплицитно је децимални, иначе нпр. o – октални, x ,или 0x – хексадецимални). За велике бројеве користе се још и додатни симболи (карактери) за запис у експоненцијалном облику (слово *e,* на пример у запису 34e3 означава да је број $34*10^3$).

Сви остали подаци се сврставају у бинарне (бајтовске записе). Мултимедијални подаци спадају у ову категорију (слика, звук, видео), као и записи архивираних и компресованих података. Бајтовски записи имају начелне формате како би се омогућила њихова размена између система вештачке интелигенције и осталих система.

7.1. Типови података

У односу на потребе машинског учења, подаци се могу поделити у:

- структуиране,
- делимично структуиране и
- неструктуиране.

Структуирани подаци су организовани у унапред дефинисаном формату. Формати могу да буду линеарни и нелинеарни. Линеарно структуирани подаци се појављују у виду низова (вектора), листа, матрица и других

структура са више димензија које омогућавају директан приступ елементима (коришћењем индекса), или секвенцијалан приступ (претрагом листе елемената). Нелинеарне структуре су организоване у виду једног или више стабала (хијерархијске структуре), или као мреже (графови).

Неструктуирани подаци немају унапред дефинисан формат. Примери неструктуираних података су слободне форме текстуалних података као што су е-писма и постови на социјалним мрежама, ту спадају и слике, аудио и видео записи. Неструктуирани подаци су тежи за обраду и готово увек је потребна претходна припрема (енгл. *preprocessing*), како би били употребљиви у системима вештачке интелигенције.

Делимично структуирани подаци садрже неки вид организације, али само ради обезбеђивања форме која олакшава претрагу садржаја и конверзију у различите типове података. Садржај делимично структуираних података је врло флексибилан по броју димензија, по величини и по типу података. Пример таквих формата података су XML (од енгл. *Extensible Markup Language*) и JSON (од енгл. *JavaScript Object Notation*). За обраду делимично структуираних података потребно је познавање њихове унутрашње организације (шеме података).

7.2. JSON (JavaScript Object Notation)

JSON представља текстуални формат за представљање и размену сложених података (објеката) у основи структуиран у кључ-вредност парове. JSON је настао као резултат оптимизације XML формата, како би се смањила укупна количина података (енгл. *overheading*) у размени између апликација на мрежи. За разлику од XML формата, који за сваки елемент структуре дефинишу експлицитно почетно и завршно обележје (енгл. *tag*) код JSON формата је за ту намену искоришћена витичаста заграда као основни сепаратор (Слика 7.1):

```
{
    "oid": 9,
    "ime": "Горан Мишић",
    "lozinka": "23413dsfs",
    "profesija": "administrator"
}
```

Слика 7.1: пример једноставног објекта записаног у JSON формату

Без обзира на једноставност, JSON омогућава представљање линеарних и хијерархијских структура. Тако да се JSON-ом могу представити сложени објекти, може се вршити угњеждавање једног објекта у другом и представљање колекција (Слика 7.2).

```
{
    "ime":      "Petar",
    "prezime": "Marković",

    "adresa":  {
        "država": "Srbija",
        "grad":   "Beograd",
        "ulica":  "Grofa Drakule",
        "broj":   41
    },

    "email":                "p_markovic@xyzf.bg.ac.rs",
    "datum_rođenja":        "06.04.1995.",
    "prosek_ocena":         9.21,
    "redovan_student":      true,
    "postdiplomske_studije": false,
    "objavljeni_radovi":    null,

    "kursevi_asistent": [
        "Osnove programiranja",
        "Uvod u Javu",
        "Uvod u Python"
    ]
}
```

Слика 7.2: JSON запис сложеног објекта

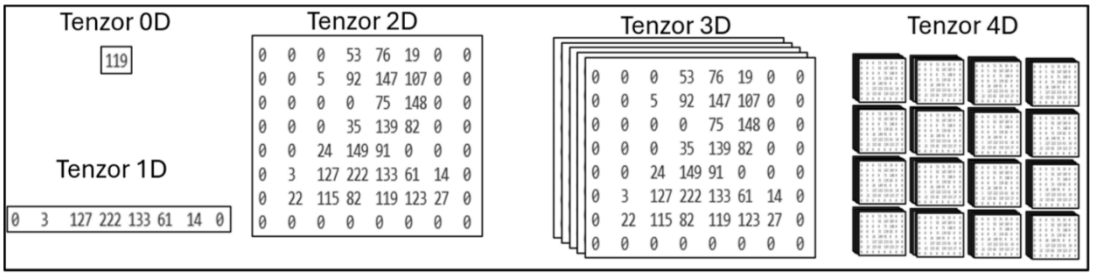
7.3. Структуре података за машинко учење

Модели вештачке интелигенције који имплементирају машинско учење су дизајнирани тако да се могу обучавати и користити податке који су различито структуирани. Конвенционални типови (на пример ентитетске класе, најчешће коришћене у објектно орјентисаним системима) су неефикасне за коришћење у системима машинског учења. Други проблем је коришћење података који имају слабу структуираност и могу имати

различит садржај и различите димензије. Из наведених разлога у машинском учењу се користе неконвенционални типови података који омогућавају већу флексибилност и ефикасније руковање подацима.

7.3.1. Тензори

Тензори (енгл. *Tensor*) су флексибилне структуре података циљно дизајниране за машинко учење. Математичка интерпретација тензора указује да се ради о вишелинеарној (више вектора, или низова) структури, која садржи податке истог типа (најчешће нумеричке), намењеној за мапирање простора вектора доменских података у простор вектора неког предефинисаног опсега вредности. У системима вештачке интелигенције, тензори представљају прилагодљиве јединице података за обраду. Тензор може да буде изоловани податак (нема димензију), вектор (једна димензија), матрица (две димензије), или нека сложенија линеарна структура (Слика 7.3).



Слика 7.3: примери тензора

На пример, ако су улазни подаци низови матрица (имају 3 димензије), а на улазу се очекују вектори (1 димензија), тензори омогућавају оптимизовану и ефикасну трансформацију из низа матрица у вектор. Исто се дешава ако су улазни подаци матрично структуирани, а на улазу се очекују сложенији типови. Тензори омогућавају брзо додавање потребних димензија. Поред наведеног битно је и то да тензори омогућавају брзу измену редоследа димензија. На пример, ако се ради о моделима вештачке интелигенције намењеним за предикцију, доминантна је временска димензија у односу на остала својства која се разматрају. Други пример представља класификација слика, у ком случају су доминантне просторне димензије и димензије (*RGB*) канала слике.

Због перформанси, тензори се користе и за чување међу-података у процесу машинског учења као што су синаптичке тежине и прагови

осетљивости у вештачким неуронским мрежама и матрица својстава у конволуционим мрежама. Поред тога, омогућавају брзе матричне операције и диференцирање података за процес обучавања модела.

7.3.2. *DataFrame* формат података

DataFrame је други најчешће помињан формат података у машинском учењу. Ради се о *Python* класи која енкапсулира (угњеждава) дводимензионалну структуру податка, која за разлику од тензора, може да садржи податке различитих типова. Поред тога, *DataFrame* омогућава именовање колона и редова, што одговара организацији података у поља и записе у релационим базама података. На тај начин, *DataFrame* објекти су врло ефикасни за учитавање велике количине података из релационих база, као и из датотека као што су *JSON, CSV, MS Excel* и *Open Office Calc*.

DataFrame класа садржи многобројне методе које омогућавају манипулацију подацима како што је:

- филтрирање и екстракција података по задатим критеријумима,
- чишћење и измена података који су непотпуни, избацивање дупликата, допуњавање непостојећих података,
- трансформација структуре, на пример редови у колоне и обратно, или из листе у посебне записе елемената,
- израчунавање статистичких параметара скупа података (на пример, средња вредност, минимуми, максимуми, број елемената, стандардна девијација),
- груписања података по критеријуму, израчунавање у померајућим прозорима и
- брзо памћење у поменуте формате датотека, у релационе базе.

7.3.3. *Dataset* формат података

Dataset је формат података (*Python* класа) прилагођен машинском учењу, сличан *DataFrame* формату, али једноставнији и са другом наменом. Док је код *DataFrame* формата посебно важно именовање редова и колона, код *Dataset* формата ради се о колекцији узорака (н-торке), којима се приступа секвенцијално, један по један узорак у итерацијама. *Dataset* формат је погодан за велике серије података, погодан је за руковање комплексним објектима, као што су текст, слике и тензори. Често се користи за обучавање

модела, обзиром да омогућава секвенцијално захватање података из спољње меморије, што *DataFrame* формат не омогућава (код кога се читав скуп података одједном учитава у радну меморију). *Dataset* објекат такође може да садржи податке различитих типова. *Dataset* објекат се често користи у комбинацији са *DataFrame* објектом обзиром да је *DataFrame* објекат ефикаснији за учитавање података, док је *Dataset* објекат ефикаснији у процесу обучавања модела вештачке интелигенције.

7.4. Учитавање података

Прва фаза рада са подацима у машинском учењу је њихово учитавање у радну меморију апликације. На следећим примерима представљена су учитавања података из датотека *csv* (енгл. *Comma Separated Values*) и xlsx (*MS Excel*) формата (Слика 7.4). Прва и трећа наредба користи *Python* модул *pandas* ради учитавања *csv* и *xlsx* датотека (методе *read_csv* и *read_excel*).

```
dataframe1 = pd.read_csv("indicator_values.csv")
dataframe2 = pd.read_excel("indicator_values.xlsx")
```

Слика 7.4: различити начин учитавања података

Учитавање података из система за управљање базама података се врши позивом методе *read_sql* из *Python* модула *pandas* (Слика 7.5). Као параметри се прослеђују упит за издвајање жељених записа података (параметар *sql_upit*) и конекциони стринг, или објекат, што се разликује у зависности од типа система за управљање базама података (параметар *konekcioni_string*). Резултат функције је већ формиран *DataFrame* објекат (промењива *dataframe_obj*). Након извршења упита, конекција са базом се раскида методом *close*.

```
dataframe_obj = pandas.read_sql(sql_upit, konekcioni_string)
konekcioni_string.close()
```

Слика 7.5: учитавање података из система за управљање базама података

7.5. Припрема података

Након учитавања следи припрема података за машинско учење. Овај временски захтевни процес укључује различите активности као што су:

- чишћење података – обухвата уклањање нетачних података, дупликата, допуњавање недостајућих података;
- трансформација података – обухвата скалирање, нормализацију, кодирање не-нумеричких података, на пример замена текстуалних и логичких вредности целим бројевима;
- издвајање својстава од интереса – уместо захватање свих обележја, издвајање својстава која имају информативну вредност у односу на задатке модела машинског учења; уједно се смањује и комплексност скупа података; реализује се анализом главних компоненти (енгл. *principal component analysis - PCA*) и регресивном анализом;
- дељење података на подскупове за обучавање и евалуацију модела машинског учења;
- стратификација (балансирање) података – обухвата поступке како би подаци за обучавање били избалансирани у смислу изједначавања броја узорака у класама, или другим видовима груписања података; некада се захтева додавање синтетички генерисаних података (енгл. *over-sampling*), а у случају великих скупова података се примењује чак уклањање узорака који припадају доминантној класи (енгл. *under-sampling*);
- форматирање података – обухвата усклађивање оригиналног формата података улазном формату модела машинског учења.

7.5.1. Чишћење података

Чишћењем података откривају се грешке у подацима, које се затим исправљају или уклањају. Такође, откривају се недостајући подаци који се надопуњавају синтетички генерисаним како би се добио конзистентан скуп за обучавање модела машинског учења. *DataFrame* класа је посебно погодна за ову намену. Она садржи методе за све наведене функционалности:

- *drop_duplicates* – метода којом се избацују дупликати - редови података који садрже идентичне вредности,
- *dropna* - метода којом се избацују редови који садрже недостајуће вредности;
- *fillna* - метода којом се надопуњавају недостајуће вредности; ако су у питању нумерички подаци обично се синтетички генеришу

вредности на бази мера централне тенденције (*mean, median*) и на бази стандардне девијације популације узорака; у случају текстуалних података, надопуњавање се врши неким предефинисаним, или подразумеваним вредностима;

- *astype* - метода којом се тип података конвертује у циљни тип;
- *clip* - метода често коришћена и којом се уклањају екстремне вредности (енгл. *outliers*) из скупа за обучавање

7.5.2. Нормализација података

Изворни подаци различитих својстава у истом скупу података могу се веома разликовати у опсезима вредности, што може да доведе до проблема у процесу обучавања и коришћења модела машинског учења. Задатак нормализације је да вредности свих својстава сведе на исти интервал вредности. Најчешће је скалирање у интервалу [0,1] и у ту сврху се користи *минимум – максимум* метода (Израз 7.1), која даје нормализовану вредности податка својства у скупу x' као количник који у бројиоцу има разлику вредности која се нормализује x и минималне вредности својства x_{min}, док у имениоцу има разлику максималне x_{max} и минималне вредности својства у скупу података.

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \qquad \text{И.7.1}$$

Применом *минимум – максимум* методе, подаци су сведени на [0,1] интервал, при чему њихови узајамни односи нису нарушени. За примену *минимум – максимум* методе у *Python*-u користи се класа *MinMaxScaler* из модула *sklearn.preprocessing* (Слика 7.6). Након инстанцирања објекта, нормализација се изводи позивом фукције *fit_transform*, којој се прослеђују подаци као *DataFrame* објект (параметар *dataframe*).

```
scaler = MinMaxScaler()
darray_scaled = scaler.fit_transform(dataframe)
```

Слика 7.6: Нормализација минимум – максимум *методом*

Као резултат, функција враћа матрицу од које је потребно направити нови *DataFrame* објект ради даље припреме података.

7.5.2. Дељење података

Дељење података у машинском учењу се врши с циљем избора и формирања скупа података за обучавање и скупа за проверу обучености (евалуацију) модела. За ту сврху користи се метода *train_test_split* из модула *sklearn.model_selection* (Слика 7.7). Ова метода има 3 параметра: *DataFrame* објекат са нормализованим подацима (*dataframe_scaled*), величина дела података за проверу обучености модела (*test_size*) и начин на који се узоркују (*random_state*).

```
train_df, test_df = train_test_split(dataframe_scaled, test_size=0.2, random_state=42)
```

Слика 7.7:дељење података за обучавање и проверу модела машинског учења

У случају малих скупова података за машинско учење, пропорција приликом поделе треба да буде у корист подскупа за обуку, која треба да се заснива у што већој мери на оригиналним подацима. За проверу обучености могу да се користе и класе синтетички генерисаних података који ће се кретати у маргинама очекиваних вредности за дистрибуцију података у оригиналном скупу.

7.5.3. Дељење података стратификацијом

Чест случај је да подаци у скупу података за машинско учење нису добро избалансирани у смислу фреквенције података по класама за класификацију. На пример, ако постоје 2 класе у које се подаци класификују (нпр. позитивни и негативни узорци) дисбалансом се може назвати ситуација у којој позитивних има неколико пута више од негативних узорака (података) у скупу за обучавање у односу на скуп за проверу обучености модела машинског учења. *Python* модул *sklearn* нуди као решење такозвану стратификација података, која одржава баланс података (фреквенцију класа) у скуповима за обучавање и за проверу обучености. Стратификација се може извршити већ у току дељења података *train_test_split* методом, додавањем параметра *stratify* са *y* вредношћу (Слика 7.8).

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=42, stratify=y)
```

Слика 7.8: дељење података стратификацијом

У случају да у скупу података енкапсулираном у *DataFrame* објект у још нису дефинисане класе, стратификација се може извршити и коришћењем *DataFrame* методе *sample* (Слика 7.9), која врши насумични избор података и формира нови *DataFrame* објект из кога се даље издвајају подскупови за обучавање и проверу обучености.

```
df = dataframe_scaled.sample(frac=1, random_state=42).reset_index(drop=True)
```

Слика 7.9: стратификација података

7.6. Припрема текстуалних података

Припрема текстуалних података (докумената) за машинско учење је комплекснија обзиром на својства говорног језика и на чињеницу да се текстуални подаци не могу обрађивати у моделима машинског учења, нити процесуирати алгоритмима на бази математичких операција. Пре припреме текста потребно је извршити конверзију у слова исте величине (најчешће у мала слова). За специфична национална писма потребно је текст конвертовати у писмо које је препознатљиво алатима за припрему текста. На пример, ако је текст у ћириличном писму, потребно га је пресловити у латинично ако алати не препознају ћирилицу.

Припрема текста се одвија у следећим фазама:

- Дељење (енгл. *tokenization*) – текст (документ) се дели на вектор појединачних речи;
- Уклањање интерпункције и стоп речи – из текста се уклања садржај који није информативан, то су сви знаци интерпункције, предлози, везници, речце и поштапалице ,
- Нормализација речи (лематизација, енгл. *stemming*) – све речи се доводе у основну форму која најчешће представа префиксну основу речи која је задржала информативни садржај (Слика 7.10); на пример, глаголи се доводе у инфинитивни облик, а заједничке именице у прво лице једнине;

```
У време обновљања уметности, дефиниције уметности теже да постану сумњиве.
                                ⇩
          vrem obnovljanj umetnos definicij umetnos tezx posta sumnjiv
```

Слика 7.10: изглед оригиналног текста и текста након лематизације

- Конверзија у нумеричку форму – текстуални податак се конвертује у формат усклађен са моделима машинског учења. представља нумеричким вредностима добијеним кроз статистичку обраду текста и његову векторизацију како би имао формат прилагођен улазном формату модела машинског учења.

7.6.1 Конверзија текста у нумеричку форму

Конверзија текста у нумеричку форму изводи се кроз статистичку обраду текста и његову векторизацију како би имао формат прилагођен улазном формату модела машинског учења. Као статистичке вредности, најчешће се користе фреквенција термина (енгл. *Term Frequency - TF*) и инверзна фреквенција документа (енгл. *Inverse Document Frequency - IDF*). Комбинација ове две вредности представља нумеричку репрезентацију текста, која се затим користи приликом формирања скаларног вектора. Овај вектор садржи, поред индекса документа и речи, нумеричку вредност TF-IDF – тежински фактор чије су вредности у интервалу [0,1].

Фреквенција термина t у документу d се израчунава (Израз 7.2) као количник броја појављивања речи t у документу $N_{t,d}$ и укупног броја речи у документу $N_{r,d}$. На пример, ако документ има 4000 речи и ако се реч *модел* појављује у документу 40 пута, онда је вредност 0.01 његова фреквенција.

$$TF_{t,d} = \frac{N_{t,d}}{N_{r,d}} \qquad \text{И.7.2}$$

Инверзна фреквенција докумената IDF_t за реч t представља фреквенцију речи у скупу докумената који се анализирају и израчунава се (Израз 7.3) као логаритамска вредност количника укупног броја докумената који се анализирају N и броја докуменатаN_t који садрже барем једно појављивање речи t (јединица је додата у количник да би се избегло дељење нулом). На пример ако има 4000 докумената, а у 400 докумената постоји реч *модел*, вредност IDF_t је 0.998; у случају да реч *модел* постоји само у 4 од 4000 докумената, вредност IDF_t је 0.29;

$$IDF_t = \log(\frac{N}{N_t + 1}) \qquad \text{И.7.3}$$

Множењем фреквенције термина и инверзне фреквенције документа за сваку реч у документу добија се резултујућа вредности означена као TF-IDF фактор од кога се формира векторска репрезентација документа.

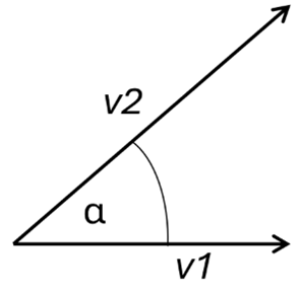
На следећој илустрацији је пример векторизованог текста (Слика 7.11). Вектор се састоји из уређених парова бројева од којих први представља индекс докумената, а други индекс речи у документу. Сваком пару је придружена вредност TF-IDF фактора. У процесима класификације овакви вектори могу узајамно да се упоређују, при чему је величина сличности, или разлике у потпуности квантитативно одређена.

```
(0, 32)       0.34092454627167806
(0, 23)       0.34092454627167806
(0, 36)       0.34092454627167806
(0, 2)        0.34092454627167806
(0, 38)       0.5501115545929777
(0, 18)       0.34092454627167806
(0, 39)       0.34092454627167806
(1, 1)        0.3333333333333333
(1, 24)       0.3333333333333333
(1, 4)        0.3333333333333333
(1, 7)        0.3333333333333333
(1, 8)        0.3333333333333333
(1, 10)       0.3333333333333333
(1, 19)       0.3333333333333333
(1, 13)       0.3333333333333333
(1, 11)       0.3333333333333333
(2, 3)        0.27065689895808104
(2, 37)       0.27065689895808104
```

Слика 7.11: приказ структуре TF-IDF вектора

7.6.2. Косинусна сличност

Једна од најчешће коришћених мера сличности између два вектора података је *косинусна* сличност (eng. *Cosine Similarity*). Косинусна сличност се заснива на *косинус* тригонометријској функцији. Косинусна функција као параметар има угао између 2 вектора. Вредности косинусне функције су у интервалу [-1, 1]. На примеру (Слика 7.12), вредност косинуса угла α, који заклапају вектори v_1 и v_2 представља однос скаларних вредности ова два вектора (v_1/v_2).



Слика 7.12: вектори и косинус α

Косинусна сличност је мера која се користи у различитим областима вештачке интелигенције. На пример: у обради природног језика, у системима за одлучивање, у машинском учењу и у анализи података.

7.6.2.1. Пример имплементације косинусне сличности у Python-u

У примеру је представљено мерење сличности 3 кратка текста на основу вредности TF-IDF фактора и косинусне сличности векторске репрезентације текстова. У имплементацији је коришћена функција *cosine_similarity* из *sklearn.metrics.pairwise* модула (Слика 7.13). За векторизацију текста коришћена је класа *TfidfVectorizer*, која конвертује прослеђена документа (текстове) у матрицу вредности TF-IDF фактора. Након векторизације, функција *cosine_similarity* користи матрицу вредности TF-IDF фактора како би израчунала сличност између текстова.

```
tekst1 = "Danas je subota"
tekst2 = "Danas je subota pre podne"
tekst3 = "Danas je nedelja poslepodne"

tekstovi = [tekst1, tekst2, tekst3]

vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(tekstovi)
cosine_sim_matrix = cosine_similarity(tfidf_matrix)

print(f"Kosinusna sličnost tekstova '{tekst1}' i '{tekst2}':", cosine_sim_matrix[0][1])
print(f"Kosinusna sličnost tekstova '{tekst1}' i '{tekst3}':", cosine_sim_matrix[0][2])
print(f"Kosinusna sličnost tekstova '{tekst2}' i '{tekst3}:'", cosine_sim_matrix[1][2])
```

Слика 7.13: пример упоређивања 3 текста на бази косинусне сличности

Из представљених резултата упоређивања текстова (Слика 7.14), уочава се је највећа сличност прва два документа (~0.62), док је најмања сличност између другог и трећег документа.

```
Kosinusna sličnost 'Danas je subota' i 'Danas je subota pre podne': 0.6241069585778812
Kosinusna sličnost 'Danas je subota' i 'Danas je nedelja poslepodne': 0.37602162360618324
Kosinusna sličnost 'Danas je subota pre podne' i 'Danas je nedelja poslepodne:' 0.23467771186837183
```

Слика 7.14: резултати упоређивања 3 текста на бази косинусне сличности

Косинусна сличност омогућава да се упоређују документа различитих величина, што је искоришћено у претрагама докумената по задатим критеријумима, који садрже само неколико кључних термина. Векторска репрезентација критеријума претраге се доводи у везу са векторима докумената чиме је омогућено мерење косинусне сличности између тих вектора и систем за претрагу даје резултате рангиране на основу тих вредности. Вредности косинусне сличност блиске јединици указују на велику сличност између упоређиваних вектора, док негативне вредности и вредности у околини 0 указују да сличност између вектора не постоји или је занемарљива.

7.7. Закључак

Припрема података за машинско учење је захтеван процес од кога у великој мери зависи будућа ефикасности модела за класификацију, предикцију и кластеровање. Најчешће коришћене структуре података у машинском учењу су тензори и објекти *DataFrame* класе. Ова два типа, која се надопуњују, разликују се по структури и намени. Тензори немају ограничења у димензионалности, али су ограничени на један основни тип податка (нумерички). *DataFrame* објекти су увек дводимензионални, али омогућавају груписање различитих типова података и ефикасну манипулацију подацима. Поред наведених типова постоје и други формати, са специјализованом наменом, на пример као што су ретке матрице, мрежне структуре и временске серије. Нарочито је важна припрема података за обучавање модела машинског учења. Поред потребног броја података, неопходно је обезбедити избалансираност података у односу на класе за класификацију. Припрема текстуалних података за коришћење у моделима машинског учења је захтеван процес издељен у сукцесивне фазе, кроз које се од изворног текста на крају добија његова нумеричка векторска репрезентација, погодна за обраду.

7.8. Питања

1. Зашто су текстуални подаци значајни у размени података система вештачке интелигенције са другим системима?

2. Која је разлика између структуираних и делимично структуираних података?

3. Каква би била графичка репрезентација тензора који има 5 димензија?

4. Из којих формата датотека *DataFrame* објекти могу да читају/уписују податке?

5. Које врсте манипулације подацима омогућавају методе *DataFrame* објекта?

6. Које су сличности, а које разлике између *DataFrame* и *Dataset* објекта?

7. Шта обухвата чишћење података у оквиру припреме за обучавање модела машинског учења?

8. Који су циљеви издвајања својстава од интереса?

9. Шта је стратификација?

10. Којом методом се искључују екстремне вредности из скупа података за обучавање?

11. Који је однос фреквенције термина и инверзне фреквенције докумената?

12. Да ли је већа косинусна сличност вектора који су у првом и четвртом квадранту, или вектора који су у другом и трећем квадранту?

8. Коришћење података у системима вештачке интелигенције

8.1. Регресија

Регресија је једна од основних метода вештачке интелигенције. Регресија је статистички метод који доводи у везу зависну промењиву са једном, или више независних промењивих. Регресивна анализа је статистички поступак који доказује зависност између посматраних промењивих. Свака промењива представља једно својства (енгл.. *feature*) које је од значаја на пример, за класификацију података, за предвиђање будућих вредности података, у процесима доношења одлука и слично.

Својства (промењиве) којима се описује (моделује) реалан систем најчешће утичу једно на друго. За промењиве које су узајамно зависне каже се да су у корелацији. Корелација је термин изведен из термина релација што значи веза, док префикс ко (лат.*co*) указује да се промењиве разматрају заједно. Другим речима – анализира се њихова узајамна повезаност и открива да ли/како утичу једна на другу.

Утицај између промењивих може бити директан (непосредан), или индиректан (посредан). Ако је утицај између промењивих директан, онда је најчешће и очигледан. У примеру промењивих *облачност* и *падавине*, ако је чињеница да је *облачно*, онда има услова за *падавине*. У примеру је очигледна повезаност независне промењиве *облачност* и зависне промењиве *падавине*.

У проблемима којима се бави вештачка интелигенција најчешће постоји мноштво својстава (промењивих), док узајамне зависности нису очигледне и морају се доказивати. У доказивању зависности користи се регресиона анализа. Зависност између две промењиве може бити линеарна и нелинеарна.

Регресија такође може бити линеарна и нелинеарна, али за разлику од зависности између својстава, код регресије ови термини имају друкчија значења. Регресивном анализом покушава се направити модел који најбоље *одговара* (често се користи термин *фитује* од енгл. глагола *fit*) зависности између посматраних промењивих. У реалним проблемима

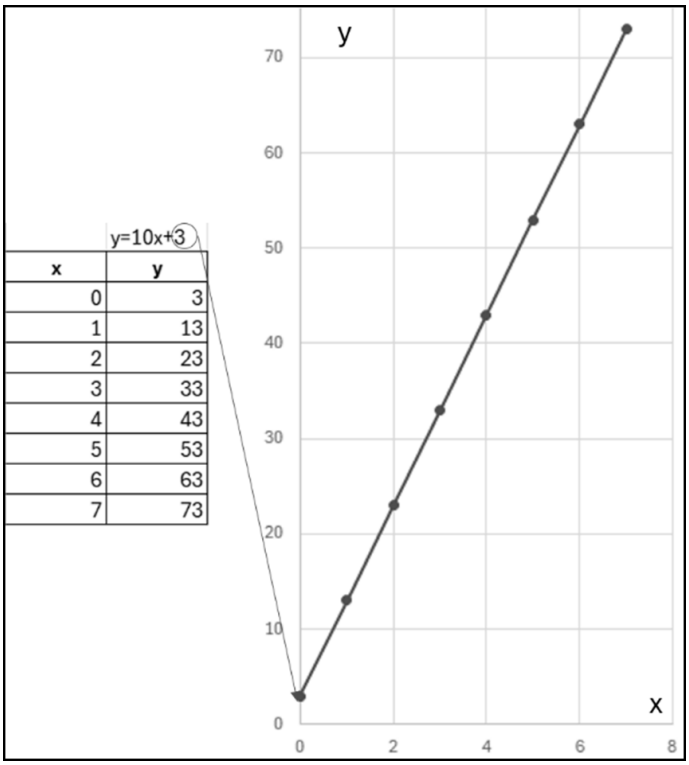
готово да и не постоји линеарна зависност између нека два својства (на основу измерених вредности), али нешто што је „приближно“ томе представља линеарну регресију. Слично је и са нелинеарним зависностима – идеални случајеви су нелинеарне математичке функције, али у пракси њих скоро да и нема, па се покушава направити модел који приближно одговара реалном систему. Тако добијен модел је нелинеарни регресивни модел.

8.1.1. Линеарна регресија

У случају две узајамно зависне промењиве, линеарна регресија се може представити математички линеарном функцијом као у следећем изразу (Израз 8.1), у коме је *y* зависна промењива, *x* независна промењива, *k* је коефицијент нагиба регресивне линије и *a* одређује тачку пресека функције на *y* оси.

$$y=kx+a \qquad \text{И.8.1}$$

На следећој илустрацији (Слика 8.1) је пример линеарне зависности података у табели лево (колоне *x* и *y*). Коефицијент нагиба има вредност 10, а тачка пресека на ординати има вредност три.

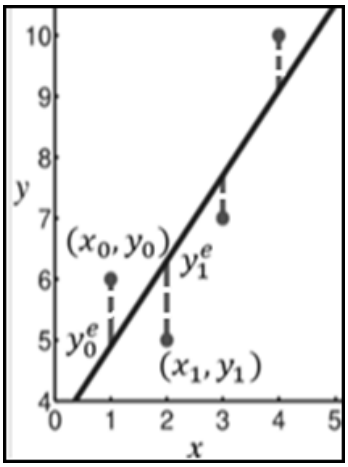


Слика 8.1: Пример линеарне зависности

Да би се омогућило моделовање реалних система, у линеарну функцију се уводи још један параметар, којим се представља одступање података зависне промењиве од очекиваних. Модификована једначина (Израз 8.2) тако добија додатни параметар ϵ, који представља грешку (енгл. *error / residual*).

$$y = kx + a + \epsilon \qquad \text{И.8.2}$$

На следећем примеру (Слика 8.2) дате су 4 тачке података, које одступају од регресивне линије. Величине одступања за сваки узорак (x_i, y_i) представљен је вертикалним испрекиданим линијама које повезују тачке података и регресивну линију.



Слика 8.2: Одсшуйања реалних йодашака од реїресионе линије

Величине одступања се математички израчунавају као разлика очекиване и реалне вредности зависне промењиве y_i . Ова разлика може да има позитивну и негативну вредност. На примеру прва два узорка (y_0 и y_1), то се може исказати следећим изразима у којима је одступање за прва два узорка представљено са ϵ_0 и ϵ_1 док су очекиване вредности зависне варијабле представљене са y_0^e и y_1^e (Израз 8.3).

$$\epsilon_0 = y_0 - y_0^e > 0\ ,\ \epsilon_1 = y_1 - y_1^e < 0 \qquad \text{И.8.3}$$

У случају ϵ_0, обзиром да је одступање реалне од очекиване вредности већа од нуле, каже се да је реална вредности зависне промењиве *йоцењена* (енгл. *underestimation*). У другом случају (ϵ_1), обзиром да је одступање реалне од очекиване вредности мања од нуле, ради се о *йрецењивању* (енгл. *overestimation*) реалне вредности зависне промењиве.

Вредност одступања реалне од очекиване вредности зависне промењиве ϵ може се изразити сумом (Израз 8.4), у којој N представља број узорака, док e_i представља одступање i-тог узорка од очекиване вредности. Овако добијена вредности може бити позитивна, или негативна, што зависи да ли има више позитивних од негативних одступања и од саме величине одступања.

$$\epsilon = \sum_{i=1}^{N} e_i \qquad \text{И.8.4}$$

Други начин је представљање сумом квадрата (енгл. *SSE – sum of squared errors*, или *SSR – sum of squared errors*), који се више користи (Израз 8.5) обзиром да је увек позитиван и боље наглашава случајеве када су постоје велика одступања стварних и очекиваних вредности.

$$\epsilon = \sum_{i=1}^{N} (e_i)^2 \qquad \text{И.8.5}$$

8.1.1.1. Процена зависности између две променљиве код линеарне регресије

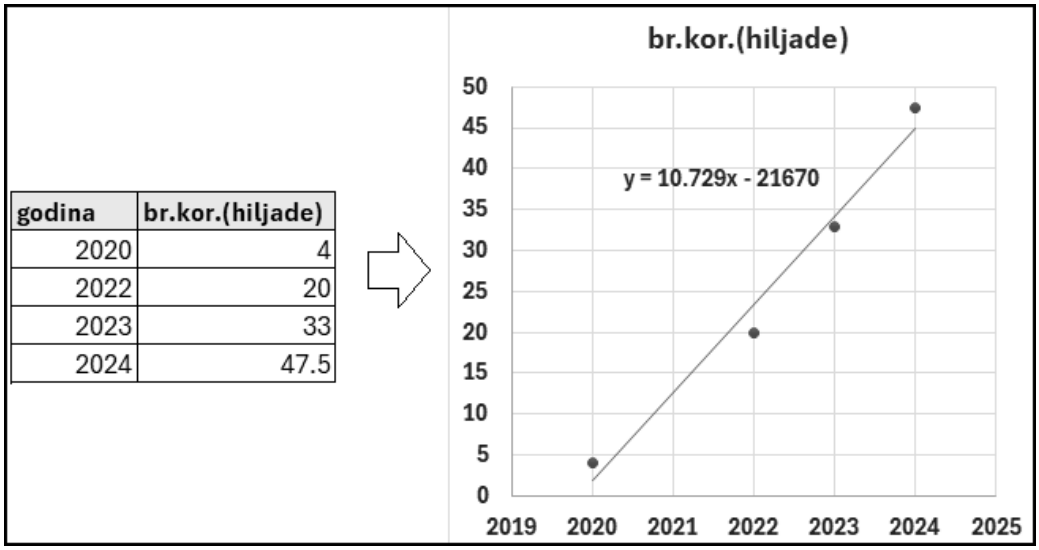
Код линеарне регресије, у случају великих одступања реалних вредности у популацији узорака од регресивне линије, потребно је извршити додатну процену зависности између две промењиве. За ту намену користи се Пирсонов коефицијент корелације (енгл. *Pearson correlation coefficient - PCC*). Вредност Пирсоновог коефицијента се израчунава изразом (Израз 8.6) у коме се користе вредности и независне и зависне промењиве (x_i, y_i) и број узорака у популацији (N).

$$r = \frac{N(\sum_{i=1}^{N} x_i y_i) - \sum_{i=1}^{N} x_i \sum_{i=1}^{N} y_i}{\sqrt{[N \sum_{i=1}^{N} x_i^2 - (\sum_{i=1}^{N} x_i)^2][N \sum_{i=1}^{N} y_i^2 - (\sum_{i=1}^{N} y_i)^2]}} \qquad \text{И.8.6}$$

Вредност Пирсоновог коефицијента се креће у интервалу [-1,1]. Негативне вредности овог коефицијента указују на негативну корелацију независне и зависне промењиве. У пракси то значи да ће позитиван тренд (пораст) вредности независне промењиве довести до негативног тренда (опадања) вредности зависне промењиве и обрнуто. Ако су вредности коефицијента блиске нули, онда је корелација безначајна, а за вредности коефицијента блиске |1| корелација је значајна. Средње вредности коефицијента (у околини |0.5|) указују на средњу корелацију између промењивих и у зависности од конкретног случаја могу се узети као значајне, или не.

Дат је пример пораста броја корисника услуге мобилног банкарства у једној банкарској установи. Подаци (Слика 8.3) се односе за 4 године од којих прва година није узастопна – другом речју, недостајући су подаци за 2021. годину. Очигледно да *година* представља независну промењиву, а *број*

корисника зависну промењиву у релацији. Дијаграм који одговара датим подацима указује да је зависности ове две промењиве у великој мери линеарна. Приказана је и регресивна линија као и њена једначина.



godina	br.kor.(hiljade)
2020	4
2022	20
2023	33
2024	47.5

Слика 8.3: Подаци о порасту броја корисника и регресивна линија

На основу израза за израчунавање Пирсоновог коефицијента (Израз 8.6), у иницијалну табелу додате су колоне за међу-резултате ($x_i y_i$, x_i^2, y_i^2) и за суме (Слика 8.4). За последњу фазу израчунавања додата је посебна табела која садржи резултујуће вредности бројиоца и имениоца, као и сам количник који представља вредност Пирсоновог коефицијента.

godina	br.kor.(hiljade)	xy	xe2	ye2
2020	4	8080	4080400	16
2022	20	40440	4088484	400
2023	33	66759	4092529	1089
2024	47.5	96140	4096576	2256.25
SUME				
8089	104.5	211419	16357989	3761.25

Pearson	BROJILAC	375.5
	IMENILAC	379.9556
	KOLICNIK	0.988273

Слика 8.4: Израчунавање Пирсонової коефицијента

Добијен резултат (0.98) указује да постоји врло значајна (*јака*) позитивна корелација између година и броја корисника као независне и зависне промењиве.

8.1.1.2 Имплементација примера линеарна регресије у Python-у

У овом поглављу је представљено креирање и коришћење предикативног модела на бази линеарне регресије са имплементацијом у *Python*-у.

Искоришћен је претходни пример о порасту броја корисника услуге мобилног банкарства у једној банкарској установи из претходног поглавља (секција 8.1.1.1). У ту сврху је искоришћена класа *LinearRegression* из *sklearn.linear_model* библиотеке (Слика 8.5).

```
import numpy as np
from sklearn.linear_model import LinearRegression
```

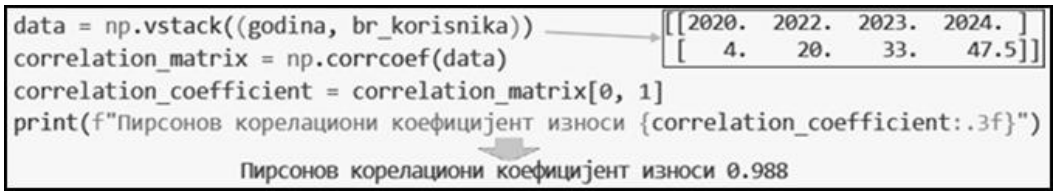
Слика 8.5: ресурси потребни за имплементацију примера

Пошто модел *LinearRegression* омогућава имплементацију и сложенијих регресивних модела, улазне податке за обучавање (једне или више независних промењивих) очекује као матрицу, а за вредности зависне промењиве очекује вектор. У циљу прилагођавања моделу, од листа година и листа бројева корисника по тим годинама се формирају тражене структуре. За ту сврху искоришћена је *array* метода *numpy* модула. Да би се добила матрица 4x1 за године (промењива *godina2d*), искоришћена је метода *reshape* (Слика 8.6).

```
godina = np.array([2020, 2022, 2023, 2024])    [[2020]
godina2d = godina.reshape(-1,1)                 [2022]
br_korisnika = np.array([4, 20, 33, 47.5])      [2023]
                                                [2024]]
                  [ 4.  20.  33.  47.5]
```

Слика 8.6: прилагођавање података моделу LinearRegression

Процена зависности независне и зависне промењиве у конкретном случају проверава се *Пирсоновим* коефицијентом корелације (секција 8.1.1.1.). Подаци (промењива *data*) се најпре *vstack* методом припремају за формирање корелационе матрице, која се конструише позивом *corrcoef* методе којој се прослеђују подаци спремљени у виду матрице (Слика 8.7). Као што је приказано, постоји јака позитивна корелација независне и зависне промењиве (0.988), на основу чега је закључак да постоји потпуна оправданост представити ову зависност линеарним регресивним моделом.



Слика 8.7: Процена зависности независне и зависне променљиве Пирсоновим коефицијентом корелације

Преостале карактеристике модела – коефицијент регресионе линије и тачка пресека са *y* осом могу се добити преко параметара модела (Слика 8.8).

```
print(f"Presek: {model.intercept_}")
print(f"Koeficijent regresije: {model.coef_}")

Presek: -21669.728571428568
Koeficijent regresije: [10.72857143]
```

Слика 8.8: коефицијент регресионе линије и тачка пресека са y осом

За конструкцију модела линеарне регресије је искоришћен подразумевани конструктор класе *LinearRegression* (Слика 8.9). Конструисан модел (промењива *model*) се затим прилагођава (обучава) позивом методе *fit* прослеђивањем матрице вредности независне промењиве *godina2d* и вектором вредности зависне промењиве *br_korisnika*.

```
model = LinearRegression()
model.fit(godina2d, br_korisnika)
```

Слика 8.9: конструкција и обучавање модела

Након што је модел обучен, може се користити за предвиђање недостајућих, или/и будућих вредности зависне промењиве у односу на независну. На пример, ако треба извршити предвиђање броја корисника услуге мобилног банкарства за период 2025-2028, поступак са подацима независне промењиве је исти као и приликом обучавања модела (Слика 8.10).

```
godine_buduce = np.array([2025, 2026, 2027, 2028])
godine_buduce = godine_buduce.reshape(-1,1)
```

Слика 8.10: припрема података за предикцију

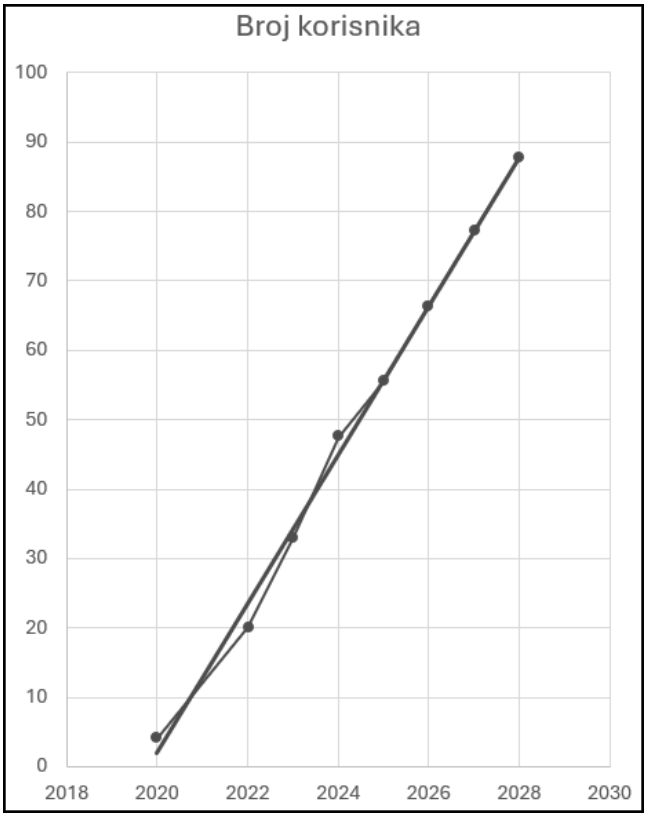
Предвиђање се врши позивом методе *predict* на обученом моделу (Слика 8.11), при чему јој се прослеђују припремљени подаци независне промењиве (*godine_buduce*). Резултат ове функције је предикција (промењива *br_kor_predikcija*) – вектор предвиђања броја корисника за сваку годину дату у матрици улазних података.

```
br_kor_predikcija = model.predict(godine_buduce)
print(br_kor_predikcija)

[55.62857143 66.35714286 77.08571429 87.81428571]
```

Слика 8.11: коришћење регресивног модела у предикцији броја корисника за 2025-28

Визуализацијом регресивне линије, стварних података и података добијених предвиђањем, уочава се да предвиђања немају одступања у односу на регресивну линију (Слика 8.12).



Слика 8.12: графички приказ резултата предвиђања

8.1.2. Вишеструка линеарна регресија

Као што се могло видети у претходном поглављу, линеарна регресија је примењива само у одређивању корелације зависне промењиве са једном независном промењивом. Међутим у пракси је чест случај да на зависну промењиву утиче више независних промењивих. Вишеструка линеарна регресија (енгл. *Multiple Linear Regression - MLR*) представља решење (модел) за случај више независних промењивих и може се представити једначином (Израз 8.7) у којој је *N* број независних промењивих, k_i је коефицијент нагиба регресивне линије у случају i-те независне промењиве, x_i је вредност i-те независне промењиве, фактор *a* одређује тачку пресека функције на *y* оси и ϵ представља грешку која се израчунава на један од два понуђена начина (Изрази 8.4 и 8.5).

$$y = \sum_{i=1}^{N} k_i x_i + a + \epsilon \qquad \text{И.8.7}$$

Промењиве x_i се још називају *регресорима*, а коефицијенти k_i се још називају *регресивни коефицијенти*. Изградња модела вишеструке линеарне регресије започиње откривањем својстава – потенцијалних регресора, који су у значајној позитивној, или негативној корелацији са својством које се посматра као зависна промењива. У следећем кораку одређују се регресивни коефицијенти (k_i) за појединачне линеарне регресивне моделе зависне промењиве за сваку (i-ту) независну промењиву. Ови коефицијенти стрмине се одређују тако да минимизирају грешку ϵ. Савремени статистички алати омогућавају аутоматизована израчунавања свих регресивних параметара.

8.1.2.1. Вишеструка линеарна регресија - пример у Python-у

За пример вишеструке линеарне регресије коришћени су подаци о моторним возилима представљени табеларно (Слика 8.13). Прецизније, представљене су вредности пет својстава: марка возила, модел, запремина мотора, тежина возила и емисија угљен-диоксида коју возило ствара.

	A	B	C	D	E
1	**vozilo**	**model**	**zapremina_motora**	**tezina_vozila**	**CO2**
2	Toyota	Aygo	1000	790	99
3	Mitsubishi	Space Star	1200	1160	95
4	Skoda	Citigo	1000	929	95
5	Fiat	500	900	865	90
6	Mini	Cooper	1500	1140	105
7	VW	Up!	1000	929	105
8	Skoda	Fabia	1400	1109	90
9	Mercedes	A-Class	1500	1365	92
10	Ford	Fiesta	1500	1112	98
11	Audi	A1	1600	1150	99
12	Hyundai	I20	1100	980	99
13	Suzuki	Swift	1300	990	101
14	Ford	Fiesta	1000	1112	99
15	Honda	Civic	1600	1252	94
16	Hundai	I30	1600	1326	97
17	Opel	Astra	1600	1330	97
18	BMW	1	1600	1365	99
19	Mazda	3	2200	1280	104
20	Skoda	Rapid	1600	1119	104
21	Ford	Focus	2000	1328	105

Слика 8.13: преглед својстава различитих врста возила

Задатак је да се направи регресивни модел који ће да одређује емисију угљен диоксида у зависности од тежине возила и запремине мотора. У првом кораку одређују се својства која се укључују у модел. Емисија угљен-диоксида представља зависну промењиву, а тежина возила и запремина мотора представљају независне промењиве.

Python представља имплементационо окружење за регресивни модел, тако да се прво укључују потребни Python ресурси: *pandas* модул за учитавање података из датотека и класа *LinearRegression* из *sklearn.linear_model* модула. У другом кораку (Слика 8.14) се најпре учитава скуп података из CSV датотеке која садржи податке, затим се издвајају својства од интереса: *zapremina_motora* и *tezina_vozila* као независне промењиве у објекат *X* класе *DataFrame*, *CO2* као зависна промењива у објекат *y* класе *DataFrame*

```
df = pandas.read_csv("podaci.csv")
X = df[['zapremina_motora', 'tezina_vozila']]
y = df['CO2']
```

Слика 8.14: Учитавање и припрема података

У следећем кораку (Слика 8.15) се конструише објекат *model* класе *LinearRegression*. Затим се моделу позива метода *fit* којој се прослеђују припремљени подаци, како би се извршила израчунавања свих потребних регресивних параметара ().

```
model = LinearRegression()
model.fit(X.values, y)
```

Слика 8.15: Конструкција и подешавање регресивног модела

Након описане припреме модел је спреман за коришћење. У конкретном случају (Слика 8.16) тражи се од модела да предвиди колику ће емисију угљен-диоксида произвести возило које би имало запремину мотора 2300 cm^3 и тежину 1300 kg (такво возило не постоји у датим подацима)

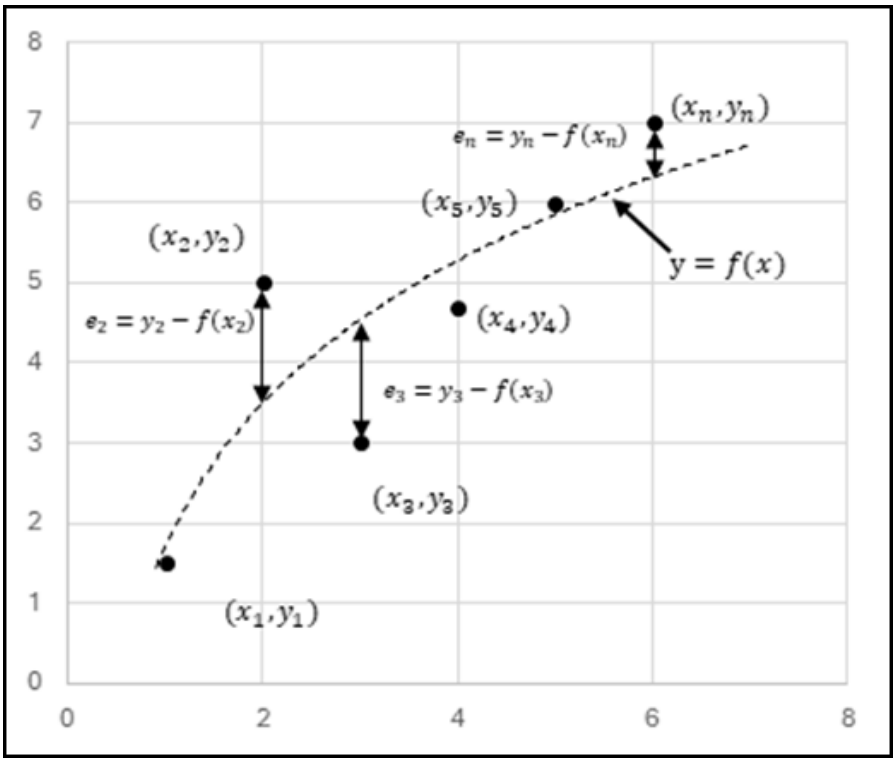
```
predictedCO2 = model.predict([[2300, 1300]])
print(predictedCO2) ⟶ [107.46304306]
```

Слика 8.16: Коришћење регресивног модела

Као резултат резоновања, добијена је предикција емисије угљен диоксида 107.46 за возило које би имало запремину мотора 2300 cm^3 и тежину 1300 kg.

8.1.2. Нелинеарна регресија

Нелинеарна регресија се користи када нелинеарним моделом треба представити сложену везу зависне промењиве са једном или више независних промењивих. На следећем дијаграму је представљен пример модела нелинеарне регресије (Слика 8.17). Као што се види, регресивна линија није линеарна с циљем да се минимизира грешка e_n представљена разликом између стварних вредности зависне промењиве (y_n) у популацији узорака и вредности добијених предикцијом нелинеарног регресивног модела – регресивном функцијом $f(x_n)$.



Слика 8.17: ӣример модела нелинеарне реӣресије

Постоје различите врсте нелинеарних регресивних модела:

- Експоненцијални модел
- Степеновани модел (независна промењива се степенује: $y = ax^k$)
- Модел засићења (npr. $y = \frac{kx}{a+x}$)
- Harmonični opadajući (npr. $y = \frac{a}{1+kx}$)

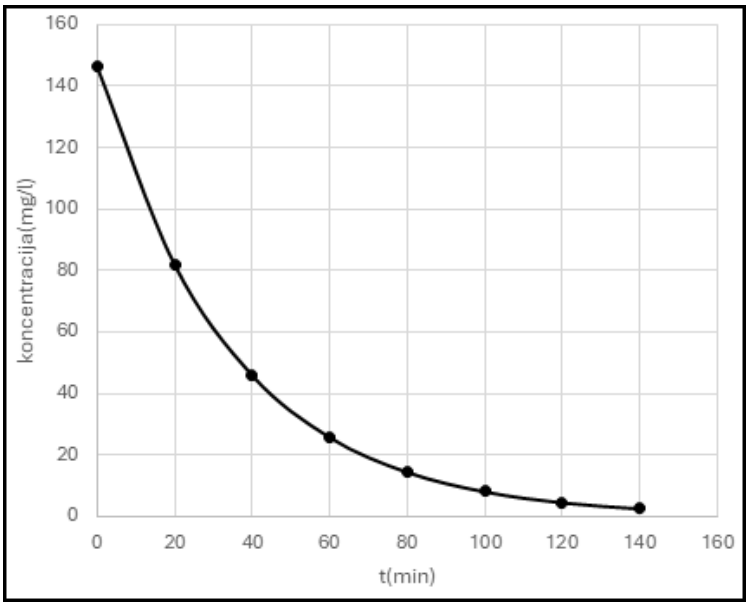
8.1.2.1 Ексӣоненцијални модел

Код експоненцијалног регресивног нелинеарног модела независна промењива је у експоненту (Израз 8.8). Константе модела су представљене са две промењиве *a* и *k*, при чему је друга у експоненту. То значи да у случајевима да је *k* негативна вредност зависне промењиве је опадајућа

$$y = a * e^{kx} \qquad \text{И.8.8}$$

На следећем примеру (Слика 8.18) је приказ регресивне криве експоненцијалног регресивног модела који описује опадање концентрације лека у крви временом. У овом случају вредност коефицијента *k* је негативна (-0.029), тако да је стрмина опадајућа. Вредност

константе *a* одређује пресек регресивне линије на *y* оси. У представљеном примеру то је почетна вредност концентрације лека у крви и износи 146 mg/l (вредности у тренутку *t=0*).



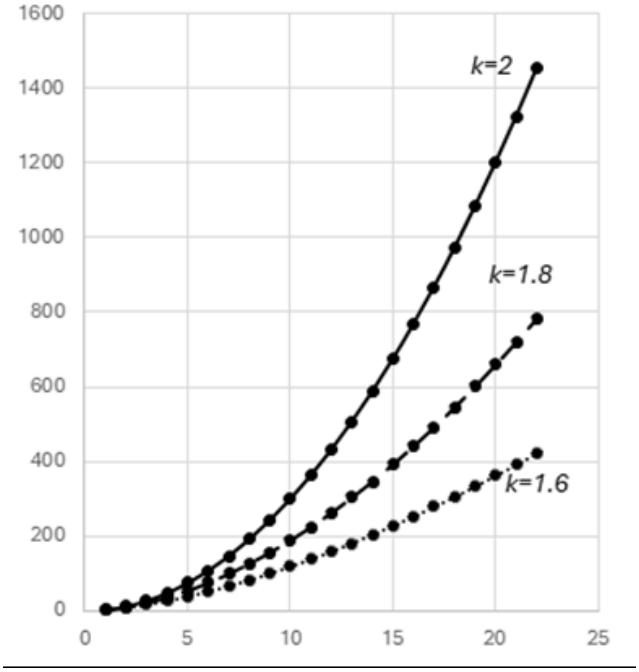
Слика 8.18: пример експоненцијалне нелинеарне регресије

8.1.2.2 Степеновани модел

Код степенованог модела независна промењива се степенује и може се описати следећим функционалним изразом (Израз 8.9) у коме је коефицијент *k* експонент функције, односно константа којом се степенује вредности независне промењиве. Коефицијентом *a* врши се додатна контрола (скалирање) вредности зависне промењиве. Тачка пресека регресивне криве у координатном почетку.

$$y = a * x^k \qquad \text{И.8.9}$$

Степеновани модел приказује експоненцијалну везу независне и зависне промењиве (Слика 8.19). Коефицијентом *k* одређује се нагиб стрмине, тако да повећањем *k* стрмина је већа.



Слика 8.19: стейенови регресивни модел

Експоненцијални модел погодан је за различите примене у пракси. На пример, користи се код моделовања повећања потрошње горива у зависности од брзине возила, нагиба терена, надморске висине и/или густине флуида који испуњава средину у коме се возило креће.

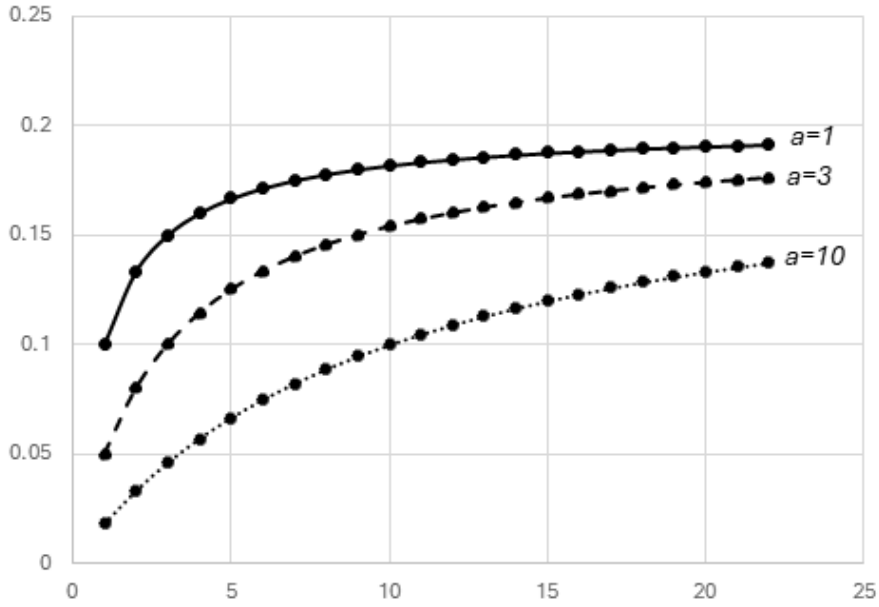
8.1.2.3 Модел засићења

Модел засићења се може описати као количник производа независне промењиве са коефицијентом *k* и њеног збира са константом *a* (Израз 8.10). Може се закључити да је тачка пресека регресивне криве у координатном почетку.

$$y = \frac{kx}{a + x} \qquad \text{И.8.10}$$

Модел засићења приказује нелинеарну везу зависне и независне промењиве такву да са порастом независне промењиве, постоји пораст зависне промењиве, али порастом независне промењиве опада прираштај пораста зависне промењиве (Слика 8.20). Као код линеарних модела, коефицијент *k* је коефицијент који одређује зависност промењивих, док се константом *a* регулише закривљеност регресивне линије. За мање

вредности константе *a* закривљеност (стрмина) је већа, а њеним повећањем долази до *исправљања* регресивне линије. На примеру су дате дасте различите вредности константе (1,3 и 10), док је коефицијент *k = 0.5* исти за сва три случаја.



Слика 8.20: регресивни модел засићења

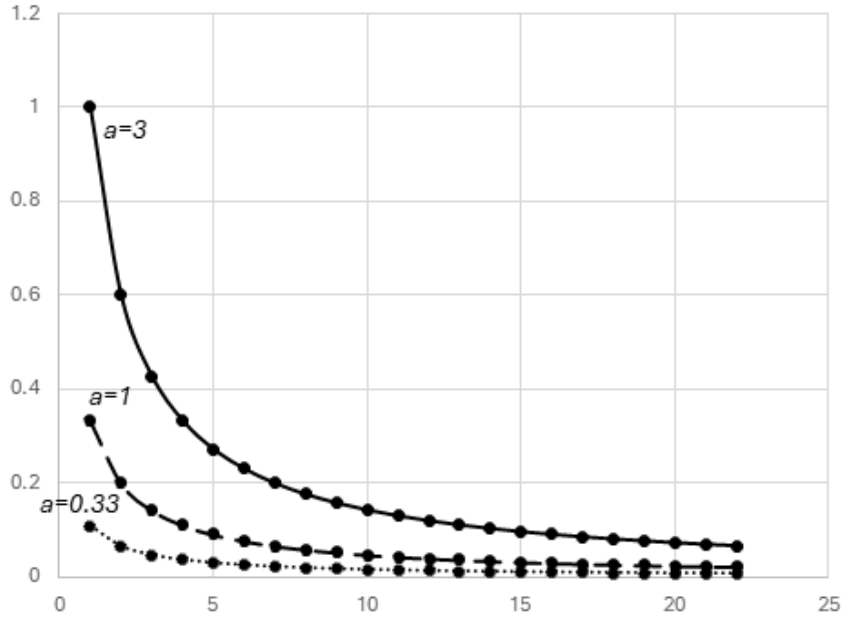
Модели засићења се интензивно користе, на пример, за представљање динамике повећања густине лекова у крви, густине популације организама у неком екосистему, или представљање динамике пораста приноса у зависности од примене минералних ђубрива у пољопривреди.

8.1.2.4 Хармонични опадајући модел

Хармонични опадајући модел се може описати (Израз 8.11) као количник константе *a* и производа *kx* увећаног за један (практично, ради избегавања дељења нулом).

$$y = \frac{a}{1 + kx} \qquad \text{И.8.11}$$

Овај модел изражава везу у којој вредност зависне промењиве опада са порастом вредности независне промењиве, тако да за мање вредности независне промењиве стрмина опадајуће криве је већа и обрнуто (Слика 8.21).



Слика 8.21: реīресивни хармонични оīадајући модел

Пример коришћења хармоничног опадајућег модела је у системима експлоатације ограничених ресурса. На пример, то могу да буду налазишта нафте, гаса или различитих руда. Ова налазишта се временом троше тако да и производња опада. Хармонична опадајућа регресија омогућава моделовање таквих система у циљу предвиђања исплативости коришћења ресурса и планирања правовременог напуштања налазишта када експлоатација престане да буде исплатива.

8.1.2.5 Полиномска реīресија

Полиномска регресија је врста нелинеарне регресије и омогућава формирање нелинеарног модела са једним ограничењем – регресивна функција је полином одређеног степена. За разлику од вишеструке линеарне регресије која оперише са више регресора (независних промењивих), код полиномске регресије постоји један регресор у више монома. Модел полиномске регресије (Израз 8.12) је сличан изразу за вишеструку линеарну регресију са разликом да (уместо x_i, *i*-ти регресор) садржи x^i (*i*-ти степен од *x*).

$$y = \sum_{i=1}^{n} k_i x^i + a + \epsilon \qquad \text{И.8.12}$$

Ако би се мономи ($k_i x^i$) разматрали као засебне промењиве (x, x^2,...,x^n као регресори), полиномска регресија се своди на вишеструку линеарну регресију. При томе обавезно ограничење је да број узорака у популацији мора бити већи од броја монома у регресивној функцији.

8.1.2.5.1. Коефицијент детерминације (R^2)

Као што Пирсонов коефицијент одређује да ли линеарна регресија одговара за моделовање података, тако и код полиномске регресије постоји *коефицијент детермниације* који показује да ли полиномска регресија одговара за моделовање података који се обрађују. Као и код Пирсоновог коефицијента, то врши се одређивањем везе регресора (независних промењивих) и зависне промењиве. Потребни коефицијент детерминације, који се означава као R^2 (*R-Squared*) израчунава се следећом једначином (Израз 8.13) у којој фигурише количник SS_r (од енгл. *Sum of squares residual*) и SS_t (од енгл. *Sum of squares total*).

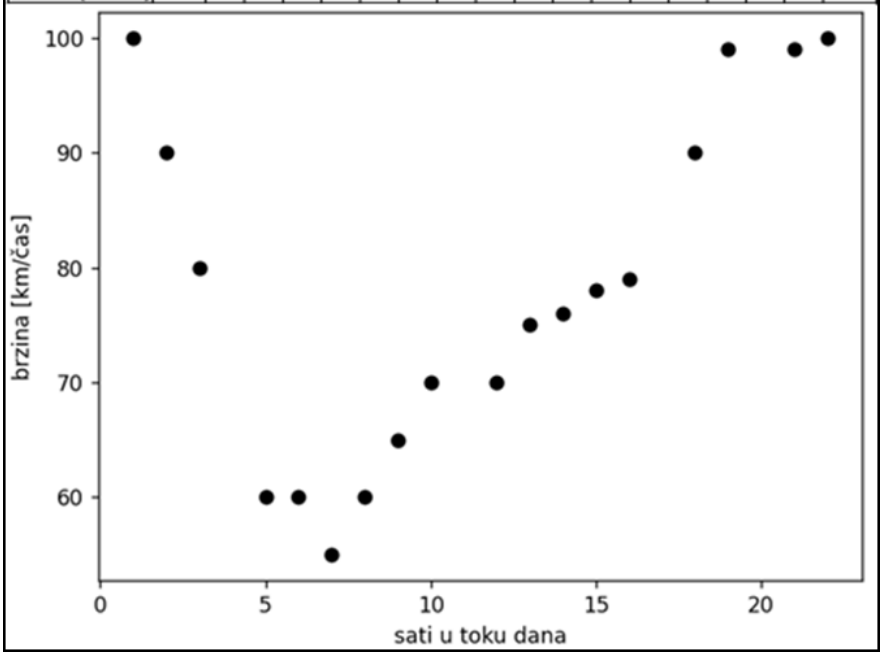
$$y = 1 - \frac{SS_r}{SS_t} = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2} \qquad \text{И.8.13}$$

У бројиоцу овог количника се налази *сума квадратних разлика* – SS_r чине квадрати разлика стварне (измерене) вредности зависне промењиве (y_i) и њене очекиване (израчунате) вредности ($\hat{y}_i$). У имениоцу овог количника се налази вредност позната као *тотална сума квадрата* – SS_t, коју чине квадрати разлика стварне вредности зависне промењиве (y_i) и просечне вредности израчунате за целу популацију узорака ($\bar{y}$). Вредност коефицијента детерминације се креће у интервалу [0,1], а интерпретира се тако да ако је вредност блиска 0, онда не постоји никаква веза (однос) између независне и зависне промењиве. Иначе, ако је вредности блиска јединици, онда постоји јака веза независне и зависне промењиве.

8.1.2.5.2. Пример полиномске регресије у Python-у

За пример полиномске регресије коришћени су подаци о сатима и просечно измереној брзини моторних возила која пролазе контролни пункт, представљени табеларно и графички (Слика 8.22). Визуелно се лако уочава да су највеће измерене брзине у току ноћних часова (пре и после поноћи), најмање у јутарњим сатима (када је већи интензитет саобраћаја), док су у поподневним сатима брзине возила око просека (~78km/h).

vreme(h)	1	2	3	5	6	7	8	9	10	12	13	14	15	16	18	19	21	22
brzina(km/h)	100	90	80	60	60	55	60	65	70	70	75	76	78	79	90	99	99	100



Слика 8.22: Подаци о сатима и просечним брзинама пролазака возила

Након учитавања података из *CSV* датотеке у *DataFrame* објекат *df*, подаци се припремају тако што се одвајају узорци независне и зависне промењиве у посебне низове *x* и *y* (Слика 8.23).

```
df = pandas.read_csv('brzine_po_casovima.csv')
x = df['vreme(h)']
y = df['brzina(km/h)']
```

Слика 8.23: учитавање и припрема података

Затим следи креирање полиномских регресивних модела 2.,3.,4., и 5. степена (Слика 8.24). Искоришћена је *numpy.polyfit* функција која прилагођава полином задатог степена подацима који су прослеђени као аргументи функције – у примеру то су низови узорака независне и зависне промењиве. Задати степен полинома је трећи параметар функције. Након прилагођавања полинома подацима, коришћењем класе *numpy.poly1d* се креирају полиномски регресивни модели. Ради упоређивања, креирана су четири модела чији су полиноми различитих степена (*model2*, *model3, model4, model5*).

```
model2 = numpy.poly1d(numpy.polyfit(x, y, 2))
model3 = numpy.poly1d(numpy.polyfit(x, y, 3))
model4 = numpy.poly1d(numpy.polyfit(x, y, 4))
model5 = numpy.poly1d(numpy.polyfit(x, y, 5))
```

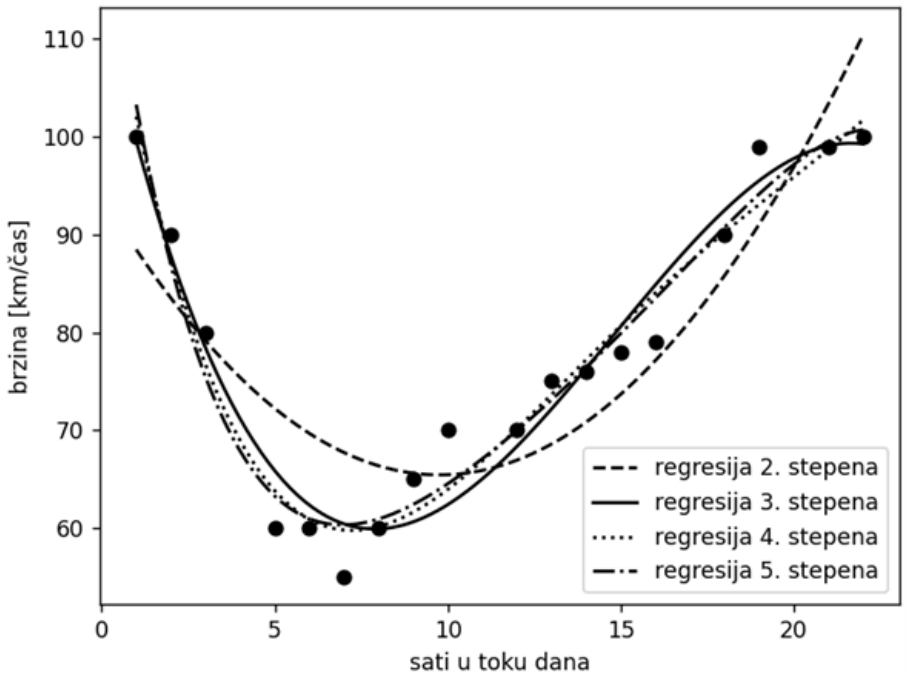
Слика 8.24: креирање полиномских регресивних модела 2.,3.,4., и 5. степена

У следећем кораку се врши евалуација модела коефицијентом детерминације (R^2). За ту сврху користила се *r2_score* функција из модула *sklearn.metrics* (Слика 8.25). Вредности коефицијента детерминације су најбољи за полиномске регресивне моделе четвртог и петог степена (>0.95), нешто мањи за модел трећег степена, а најлошији за модел другог степена (~0.76).

```
print(r2_score(y, model2(x)))   0.759777160189589
print(r2_score(y, model3(x)))   0.9432150416451026
print(r2_score(y, model4(x)))   0.9542030834699506
print(r2_score(y, model5(x)))   0.9568460139893519
```

Слика 8.25: евалуација модела коефицијентом детерминације

Визуализација података и тестираних модела представљена је на комбинованом дијаграму (Слика 8.26). На приказу је лако уочљиво да полиномски регресивни модел другог степена најмање одговара подацима, док су разлике модела четвртог и петог степена незнатне. Наведено значи да у даљој употреби могу да се користи било који од ова два модела.



Слика 8.26: Визуализација података и полиномских регресивних модела

Након избора модела, може се прећи на његово коришћење. Рецимо да је потребно предвидети колике су очекиване просечне брзине возила у 17, 23 и 24 часа, пошто ти подаци недостају. Изабраном моделу се проследе вредности независне промењиве (Слика 8.27) и као резултат добија се предвиђање - вредности зависне промењиве. У коду се види да сеод дефинисаних модела (*model2*, *model3, model4, model5*) прави листа *modeli*. Сати за предикцију су смештени у листу *sati*. За прихват резултата формирана је листа *rezultati*, а поред ње постији и листа *zaglavlje* за лабеле заглавља резултујуће табеле. Конструкција са угњежденим *for* петљама за сваки сат из листе *sati* узима један по један модел из листе *modeli*, прослеђујући им сат (промењива *sat*) као аргумент и прихвата резултат у локалну промењиву *rez*. Податак *rez* се затим, заједно са називом модела (форматирани стринг) и сатом за који се врши предикција, користи за креирање листе за испис реда у резултујућој табели. Та листа података се затим додаје у листу *rezultati*. Након извршених (3x4=12) предикција, од листе *rezultati* инстанцира се *DataFrame* објекат *df* (кроз конструктор се прослеђује и листа *zaglavlje*), чијим штампањем се добија резултујућа табела.

```
modeli = []
modeli.append(model2)
modeli.append(model3)
modeli.append(model4)
modeli.append(model5)
sati = [17,23,24]
zaglavlje = ['model','vreme(sat)','brzina(km/sat)']
rezultati=[]
i=0
for sat in sati:
    for model in modeli:
        rez = model(sat)
        rezultati.append([f'model{i%4+2}',sat,rez])
        i+=1
df = pandas.DataFrame(rezultati, columns=zaglavlje)
print(df)
```

	model	vreme(sat)	brzina(km/sat)
0	model2	17	81.175651
1	model3	17	88.873313
2	model4	17	87.210975
3	model5	17	87.137407
4	model2	23	118.068543
5	model3	23	98.095489
6	model4	23	105.034533
7	model5	23	100.319399
8	model2	24	126.318653
9	model3	24	95.452087
10	model4	24	108.927593
11	model5	24	97.711221

Слика 8.27: предикција брзине коришћењем различитих полиномских модела

На основу представљених предикција може се уочити да се предикције просечне брзине за 17 часова незнатно разликују, док су разлике за 23 и 24 часа знатно веће. Сличне су предикције модела 3 и 5, модел 2 има знатна одступања за оба сата, док модел 4 у односу на моделе 3 и 5 има знатну разлику предикције за 24 часа.

Закључак је да може да се догоди да модели дају сличне или различите предикције услед специфичности дистрибуције података (узорака у популацији). Модели који имају сличне предикције у неким сегментима узорака, у другима могу испољити значајну разлику. Као што је у примеру представљено, у пракси би требало перманентно евалуирати моделе у складу са новим, или ажурираним подацима који се користе у предикцијама.

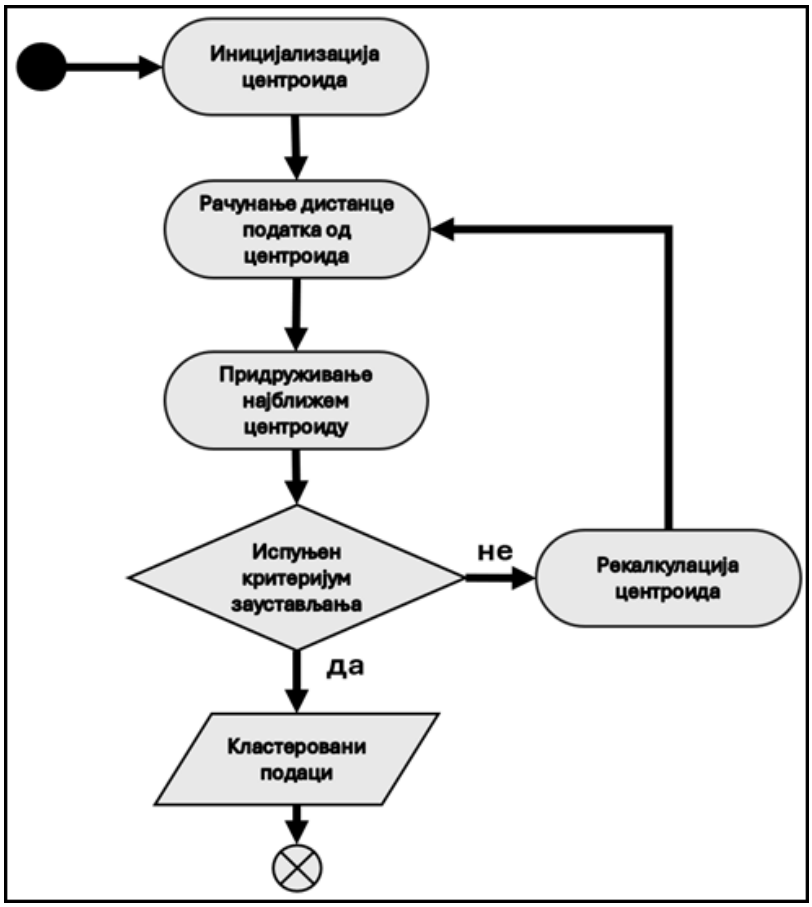
8.2. Кластеровање

Кластеровање представља груписање података у неименоване скупове података без предефинисаног критеријума. Ови скупови се називају *кластерима*, а процес њиховог добијања је *кластеровање*. Кластеровање се примењује када не постоји претходно знање о природи и дистрибуцији података. Кластерском анализом се настоје открити сличности међу подацима на основу којих би се извршило њихово груписање. Кластеровање као процес се још назива и *не надгледано учење* (енгл. *Unsupervised learning*). Кластеровање се врши према задатим алгоритмима, на основу којих се дели у пет врста:

1. Кластеровање засновано на центроидима
2. Кластеровање засновано на густини (енгл. *Density-based clustering*)
3. Кластеровање засновано на везама (енгл. *Connectivity-based clustering* или *Hierarchical clustering*)
4. Кластеровање засновано на дистрибуцији (енгл. *Distribution-based clustering*)
5. Кластеровање засновано на расплинутој логици (енгл. *Fuzzy clustering*)

8.2.1. Кластеровање засновано на центроидима

Код кластеровања заснованог на центроидима (енгл. *centroid-based clustering*) користе се непостојеће (фиктивне) вредности података зване *центроиди* како би се пронашло најоптималније груписање података, које одговара њиховој дистрибуцији у популацији. Број центроида је предефинисан и одговара броју кластера у које треба да се групишу подаци. Најпознатији алгоритам за кластеровање засновано на центроидима је алгоритам *K* – средина (енгл. *K-means algorithm*). У називу алгоритма средине (*means*) су центроиди, а број *K* представља број центроида, односно кластера. Центроиди се иницијално одређују, најчешће насумично (Слика 8.28). Затим се рачунају еуклидске дистанце података од тих центроида (корак 2). Подаци се затим придружују центроиду до кога имају најмању удаљеност (корак 3).



Слика 8.28: алгоритам K - средина

Док год критеријум заустављања алгоритма није испуњен, врши се рекалкулација центроида. Рекалкулацијом центроида добијају се нове координате центроида као просеци x и y координата свих узорака (података) које посматрани кластер садржи. Након рекалкулације се понављају кораци два и три.

Критеријум заустављања центроида је стабилизација вредности укупне суме квадрата растојања података од центроида унутар кластера (енгл. *Within-Cluster Sum of Squares - WCSS*). Ова величина се рачуна на основу једначине (Израз 8.14), у којој је K укупан број кластера, са d су представљени подаци унутар кластера C_i чији је центроид представљен симболом μ_i. Еуклидска дистанца податка је представљена изразом $\|d - \mu_i\|$, а рачуна се као квадрат разлика x и y координата податка d и центроида μ_i (Питагорина теорема).

$$WCSS = \sum_{i=1}^{K} \sum_{d \in C_i} \|d - \mu_i\|^2 \qquad \text{И.8.14}$$

Алгоритам *K* – средина се често користи као синоним за кластеровање засновано на центроидима. Овај алгоритам има и модификацију (енгл. *K-Medoids Algorithm*) у којој се уместо фиктивних центроида користе подаци – узорци из популације која се кластерује. Основа кластеровања је иницијално дељење на основу предефинисаног броја центроида тако да се као синоним неретко појављује и назив кластеровање методом дељења (енгл. *Partition methods*).

8.2.1.1 Кластеровање текстуалних докумената – пример у Python-у

У следећем примеру представљено је кластеровање текста. Кластеровање текста (докумената) се користи с циљем аутоматске организације докумената, екстракције наслова и откривању информација. Након припреме текстуалних докумената као што је то објашњено у поглављу *Припрема текстуалних података* на документа се примењује једна од метода за кластеровање. У примеру је коришћено кластеровање алгоритмом *K* – средина скупа докумената представљених промењивом *destiled_srb_txt_latin* (Слика 8.29). Креирање модела за кластеровање врши се позивом конструктора класе *KMeans* из модула *sklearn.cluster*, који као параметар очекује број кластера, који се прихвата кроз параметар *n_clusters* (у конкретном случају три). Обучавање модела (промењива *kmeans*) врши се методом *fit*, којој се прослеђују подаци – у примеру то су докумената трансформисана у *TF-IDF* векторе. Сваки вектор се састоји из уређених парова бројева који представљају индексе докумената и индексе речи. Сваком пару је придружена и вредност TF-IDF фактора.

Као исход обучавања, модел је доделио сваком документу ознаку кластера коме документ припада. Другим речима, процесом обучавања извршено је и кластеровање. Ознаке кластера су целобројне и имају вредности у интервалу [0,*N-1*], при чему је *N* предефинисани број кластера. Ове ознаке садржане су у обученом моделу (промењива *kmeans*) као податак *labels_*.

```
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(destiled_srb_txt_latin)
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
cluster_labels = kmeans.labels_
for i, label in enumerate(cluster_labels):
    print(f"Text> {serbian_text_data[i][0:55]} -> Cluster: {label}")
```

Слика 8.29: кластеровање текста алгоритмом K – средина

У примеру је представљена и циклична структура, која омогућава приказ резултата кластеровања. У добијеном приказу представљено је првих 55 карактера сваког документа и ознака кластера ком документ припада (Слика 8.30). Уочљиво је да је модел препознао узајамну сличност првог, трећег и петог документа (култура, књижевност, историја) и придружио их је истом кластеру (ознака 1). Модел је такође препознао да се други (географија) и четврти (здравство) документи разликују од осталих, тако да припадају посебним кластерима (ознаке 0 и 2).

```
Text> У време обновљања уметности, дефиниције уметности теже  -> Cluster: 1
Text> Кључевице је место у Општини Кључ на излазу из града у  -> Cluster: 0
Text> У историји културе, као и у историји уметности, термин  -> Cluster: 1
Text> Светска здравствена организација објавила је нови прегл -> Cluster: 2
Text> Сматра се да реч 'у' има највише значења у српском јези -> Cluster: 1
```

Слика 8.30: приказ резултата кластеровања текстуалних докумената

Пример показује да се могу кластеровати различите врсте података. Нумерички подаци не захтевају посебну припрему пре кластеровања (најчешће се примењује скалирање). Остале врсте података захтевају да се изврши трансформација како би се добила нумеричка својства која се могу кластеровати. У претходном примеру искоришћена је *TF-IDF* векторизација, којом се текстуални садржај докумената представио векторима индекса докумената и њиховог текстуалног садржаја, уз придружене вредности *TF-IDF* фактора.

8.2.2. Кластеровње засновано на густини

Кластеровње засновано на густини (енгл. *Density-based clustering*) има за циљ груписање података такво да да се кластери креирају у простору података тамо где су највеће густине података. Последично, код кластеровања заснованог на густини не може се предефинисати број

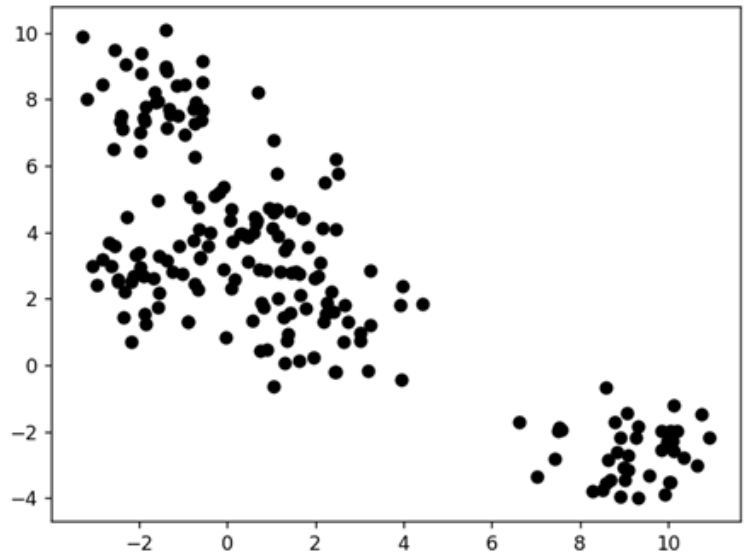
кластера, већ је то резултат обучавања модела. Кластери тако немају центроиде, већ ту улогу преузимају стварни подаци који се налазе у скупу који се кластерује. Постоје различити алгоритми за кластеровање засновано на густини, од којих су најпознатији *DBSCAN* и *Mean-Shift* алгоритам. Ови алгоритми су успешнији ако у простору података који се кластерују постоје концентрације (груписања) података. У основи, оба алгоритма настоје да препознају регионе великих густина података. Ово се постиже на основу мерења броја података (узорака, или објеката) блиских задатој тачки. Та тачка је иницијално одабрани податак.

За представљање рада ова два алгоритма генерисани су синтетички подаци који образују просторе концетрација података (мехурове). За ту намену је коришћена функција *make_blobs* из модула *sklearn.datasets* (Слика 8.31).

```
X, _ = make_blobs(n_samples=200, centers=5, random_state=0)
```

Слика 8.31: Генерисање синтетичких података

Добијен је скуп од 200 узорака просторно дистрибуираних као што је представљено (Слика 8.32).



Слика 8.32: изглед синтетичког скупа података

8.2.2.1. Кластеровање Mean-Shift алгоритмом са примером у Python-у

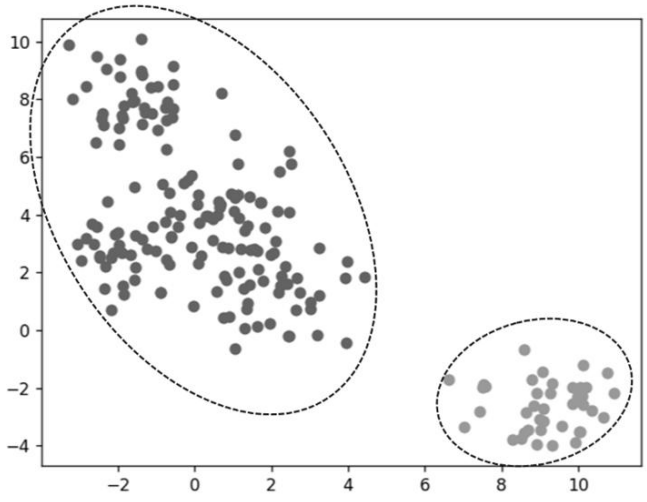
Код *Mean-Shift* алгоритма иницијално сви подаци представљају центроиде. Другим речима, сваки податак представља један иницијални кластер. Затим се за сваки центроид рачуна средња вредност (*mean* фаза) података који се налазе у одређеном радијусу. Након тога се бира нови центроид такав да буде најближи израчунатој средњој вредности (*shift* фаза). Алгоритам конвергира и зауставља се када нема више промене центроида. Резултат овог итеративног процеса су подаци који представљају центроиде кластера, којима се затим придружују преостали подаци.

У следећем примеру представљено је кластеровање синтетички генерисаног скупа података *Mean-Shift* алгоритмом. Након конструкције модела, прелази се на обучавање позивом методе *fit_predict* и прослеђивањем скупа за обуку (Слика 8.33). Ознаке кластера се добијају методом *unique*.

```
model = MeanShift()
yhat = model.fit_predict(X)
clusters = unique(yhat)
```

Слика 8.33: креирање модела и кластеровање Mean-Shift алгоритмом

Исход кластеровања *Mean-Shift* алгоритмом (Слика 8.34) су два кластера. Интересантно да је велика група података у горњем левом делу дијаграма представљена као један кластер иако се визуално може уочити барем још један кластер.



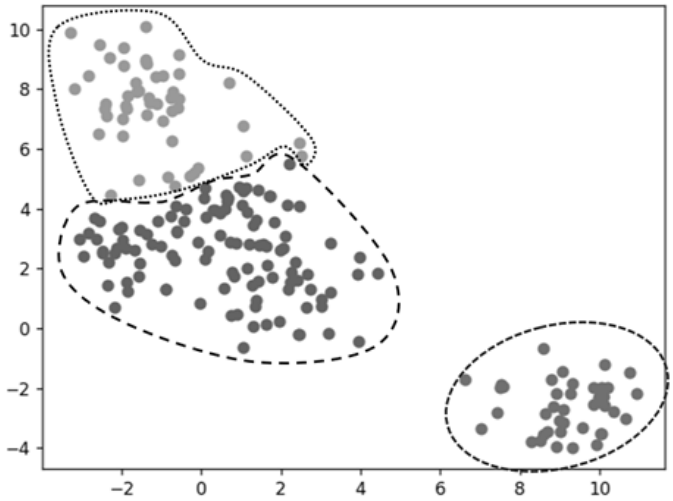
Слика 8.34: резултат кластеровања Mean-Shift алгоритмом

Ако се деси овакав случај, може да се изврши додатна калибрација алгоритма на основу података коришћењем *estimate_bandwidth* функције из *sklearn.cluster* модула (Слика 8.35). Резултат функције је реалан број - вредност радијуса у коме се налазе подаци који треба да буду укључени у израчунавање средње (*mean*) вредности података. Овај параметар назван *оцсеī* се затим прослеђује као параметар (*bandwidth*) конструктору модела.

```
bandwidth = estimate_bandwidth(X, quantile=0.2, n_samples=200)
model = MeanShift(bandwidth=bandwidth)
yhat = model.fit_predict(X)
```

Слика 8.35: ūобољшавање рада Mean-Shift алīоритма

Исход кластеровања модификованим *Mean-Shift* моделом је унапређен (Слика 8.36). У конкретном случају измерена вредност *оцсеīа* од ~2.68 као резултат кластеровања дала је три уместо два кластера као што је то било у случају недефинисаног радијуса.



Слика 8.36: резулūаū класūеровања модификованим Mean-Shift моделом

Поред наведеног, може се уочити да су сви подаци кластеровани, односно да не постоји податак који није распоређен у неки од кластера.

8.2.2.1. Класūеровање DBSCAN *алīориūмом – ūример у Python-у*

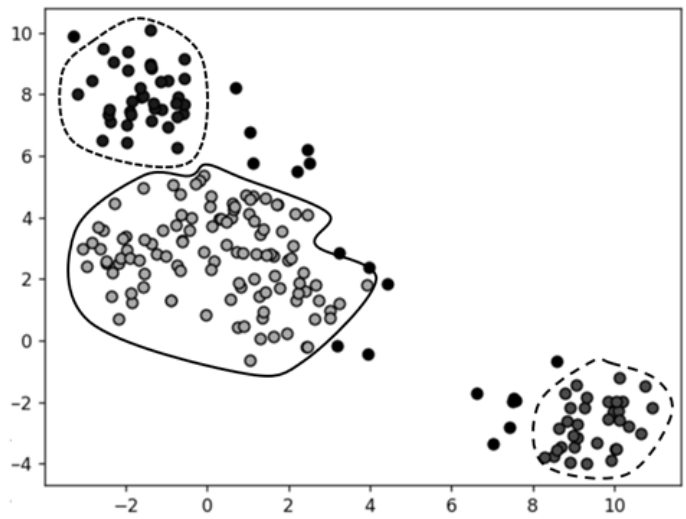
Код *DBSCAN* алгоритма постоје два критеријума, а то су *суседсūво* – односно максимални радијус задате тачке, и минималан број података - суседа који треба да буду у задатом радијусу да би се формирао кластер. Следећи *Python* скрипт (Слика 8.37) представља конструкцију и

кластеровање синтетичког скупа података *DBSCAN* алгоритмом. Наведени параметри неопходни за рад алгоритма прослеђују се конструктору као промењиве *eps* (максимални радијус задате тачке) и *min_samples* (минималан број података - суседа који треба да буду у радијусу *eps* да би се формирао кластер). Обучавање модела и кластеровање врши се позивом *fit* функције.

```
db = DBSCAN(eps=1, min_samples=8).fit(X)
```

Слика 8.37: креирање модела и кластеровање DBSCAN *алгоритмом*

Исход кластеровања *DBSCAN* алгоритмом (Слика 8.38) су три кластера. За разлику од *Mean-Shift* алгоритма, код *DBSCAN* алгоритма подаци могу да буду не увршћени у кластере. На представљеном примеру значајан број података нема дефинисану припадност кластеру. Овај проблем се решава променом параметара алгоритма (*eps* и *min_samples*), чиме алгоритам мења број кластера, као и број података који нису увршћени у меки од њих.



Слика 8.38: резултат кластеровања DBSCAN алгоритмом

Закључак у вези представљених алгоритама је да је код *Mean-Shift* алгоритма сви подаци су увршћени у кластере, док код DBSCAN алгоритма то неретко није случај. Оба алгоритма могу да се прилагођавају подацима који треба да се кластерују. Код *Mean-Shift* алгоритма то се врши анализом скупа података коришћењем *estimate_bandwith* функције. Добија се параметар назван *оцеј*, који се прослеђује *Mean-Shift* моделу алгоритма пре обучавања и кластеровања. Код DBSCAN алгоритма то су два

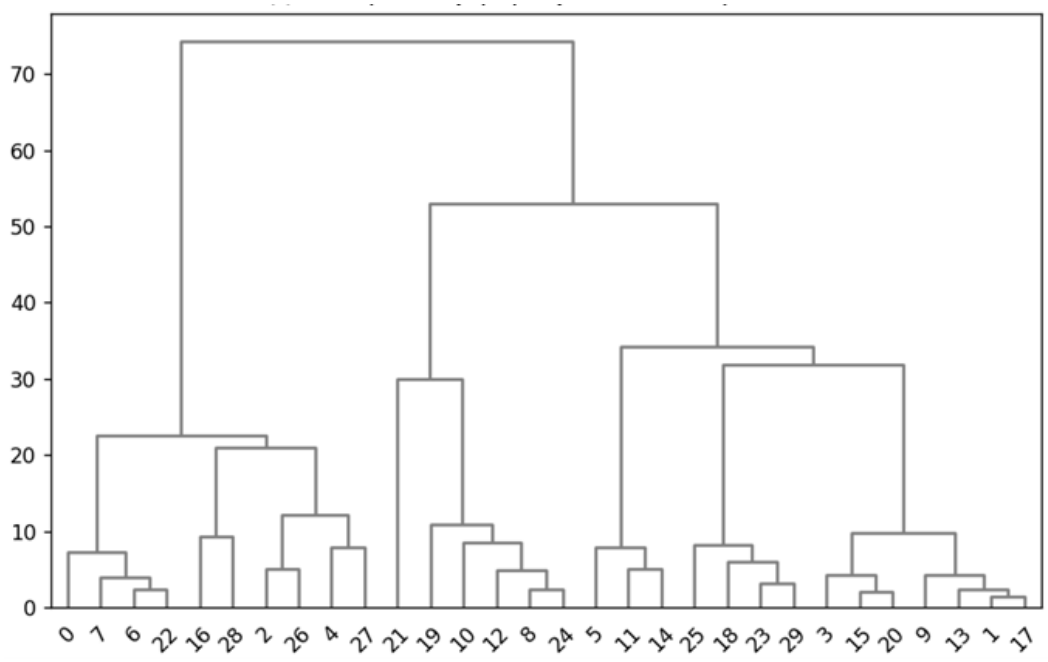
параметра. Први је радијус око података који се разматра као кандидат за центроид кластера. Други параметар је минимални δброј узорака да δи се формирао потенцијални кластер.

8.2.3. Кластеровње засновано на везама

Кластеровње засновано на везама (енгл. *connectivity-based clustering*), још се назива хијерархијско кластеровање. Код кластеровња заснованог на везама постоје два основна принципа на којима раде алгоритми. Један је принцип удруживања (агломерација), при чему на основу задатог критеријума долази до удруживања (агломерације) података, који тако формирају кластере. Још је назван принцип одоздо на горе (енгл. *Bottom-up* принцип). Иницијално сваки појединачни податак формира један кластер, а они се даље удружују и чине веће кластере. Процес се завршава када се испуни један од задатих услова за завршетак рада алгоритма (на пример δброј кластера). Други је принцип дељења (дивизиони), који формира иницијални кластер од свих података, а затим их дели на основу задатог критеријума на све мање и мање кластере све док се не испуни један од задатих услова за завршетак рада алгоритма.

Без обзира на принцип рада алгоритма, резултат процеса кластеровања је хијерархијска структура која приказује начин удруживања / дељења кластера од почетка до краја рада алгоритма. Из тог разлога се ово кластеровање још назива и хијерархијским.

На следећој илустрацији је представљен дендограм (стаδласти приказ) хијерархијског кластеровања (Слика 8.39). На *x* оси представљени су објекти који се кластерују. Вертикална оса представља величину разлике између објеката измерену на основу критеријума удруживања / дељења, односно висина спајања показује сличност између објеката унутар кластера. Тако су при дну спајања најсличнијих објеката. Порастом висине спајања сличност објеката у кластеру опада.



Слика 8.39: дендограм хијерархијског кластеровања

Критеријум удруживања / дељења могу бити:

1. Минимизација варијансе - минимизација суме квадрата разлика по кластерима (познат под називом *Ward* критеријум),
2. Максимум – минимизација максималних дистанци парова узорака у кластерима,
3. Просечна повезаност – минимизација средњих дистанци парова узорака у кластерима и
4. Појединачна повезаност – минимизација најближих (најсличнијих) парова узорака (по вредностима) у кластерима

Пред наведених постоје и други критеријуми удруживања / дељења код хијерархијског кластеровања. Избором критеријума мења се начин формирања кластера. Другим речима, без обзира на број кластера, њихов садржај ће бити различит променом критеријума удруживања / дељења. Број итерација у процесу удруживања / дељења зависи од предефинисаног броја кластера у које треба да се групишу подаци.

8.2.3.1. Пример кластеровања дељењем

За демонстрацију кластеровања дељењем генерисан је синтетички скуп података који садржи 30 узорака, са пет центара Гаусове дистрибуције са

стандардном девијацијом 10 (секција 8.2.2). Представљена је имплементација функције која врши кластеровање дељењем (Слика 8.40). Названа је *klasterovanje_deljenjem* има два параметра – *podaci* (скуп података који треба да се кластерује) и *broj_klastera* (број кластера у које подаци треба да се групишу. Функција враћа листу кластера, представљену промењивом *klasteri*. Као што се види у коду, иницијални кластер обухвата све податке. Дељење се врши у *while* петљи док се не формира број кластера задат параметром *broj_klastera*. Избор кластера за дељење врши се функцијом *najveci* којој се прослеђује листа кластера, а функција враћа кластер са највише података (промењива *klaster_za_deljenje*). Тај кластер се затим избацује из листе *klasteri*, и коришћењем *KMeans* алгоритма (секција 8.2.1.) дели на два кластера (*klaster1* и *klaster2*). Овај алгоритам ће кластере увек поделити на исти начин захваљујући постављеној вредности промењиве *random_state*. Нова два калстера се затим додају у листу кластера. Након постизања задатог броја кластера, функција враћа листу кластера као коначан резултат кластеровања дељењем.

```
def klasterovanje_deljenjem(podaci, broj_klastera=3):
    klasteri = [podaci]
    while len(klasteri) < broj_klastera:
        klaster_za_deljenje = najveci(klasteri)
        klasteri.remove(klaster_za_deljenje)

        kmeans = KMeans(n_clusters=2, random_state=42).fit(klaster_za_deljenje)
        klaster1 = klaster_za_deljenje[kmeans.labels_ == 0]
        klaster2 = klaster_za_deljenje[kmeans.labels_ == 1]

        klasteri.extend([klaster1, klaster2])
    return klasteri
```

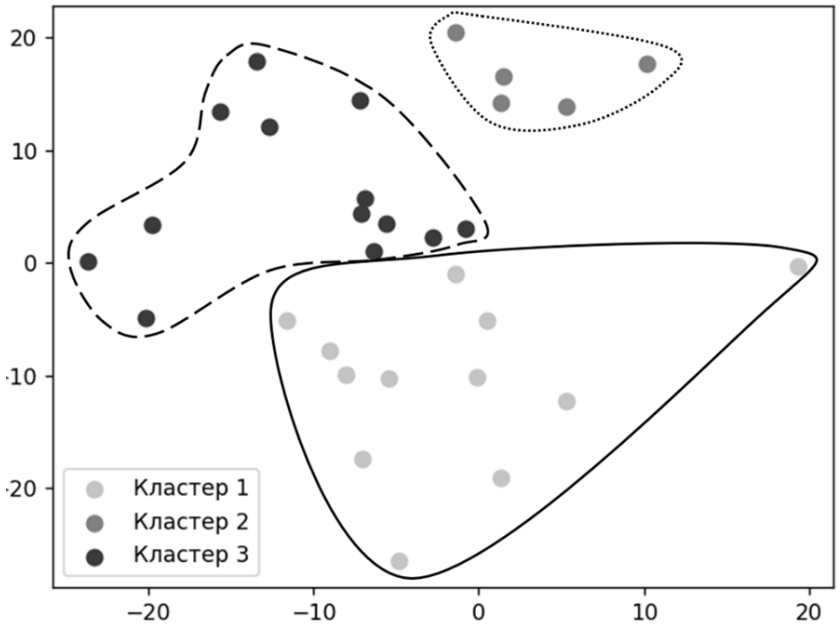
Слика 8.40: имплементација кластеровања дељењем

Као што је поменуто избор кластера за дељење врши се функцијом *najveci* (Слика 8.41) којој се прослеђује листа кластера, а функција враћа кластер са највише података. У *for* петљи се упоређују кластери према броју података који садрже, с циљем добијања индекса кластера (промењива *inajveci*) који садржи највише података. Функција враћа кластер са индексом *inajveci*.

```
def najveci(klasteri):
    inajveci = 0
    for i in range(len(klasteri) - 1):
        if len(klasteri[i]) > len(klasteri[i+1]):
            inajveci = i
    return klasteri[inajveci]
```

Слика 8.41: функција najveci која одређује кластер за дељење

Резултат хијерархијског кластеровања дељењем представљен је на следећем дијаграму (Слика 8.42).



Слика 8.42: резултат хијерархијског кластеровања дељењем

8.2.3.2. Пример кластеровања удруживањем

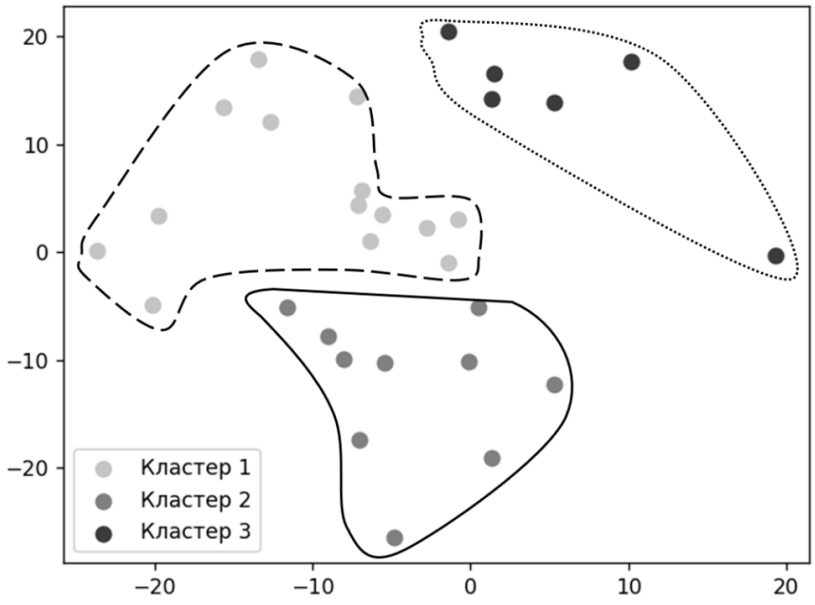
За демонстрацију кластеровања удруживањем искоришћен је исти скуп података као у претходном примеру: 30 узорака, са 5 центара Гаусове дистрибуције са стандардном девијацијом 10 (секција 8.2.3.1). За креирање модела коришћена је класа *AgglomerativeClustering* из модула *sklearn.cluster* (Слика 8.43), којој се прослеђује захтевани број кластера (3). *AgglomerativeClustering* модел се обучава и кластерује податке позивом *fit_predict* функције, којој се прослеђује скуп података за кластеровање, а то је и у овом случају листа названа *podaci_za_deljenje*. Резултат *fit_predict* функције је низ лабела (промењива *oznake_klastera*) код кога индекс показује позицију податка у скупу, а лабела указује ком кластеру податак

припада. На пример, у низу [*1 1 2 0* ...] први и други податак припадају 2. кластеру, трећи податак припада трећем, а четврти податак првом кластеру.

```
model = AgglomerativeClustering(n_clusters=3)
oznake_klastera = model.fit_predict(podaci_za_deljenje)
```

Слика 8.43: кластеровање удруживањем коришћењем AgglomerativeClustering класе

Резултат хијерархијског кластеровања дељењем представљен је на следећем дијаграму (Слика 8.44). За исти скуп података, представљена кластеровања дељењем и удруживањем дају сличну структуру кластера. У конкретном случају поклапање је ~96.7%, што значи да имплементација критеријума дељења (функција *najveci*), представљена у претходном поглављу (секција 8.2.3.1) има понашање слично имплементацији алгоритма кластеровања код *AgglomerativeClustering* класе.



Слика 8.44: резултат хијерархијског аглоративног кластеровања удруживањем

Хијерархијско кластеровање је за велике скупове података је захтевно у смислу ангажовања процесора и меморије. Обзиром на ову чињеницу, приликом избора алгоритма за хијерархијско кластеровање, потребно је извршити процену ефикасности за задате скупове података који се кластерују.

8.2.4. Кластеровање засновано на дистрибуцији

Кластеровање засновано на дистрибуцији (*distribution-based clustering*), као критеријум за груписање података користи расподелу вероватноће у скупу података. Опште прихваћен алгоритам је такозвани модел Гаусове мешавине (енгл. *Gaussian Mixture Model - GMM*). Овај модел претпоставља да су све тачке података генерисане из мешавине коначног броја Гаусових расподела са непознатим параметрима. Вероватноће се код *GMM* модела добијају из коваријационих матрица које детерминишу вероватноћу појављивања парова промењивих код података у истој групи (кластеру).

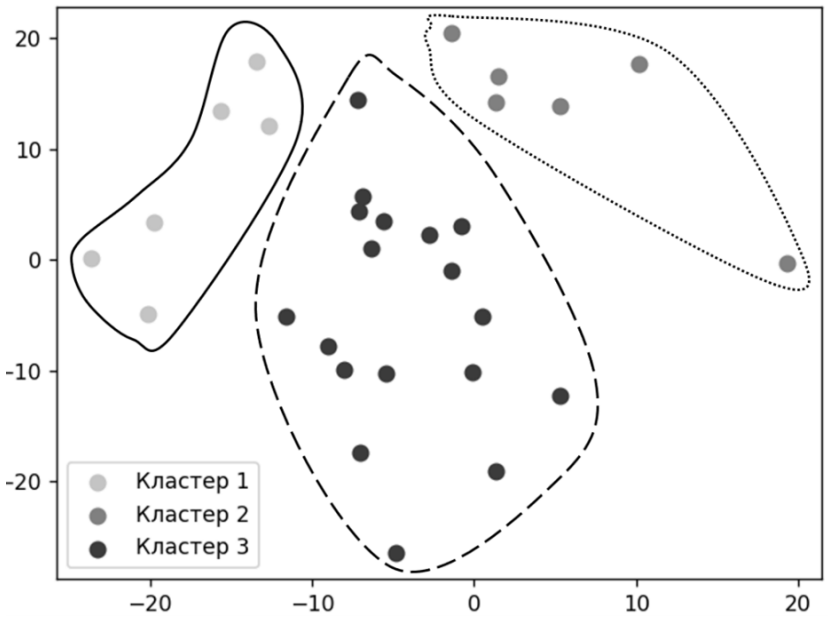
8.2.4.1. Пример кластеровања моделом Гаусове мешавине

За демонстрацију кластеровања моделом Гаусове мешавине искоришћен је исти скуп података као у претходним примерима: 30 узорака, са 5 центара Гаусове дистрибуције са стандардном девијацијом 10 (секција 8.2.3.1). За креирање модела за кластеровање коришћена је класа *GaussianMixture* из модула *sklearn.mixture* (Слика 8.45), којој се прослеђује захтевани број кластера (*n_components* параметар). *GaussianMixture* модел се обучава позивом *fit* функције и кластерује податке позивом *fit_predict* функције, којој се прослеђује скуп података за кластеровање, а то је и у овом случају листа названа *podaci_za_deljenje*. Након обучавања, модел је спреман за кластеровање методом *predict*, којом се добија низ лабела (промењива *oznake_klastera*) код кога индекс показује позицију податка у скупу, а лабела указује ком кластеру податак припада.

```
gmm = GaussianMixture(n_components=3)
gmm.fit(podaci_za_deljenje)
oznake_klastera = gmm.predict(podaci_za_deljenje)
```

Слика 8.45: : кластеровање засновано на дистрибуцији GaussianMixture моделом

Резултат кластеровања моделом Гаусове мешавине представљен је на следећем дијаграму (Слика 8.46). Разлика у односу на претходне методе кластеровања је очигледна, што је очекивано обзиром да је критеријум груписања података сличност у расподели њихових вероватноћа вероватноћу појављивања парова промењивих.



Слика 8.46: : резултат кластеровања GaussianMixture моделом

8.2.5. Кластеровње засновано на расплинутој логици

Кластеровње засновано на расплинутој логици (енгл. *Fuzzy clustering*) омогућава да податак може припадати у више кластера – у одређеном степену (секција 5), који се представља вредношћу функције припадања кластеру. Те вредности се крећу у интервалу [0,1]. Ако податак не припада кластеру вредност његове функције припадања је нула. У сваком другом случају податак припада кластеру у *одређеној мери*. Податак у потпуности припада неком кластеру ако је вредност функције припадања том кластеру један. Најчешће помињан је *Fuzzy C-means* алгоритам, који представља варијанту *K-mean* алгоритма (секција 8.2.1). Принцип рад овог алгоритма је одређивање центроида, а затим мерење функције припадања кластерима за сваки податак на бази дистанце од центроида кластера. Резултат кластеровања *Fuzzy C-means* алгоритмом је матрица функције припадања података кластерима (Слика 8.47). Ова матрица има *CxN* елемената при чему је *C* број кластера, док *N* представља број података који се кластерују. У конкретном примеру матрица функције припадања садржи 3 x 30 елемената. Подаци су означени *p* префиксом, а кластери *klaster* префиксом.

	p1	p2	p3	...	p30
klaster1	0.1	0.8	0.2	...	0.9
klaster2	0.7	0.4	0.9	...	0.5
klaster3	0.3	0.2	0.1	...	0.1

Слика 8.47: матрица функције припадања података кластерима

Вредности у матрици су у интервалу [0,1]. Што је вредност ближа јединици, податак *више* припада кластеру и обрнуто. На примеру се види да податак *p30* у *великој мери* припада кластеру *klaster1*, у средњој мери припада кластеру *klaster2* и у малој мери припада кластеру *klaster3*.

8.2.5.1. Пример кластеровања Fuzzy C-means алгоритмом

За демонстрацију кластеровања *Fuzzy C-means* алгоритмом искоришћен је исти скуп података као у претходним примерима: 30 узорака, са 5 центара Гаусове дистрибуције са стандардном девијацијом 10 (секција 8.2.3.1). За креирање модела за кластеровање коришћена је функција *cmeans* из модула *skfuzzy* (Слика 8.48), којој се прослеђују подаци за кластеровање (*podaci_za_deljenje.T*), захтевани број кластера (*n_clusters* параметар), фактор расплинутости (*m*), чије вредности што су веће од један дају већу расплинутост. Параметри *error* и *maxiter* дефинишу критеријуме заустављања обучавања алгоритма. Обучавање кроз итерације ће се зауставити када се постигне промена вредности у матрици функције припадања у две суседне итерације мања од вредности параметра *error*. Параметар *maxiter* дефинише максималан број итерација за обучавање алгоритма и зауставља алгоритам без обзира на све остале предефинисане услове. *cmeans* функција резултира са скупом исхода. Вредност *cntr* је 2Д низ који садржи вредности центроида кластера. Вредности *u* је матрица функције припадања података кластерима. Податак *u0* је иницијална матрица функције припадања (попуњена иницијалним вредностима, које могу бити све 0, насумично изабране вредности, или очекиване вредности). Податак *d* је 2Д низ који садржи еуклидске дистанце података од центроида кластера. Податак *jm* је вектор вредности коефицијента расплинутог кластеровања (енгл. *Fuzzy partitioning coefficient - FPC*) кроз итерације (у вези са параметром *error*). Податак *p* представља коначни број итерација који је извршен у фази обучавања алгоритма. Податак *fpc* је коначна вредности коефицијента расплинутог кластеровања.

```
n_clusters = 3
m = 1.5
cntr, u, u0, d, jm, p, fpc = fuzz.cmeans(
        podaci_za_deljenje.T, n_clusters, m, error=0.005, maxiter=1000
)
```

Слика 8.48: кластеровање Fuzzy C-means *алгоритмом*

На следећој илустрацији (Слика 8.49) приказана је резултујућа матрица функције припадања (промењива *u*). Након обучавања, добијена је висока вредност коефицијента расплинутог кластеровања (*FPC* = ~0.82). Резултујућа матрица има три реда - за сваки кластер по један. У сваком реду су по 30 вредности функције припадања – за сваки податак по једна. Види се да су све вредности веће од нуле и мање од један. То значи да сваки податак припада сваком кластеру у одређеној мери. У датом примеру исти податак (тачкасто уоквирене вредности) има највећу и најмању функцију припадања у матрици. То је податак (1.30219902, 14.19678053), са најмањом вредности функције припадања 3. кластеру од $1.97*10^{-5}$ и са највећом вредношћу од 0.9996 за функцију припадања 2. кластеру. Исти податак има вредност од $1.82*10^{-4}$ за функцију припадања 1. кластеру. На основу добијених резултата може се рећи да податак (1.30219902, 14.19678053) готово у потпуности припада 2. кластеру. Исти податак готово да не припада 1. и 3. кластеру.

klaster1	[[6.28797064e-01 9.94615804e-01 7.12866385e-02 4.97594683e-01 1.52813281e-02 7.68880887e-01 2.10312814e-02 1.59085003e-03 3.51878598e-04 9.61823458e-01 2.19885649e-02 9.59922951e-01 3.50258539e-04 9.71329317e-01 8.73707078e-01 8.12104281e-01 7.65432970e-03 9.90089696e-01 6.80324549e-01 9.61921192e-03 6.22661914e-01 1.51957356e-01 1.17764561e-01 9.07059920e-01 1.82411626e-04 4.69928248e-01 1.44238807e-03 8.34551961e-03 4.01093819e-02 9.37404761e-01]
klaster2	[1.52819241e-02 3.53969599e-03 2.10625659e-02 9.61373082e-02 8.15784570e-03 2.91595534e-02 1.84625973e-03 2.01944976e-04 9.99606714e-01 1.21805339e-02 9.75817524e-01 1.60250888e-02 9.99576731e-01 1.75552509e-02 3.59139173e-02 1.04196234e-01 1.46625759e-03 7.73637955e-03 2.97439702e-01 9.87284169e-01 2.68684424e-01 5.28257743e-01 6.68857220e-03 7.88565688e-02 9.99797844e-01 5.19617285e-01 3.90935465e-04 3.12707808e-03 1.28626580e-02 5.52990353e-02]
klaster3	[3.55921011e-01 1.84450006e-03 9.07650796e-01 4.06268009e-01 9.76560826e-01 2.01959560e-01 9.77122459e-01 9.98207205e-01 4.14069126e-05 2.59960085e-02 2.19391095e-03 2.40519607e-02 7.30104954e-05 1.11154324e-02 9.03790047e-02 8.36994852e-02 9.90879413e-01 2.17392409e-03 2.22357490e-02 3.09661943e-03 1.08653662e-01 3.19784901e-01 8.75546867e-01 1.40835113e-02 1.97445541e-05 1.04544664e-02 9.98166676e-01 9.88527402e-01 9.47027960e-01 7.29620382e-03]]

Слика 8.49: стварна матрица функције припадања података кластерима

Некада је потребно извршити дискретизацију резултата оваквог кластеровања. Дискретизација резултата кластеровања на бази расплинуте логике се врши најчешће одређивањем кластера за који посматрани податак има максималну вредност функције припадања. За податак (1.30219902, 14.19678053) то би био 2. кластер. За ту намену може да послужи *argmax* функција из модула *numpy*, која враћа индекс елемента низа са највећом вредношћу.

8.3. Класификација

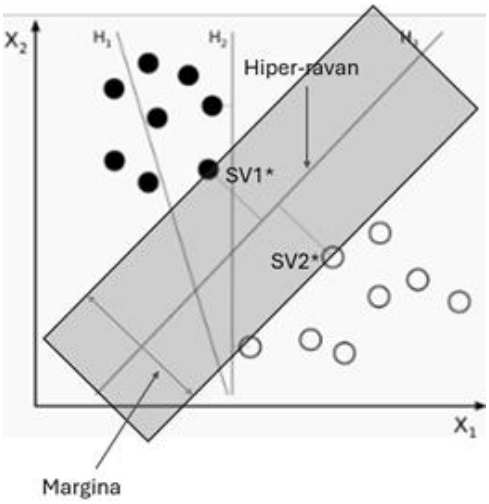
Класификација представља груписање података у именоване скупове података. Ови скупови се називају *класама*. Класе представљају типове у које се сврставају подаци процесом *класификације*. Класе се могу открити и одредити процесом кластеровања, так да се за класификацију може рећи да се заснива на кластеровању и експланаторној анализи са циљем откривања природе и дистрибуције података. Класификација као процес се још назива и *надгледано учење* (енгл. *Supervised learning*). Класификација се врши према задатим алгоритмима, на основу којих се дели у шест врста:

1. Класификација векторима подршке,
2. Класификација K најближих суседа,

3. Класификација стаблом одлучивања
4. Класификација насумичним стаблима,
5. Бајесов класификатор Naive Bayess,
6. Логиситичка регресија
7. Стабла одлучивања

8.3.1 Класификација векторима подршке

Класификација векторима подршке (SVM од енгл. *Support Vector Model*) дели податке у класе коришћењем хипер-равни и маргине која раздваја податке различитих класа. У случају линеарне класификације у две класе (Слика 8.50), хипер-раван је дуж која представља границу између простора података различитих класа. Маргина је простор око хипер-равни чија је ширина одређена дистанцом два податка из две различите класе који су најближи хипер-равни. Та два узајамно најближа податка из различитих класа називају се *вектори подршке*. У конкретном примеру, подаци *SV1* и *SV2* су вектори подршке. Они су подједнако удаљени од хипер-равни.



Слика 8.50: основни концепти класификатора векторима подршке

Поред тога, на примеру је представљен начин на који линеарни *SVM* класификатор учи. Машинско учење код *SVM* класификатора се своди на проналажење такве хипер-равни како би се добила максимална маргина да би се извршила задовољавајућа класификација. H_1 је иницијална хипер-раван, али H_1 не врши класификацију обзиром да не раздваја податке различитих класа. Други кандидат који би могао да врши класификацију је хипер-раван H_2. Проблем H_2 је што има малу маргину. Последњи кандидат

је хипер-раван H_3, која је изабрана да врши класификацију обзиром да има највећу маргину. Ако постоји више од две класе, класификатор SVM конструише више хипер-равни.

У пракси је честа немогућност да се у потпуности одвоје подаци различитих класа односно, дешава се да се подаци две класе налазе са исте стране хипер-равни која треба да их дели. Да би се наведени проблем превазишао постоје две стратегије. Прва је да се, уместо концепта стриктне (тзв. *тврде*) маргине, уводи концепт *меке* маргине. Другом речју, дозвољава се да у одређеној мери постоје изузеци у класификацији. То се постиже додавањем додатне слабе варијабле којом се омогућава *смекшавање* критеријума стриктне маргине и компензовање погрешних класификација. Друга стратегија је нелинеарна класификација.

8.3.1.1 Нелинеарна класификација код SVM класификатора

Нелинеарна класификација код *SVM* класификатора се добија *кернелом* (енгл. *kernel*) – математичком функцијом која мапира улазне податке у вишедимензиони простор својстава, који се још назива *кернел* простор. Ова функција као својство узима неку меру сличност између података на основу њиховог релативног односа. Захваљујући том својству, у Кернел простору линеарна класификација постаје могућа. На тај начин је проблем нелинеарне класификације у простору података трансформисан у решење линеарне класификације у простору својстава.

Као *кернел* функције користе се радијалне и полиномске функције. Радијална функција се најчешће користи као кернел функција код *SVM*. Она израчунава као својство еуклидску удаљеност података од неке замишљене тачке. То значи да су слични подаци који имају сличне удаљености од те замишљене тачке. Та тачка може да буде и координатни почетак. У процесу машинског учења, *SVM* модел покушава да пронађе замишљену тачку која би омогућила најпрецизнију класификацију на основу скупа података датог за учење SVM класификатора. Математички израз радијалне функције (енгл. *RBF – Radial Basis Function*) се може записати (Израз 8.15):

$$RBF(a,b) = e^{-\Upsilon\|a-b\|^2} \qquad \text{И.8.15}$$

У изразу *a* и *b* su вектори својстава, $\|a-b\|$ је еуклидска удаљеност, а Υ је контролни параметар којим се регулише својство дистанце између

података као критеријум сличности. Повећање вредности Υ значи пооштравање критеријума (на пример, само подаци који су у непосредној близини се третирају као слични – у истој класи), док смањивање значи ублажавање критеријума сличности (на пример, подаци који су удаљени се узимају као слични, да припадају истој класи).

Полиномска функција се користи као *кернел* функција услед проблема нелинеарне класификације, у случају да је потребна мања флексибилност него код радијалне функције. Нарочито је корисна када су у питању мањи скупови података за класификацију. У процесу машинског учења, *SVM* модел покушава да пронађе полиномску функцију која би омогућила најпрецизнију класификацију на основу скупа података датог за учење *SVM* класификатора (Израз 8.16).

$$RBF(a,b) = (\Upsilon\langle a-b\rangle + r)^d \qquad \text{И.8.16}$$

У изразу *a* и *b* su вектори својстава, $\langle a-b\rangle$ је разлика њихових скаларних производа, Υ је контролни параметар којим се регулише утицај својства као критеријума сличности података (исто као код радијалне фунцкије), *d* је степен полинома, док је *r* константа којом се помера вредност скаларног производа по x оси пре степеновања. При класификацији, велико *r* повећева утицај чланова нижег реда полинома, док мало p повећева утицај чланова вишег реда полинома.

8.3.1.2. Пример класификације векторима подршке

У примеру класификације векторима подршке искоришћени су историјски подаци о пацијентима са дијагнозом рака дојке ради класификације малигних и бенигних случајева. Коришћени скуп података садржи 30 различитих својстава (Слика 8.51).

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

Слика 8.51: тридесет својстава за класификацију случајева рака дојке

Скуп података садржи 569 записа, са по 30 вредности својстава за сваки запис. На следећој илустрацији су представљена прва три записа (Слика 8.52).

```
[[2.057e+01 1.777e+01 1.329e+02 1.326e+03 8.474e-02 7.864e-02 8.690e-02
  7.017e-02 1.812e-01 5.667e-02 5.435e-01 7.339e-01 3.398e+00 7.408e+01
  5.225e-03 1.308e-02 1.860e-02 1.340e-02 1.389e-02 3.532e-03 2.499e+01
  2.341e+01 1.588e+02 1.956e+03 1.238e-01 1.866e-01 2.416e-01 1.860e-01
  2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 1.203e+03 1.096e-01 1.599e-01 1.974e-01
  1.279e-01 2.069e-01 5.999e-02 7.456e-01 7.869e-01 4.585e+00 9.403e+01
  6.150e-03 4.006e-02 3.832e-02 2.058e-02 2.250e-02 4.571e-03 2.357e+01
  2.553e+01 1.525e+02 1.709e+03 1.444e-01 4.245e-01 4.504e-01 2.430e-01
  3.613e-01 8.758e-02]
 [1.142e+01 2.038e+01 7.758e+01 3.861e+02 1.425e-01 2.839e-01 2.414e-01
  1.052e-01 2.597e-01 9.744e-02 4.956e-01 1.156e+00 3.445e+00 2.723e+01
  9.110e-03 7.458e-02 5.661e-02 1.867e-02 5.963e-02 9.208e-03 1.491e+01
  2.650e+01 9.887e+01 5.677e+02 2.098e-01 8.663e-01 6.869e-01 2.575e-01
  6.638e-01 1.730e-01]
```

Слика 8.52: изглед прва три од 569 записа у скупу података

За модел класификатора искоришћена је класа *SVC* (енгл. *Support Vector Classifier*) из модула *sklearn.svm* (Слика 8.53). Модел (промењива *svm*) је конструисан прослеђивањем параметара као што су *кернел* функција *rbf* (енгл. *radial basis function*), коефицијент *кернел* функција *gamma* и регулациони параметар *C* који одређује ширину маргине у процесу обучавања модела. Након конструкције, модела се обучава позивом *fit* функције, којој се прослеђује скуп за обучавање (промењива *X*) и очекиване излазне вредности (промењива *y*) за сваки од узорака у скупу за обучавање. Скуп за обучавање добијен је дељењем изворног скупа функцијом *train_test_split* (модул *sklearn.model_selection*) чиме су се добила два

подскупа – један за обучавање (тзв. тренинг) и други за проверу (тзв. тест) модела. Подешено је да скуп за обучавање садржи 398 записа (70%), док скуп за проверу садржи преосталих 171 запис (30%).

```
svm = SVC(kernel="rbf", gamma=0.5, C=1.0)
svm.fit(X, y)
```

Слика 8.53: конструкција и обучавање модела класификатора векторима подршке

8.3.1.2.1. Евалуације модела класификатора

Евалуације модела класификатора је једноставан задатак у случају да постоје стварне вредности класификације, као што је то у примеру класификације малигних и бенигних случајева рака дојке. Функција *accuracy_score* из модула *sklearn.metrics* може да послужи за проверу тачности модела (Слика 8.54). Ова функција мери прецизност класификатора упоређујући податке предикције са стварним подацима за исте вредности улазних својстава. Подаци предикције добијени су позивом функције *predict* којој се као параметар прослеђују подаци за проверу (промењива *X_provera*). Резултат предикције (промењива *y_predikcija*) се затим прослеђује заједно са стварним резултатима (промењива *y_stvarne_vrednosti*) функцији *accuracy_score*. Резултат функције је реална вредност у интервалу [0,1] и интерпретира се тако што за вредности блиске један модел је *прецизан*, док је за вредности блиске нули модел је *нејрецизан*.

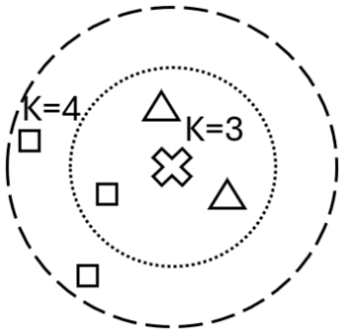
```
y_predikcija = svm.predict(X_provera)
accuracy = accuracy_score(y_stvarne_vrednosti, y_predikcija)
print(f"Preciznost: {accuracy:.4f}")
```

Слика 8.54: Евалуације модела класификатора

Израчуната прецизност модела класификатора векторима подршке у примеру је 0.9649, или ~96%, тако да је модел прецизан. Мерење је извршено на основу узорка од 171 записа података (30% од укупног скупа података коришћеног у примеру). Коришћење обученог класификатора врши се као што је описано у његовој евалуацији – позивом функције *predict* којој се као параметар прослеђују подаци, било да се ради о једном запису / пацијенту (у интерактивном раду), или више записа (*batch* обрада).

8.3.2. Класификација уз помоћ K најближих суседа

Основни принцип класификације уз помоћ K најближих суседа (*K-Nearest Neighbors - KNN*) је да се податак класификује на основу доминантне класификације суседних података. Критеријум класификације је број *K* који представља број потребних података - *сусега* који се налазе најближе податку који се класификује. За одређивање *K* најближих суседа најчешће се користи еуклидска дистанца (секција 8.2.1). Подаци који се налаза унутар те дистанце су *сусегu* и могу да припадају једној класи, или више класа. Податак који се класификује припашће доминантној класи – класи којој припада највише суседа у задатом радијусу. У следећем примеру (Слика 8.55) податак који се класификује означен је крстом. Вредност параметра *K* одређује класификацију новог податка. Ако је *K=3,* онда алгоритам тражи 3 најближа суседа. Нови податак у том случају припада класи троуглова, пошто од три суседа на израчунатој еуклидској дистанци два су троуглови.



Слика 8.55: KNN класификација зависи од вредности K

Ако је *K=4* онда алгоритам тражи 4 најближа суседа. У новој дистанци међутим постоје пет суседа од којих су три квадрати и два су троуглови. Због тога, нови податак у случају *K=4* припада класи квадрата.

8.3.2.1. Пример класификације на основу K најближих суседа у Python-у

У пример класификације на основу K најближих суседа коришћен је скуп података *iris* из јавно доступне библиотеке *sklearn.datasets*, који садржи 150 записа о три врсте ириса (биљка цветница). Сваки запис садржи податке четри својства (*sepal length* – дужина чашице цвета*, sepal width* - ширина чашице цвета*, petal length* – дужина латица цвета и *petal width* – ширина латица цвета) којима се одређује врста ириса (*setosa, versicolor, virginica*). Као у претходном примеру (секција 8.3.1.2) скуп је подељен на два подскупа

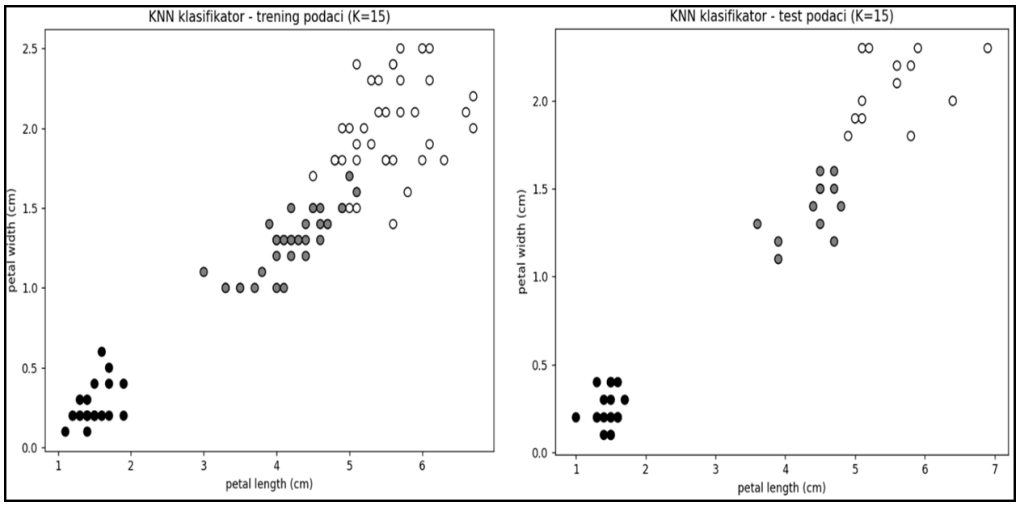
– за обучавање модела 100 записа (70%) и за проверу модела 50 записа (30%).

За модел класификатора искоришћена је класа *KNeighborsClassifier* из модула *sklearn*. *neighbors* (Слика 8.56). Модел (промењива *knn*) је конструисан прослеђивањем *K* вредности као *n_neighbors* параметра. Након конструкције, модел се обучава позивом *fit* функције, којој се прослеђује скуп за обучавање (промењива *X_train_scaled*) и очекиване излазне вредности (промењива *y_train*) за сваки од узорака у скупу за обучавање.

```
knn = KNeighborsClassifier(n_neighbors=15)
knn.fit(X_train_scaled, y_train)
```

Слика 8.56: конструкција и обучавање KNN класификатора

За евалуацију модела искоришћена је функција *accuracy_score* из модула *sklearn.metrics* (секција 8.3.1.2.1). Добијени резултати зависе од избора *K* вредности. Прецизност модела је потпуна (100%) за *K* вредности у интервалу [5,25], преко 90% је у интервалима [1,4] и [26,31]. Из евалуације се може закључити да је дистрибуција података у класама таква да постоји велико груписање података око центроида , а тиме је и висок степен диференцијације података између класа. На следећој илустрацији (Слика 8.57) су приказани резултати класификације података - два својства (дужина и ширина чашице цвета) из скупова података за обуку и евалуацију *KNN* класификатора.

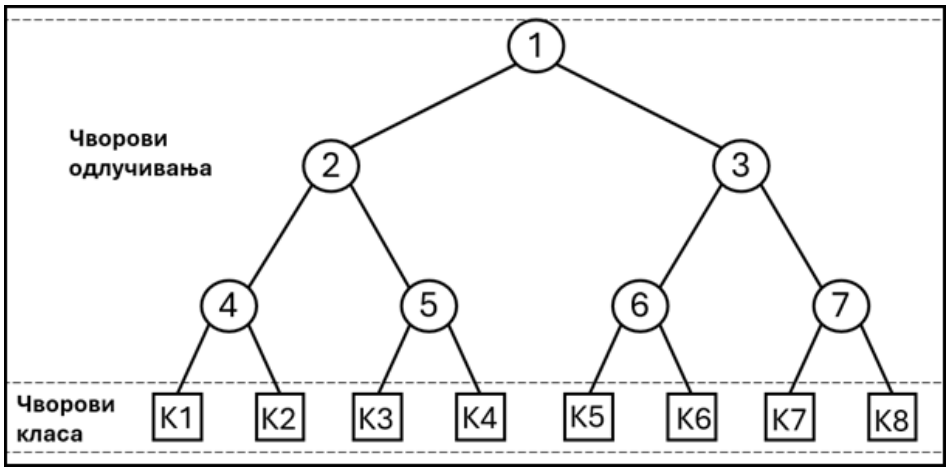


Слика 8.57: приказ класификације тренинг и тест података (дужина и ширина чашице цвета)

Коришћење обученог класификатора врши се као што је описано у његовој евалуацији – позивом функције *predict* којој се као параметар прослеђују подаци, било да се ради о једном узорку биљке, односно запису четири наведена својства цветнице (у интерактивном раду), или више записа (*batch* обрада).

8.3.3. Класификација стаблом одлучивања

Класификација стаблом одлучивања је у великој мери слична са хијерархијским кластеровањем дељењем (секција 8.2.3). Структура стабла одлучивања (Слика 8.58) је хијерархијска и састоји се од интерних (унутрашњих) чворова, који се називају *чворовима одлучивања* (чворови 1 - 7) и екстерних (спољњих) чворова који се називају *чворовима класа* (чворови K1 – K8). Приказано стабло одлучивања има фактор гранања 2, што значи да су одлуке логичког типа (тачно - нетачно). Стабла одлучивања могу да имају већи фактор гранања, што зависи како од природе проблема који се решава, знања уграђеног у систем, скупа података, као и од самог дизајна (решења) модела класификатора.



Слика 8.58: пример стабла одлучивања

Алгоритам за класификацију стаблима одлучивања је рекурзивни, извршава се одозго на доле (*top-down*), од кореног чвора према чвору класе у који ће се податак класификовати. У сваком чвору одлуке се испитује једно својство које садржи скуп података који се класификује, одлуком се испитивање наставља по дубини, све до екстерног чвора одређене класе. Посматрајући један податак из скупа који се класификује, он се креће једном путањом од кореног чвора до чвора класе у коју је класификован. Та путања је одређена вредностима својстава садржаних у податку.

Приликом класификације, кроз стабло се пропушта читав скуп података, који се практично дели у чворовима одлуке на подскупове, који су тако све мањи како класификација напредује у дубину стабла. Процес се зауставља када сви подаци достигну чворове класа. То значи да је скуп података класификован.

8.3.3.1. Пример класификације стаблом одлучивања у Python-у

У примеру класификације стаблом одлучивања коришћен је скуп података *iris* из јавно доступне библиотеке *sklearn.datasets* описан у претхоном поглављу (секција 8.3.2.1), који садржи 150 записа, сваки са по 4 својства која одређују три класе - врсте ириса. Као и у претходним случајевима (секција 8.3.1.2) скуп је подељен на два подскупа – за обучавање модела 100 записа, (70%) и проверу (евалуацију) модела – 50 записа (30%).

За модел класификатора искоришћена је класа *DecisionTreeClassifier* из модула *sklearn.tree* (Слика 8.59). Након конструкције модела

класификатора (промењива *dt_classifier*), модел се обучава позивом *fit* функције, којој се прослеђује скуп за обучавање (промењива *X_train*) и очекиване излазне вредности (промењива *y_train*) за сваки од узорака у скупу за обучавање.

```
dt_classifier = DecisionTreeClassifier()
dt_classifier.fit(X_train, y_train)
```

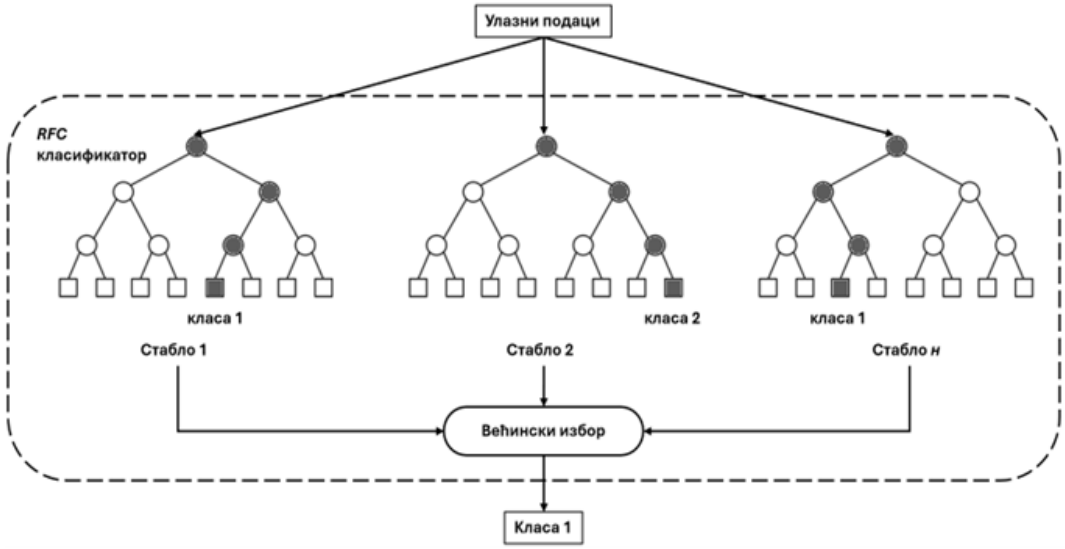
Слика 8.59: конструкција и обучавање класификатора стаблом одлучивања

За евалуацију модела искоришћена је функција *accuracy_score* из модула *sklearn.metrics* (секција 8.3.1.2.1). Постигнута прецизност модела је 100% за 50 насумичних записа из скупа података о класификацији ириса. У случајевима да се обучавањем не постиже задовољавајућа прецизност модела за класификацију, омогућено је да се при конструкцији модела коришћењем класе *DecisionTreeClassifier* експлицитно мењају многобројни параметри обучавања од којих издвајамо функцију - критеријум дељења података, максималну дубину стабла, фактор гранања стабла, максималан број својстава која се укључују у обучавање и класификацију. Детаљне информације о параметрима могу се пронаћи у документацији класе.

8.3.4. Класификација насумичним стаблима

Класификација насумичним стаблима, названа још *RFC* класификација (енгл. *Random Forest Classification - RFC*), представља надградњу класификације стаблом одлучивања. Заправо, *RFC* класификатор садржи више стабала одлучивања. Свако стабло се обучава на посебном подскупу података (делови скупа података за обучавање). Сваки подскуп података садржи део својстава над којима класификацију врше појединачна стабла *RFC* класификатора. Избор података и својстава за обучавање појединачних стабла је насумичан, тако да је структура стабала (чворови одлучивања) формирана у процесу обучавања модела класификатора такође на неки начин насумична. То је разлог зашто *RFC* класификатори имају овај придев у називу. Захваљујући наведеним особинама, могуће је на бази стабла одлучивања ефикасно класификовати комплексне и обимне скупове података са великим бројем својстава. Другим речима, *RFC* класификатор представља решење на бази декомпозиције за сложене проблеме класификације.

Структура *RFC* класификатора представљена је на следећој илустрацији (Слика 8.60). Класификатор се састоји од предефинисаног броја стабла, којима се у процесу обучавања прослеђују делови скупа за обуку и различита својства. Након обучавања, *RFC* класификатору се прослеђују подаци за класификацију (на слици означени као *улазни подаци*), који се доводе на улаз свих стабла. Обзиром да су стабла била обучавана на различитим својствима, као и на различитим подскуповима података, резултати класификација појединачних стабала одлучивања су такође различита. *RFC* класификатор садржи додатну компоненту која као улаз има исходе класификације податка сваког стабла одлучивања и доноси одлуку по принципу већинског избора. У примеру је класификатор *одлучио* да податак припада *класи 1*.



Слика 8.60: Класификација насумичним стаблима (RFC)

8.3.4.1. Пример класификације насумичним стаблима у Python-у

У примеру класификације насумичним стаблима такође је коришћен скуп података *iris* (секција 8.3.2.1), који садржи 150 записа, сваки са по 4 својства која одређују три класе - врсте ириса. Као и у претходним случајевима (секција 8.3.1.2) скуп је подељен на два подскупа – за обучавање модела 100 записа (70%) и за проверу (евалуацију) модела 50 записа (30%).

За модел класификатора искоришћена је класа *RandomForestClassifier* из модула *sklearn.ensemble* (Слика 8.61). При конструкцији модела

класификатора (промењива *rfc*) дефинише се број стабла одлучивања (промењива *n_estimators*). Вредност промењиве *n_estimators* показује да се у *RFC* класификатору конструише сто појединачних стабла одлучивања. Као и код *DecisionTreeClassifier* модела, постоји низ параметара за фино подешавање процеса обучавања класификатора.

Након конструкције модел се затим обучава позивом *fit* функције, којој се прослеђује скуп за обучавање (промењива *X_train*) и очекиване излазне вредности (промењива *y_train*) за сваки од узорака у скупу за обучавање. У циљу евалуације модела, извршена је класификација на подскупу за проверу (*X_test*).

```
rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(X_train, y_train)
y_pred = rfc.predict(X_test)
```

Слика 8.61: конструкција и обучавање класификатора насумичним стаблима

Упоређивањем резултата класификације података за 50 узорака цвета ириса (промењива *y_pred*) са њиховим стварним класама врши се евалуација модела. За ту сврху, искоришћена је функција *accuracy_score* из модула *sklearn.metrics* (секција 8.3.1.2.1). Постигнута прецизност модела је 100% за 50 насумичних записа из скупа података о класификацији ириса. У случајевима да се обучавањем не постиже задовољавајућа прецизност модела за класификацију, омогућено је да се при конструкцији модела експлицитно мењају параметри обучавања.

8.3.5. Бајесов класификатор

Бајесов класификатор (енгл. *Naïve Bayes Classifier - NBC*) врши класификацију података на основу Бајесове теореме (секција 7) и (*наивне*) претпоставке да су својства, која су укључена у податке за класификацију, узајамно независна и подједнако важна. Поред тога подразумева се да својства која имају континуалне вредности имају Гаусову (нормалну) дистрибуцију, а својства која имају дискретне вредности имају полиномску расподелу.

8.3.4.1. Пример класификације Бајесовим класификатором у Python-у

У примеру класификације Бајесовим класификатором коришћен је скуп података *iris* из јавно доступне библиотеке *sklearn.datasets* описан у

претхоном поглављу (секција 8.3.2.1), који садржи 150 записа, сваки са по 4 својства која одређују три класе - врсте ириса. Као и у претходним случајевима (секција 8.3.1.2) скуп је подељен на два подскупа – за обучавање модела 100 записа, (70%) и проверу (евалуацију) модела – 50 записа (30%).

За модел класификатора искоришћена је класа *GaussianNB* из модула *sklearn.naive_bayes* (Слика 8.62). При конструкцији модела класификатора (промењива *model*), параметри нису неопходни. Ако постоје подаци о априорним вероватноћама класа у које се подаци класификују, онда се укључују у конструкцију као параметар *priors* чија је вредности низ вредности априорних вероватноћа класа. Након конструкције модел се затим обучава позивом *fit* функције, којој се прослеђује скуп за обучавање (промењива *X_train*) и очекиване излазне вредности (промењива *y_train*) за сваки од узорака у скупу за обучавање. Да би се извршила евалуација модела, функцији *predict* прослеђује се подскуп података за проверу (*X_test*). Иста функција се позива и за коришћење модела у класификацијама.

```
model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Слика 8.62: конструкција и обучавање Бајесовог класификатора

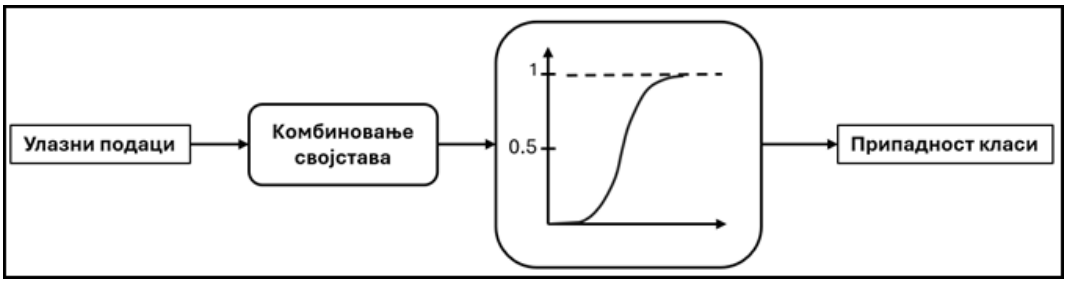
Добијен разултат предикције се заједно са правим вредностима класа узорака прослеђује функцији *accuracy_score* из модула *sklearn.metrics* (секција 8.3.1.2.1). У примеру је добијена тачност класификације цветова ириса од 97.78%, док је над истим скупом података коришћењем полиномског Бајесовог класификатора (класа *MultinomialNB*) добијена тачност од 95.56%, што значи да је избор модела класификатора у односу на дистрибуцију вероватноћа података у скупу био исправан. Поред ова два модела класификатора, постоји и *BernoulliNB* – Бајесов класификатор специјализован за бернулијеву расподелу вероватноћа узорака у популацији. Са *BernoulliNB* моделом постигнута је најмања прецизност класификације од свега 28.89%.

8.3.6. Класификатор логистичком регресијом

Класификатор логистичком регресијом врши мапирање улазног података, који садржи вредности једног, или више својстава, у излазну вредност вероватноће [0,1] којом улазни податак припада одређеној класи, или не. За мапирање се у случају класификатора логистичком регресијом користи логистичка (*сигмоидна*) функција (Израз 8.17), која представља вредност вероватноће улазног податка x, код које је μ – параметар који одређује положај сигмоиде (криве логистичке функције) у односу на x осу и s је параметар за скалирање како би се добиле излазне вредности у [0,1] интервалу.

$$p(x) = \frac{1}{1 + e^{-(x-\mu)/s}} \qquad \text{И.8.17}$$

Једноставна структура класификатора (Слика 8.63) садржи два процесора података – један за комбиновање својстава из улазних података, а други који одређује припадност класи тих улазних података на основу логистичке функције.



Слика 8.63: структура класификатора логистичком регресијом

Класификатор логистичком регресијом може да се користи и у случајевима када постоји више од две класе у које треба сврстати улазне податке. У том случају се користи полиномски регресивни модел (секција 8.1.2.5), док се у моделу уместо логистичке - користи функција заснована на експонентима улазних података и тежинских фактора које они имају у односу на задате класе. Ова функција конвертује вектор резултата у расподелу вероватноће припадања задатим класама (Израз 8.18).

$$P(Y = c|X = x) = \frac{e^{w_c x + b_c}}{\sum_{i=1}^{K} e^{w_i x + b_i}} \qquad \text{И.8.18}$$

Вероватноћа припадања податка x класи c се израчунава као количник који у бројиоцу садржи експонент суме коефицијента помераја b_c и производа податка x и тежинског фактора w_c који онима за класу c. За нормализацију израза у имениоцу је сума коефицијента помераја (b_i) и производа податка и тежинских фактора ($w_i x$) за сваку класу у скупу од укупно *K* класа.

8.3.6.1. Пример класификације логистичком регресијом у две класе

У примеру класификације логистичком регресијом у две класе, коришћен је скуп *breast cancer Wisconsin dataset* о пацијентима са дијагнозом рака дојке ради класификације малигних и бенигних случајева. Коришћени скуп података садржи 569 записа, са по 30 различитих својстава из јавно доступне библиотеке *sklearn.datasets*. Као и у претходним примерима, скуп података је подељен у два подскупа, за обуку 80% (*X_train*) и за евалуацију 20% узорака (*X_test*). За модел класификатора искоришћена је класа *LogisticRegression* из модула *sklearn.linear_model* (Слика 8.64). При конструкцији класификатора (промењива *lr_klasifikator*) задат је број итерација за обуку модела (параметар *max_iter*). Други параметар (*random_state*) омогућава избор интерне функције за узорковање података у току обучавања. Након конструкције, модел се обучава позивом функције *fit* и прослеђивањем података за обуку и њихових ознака (припадања класама). Након обучавања, модел се евалуира и користи позивом функције *predict* уз прослеђивање података који се класификују. Резултат функције је низ ознака за припадање податка класама.

```
lr_klasifikator = LogisticRegression(max_iter=10000, random_state=0)
lr_klasifikator.fit(X_train, y_train)
predikcije = lr_klasifikator.predict(X_test)
```

Слика 8.64: конструкција и обучавање модела класификатора логистичком регресијом

За евалуацију је коришен добијен разултат предикције (параметар *predikcije*) који је прослеђен заједно са правим вредностима класа узорака функцији *accuracy_score* из модула *sklearn.metrics* (секција 8.3.1.2.1). У примеру је добијена тачност класификације бенигних и малигних тумора од 96.49%.

8.3.6.2. Пример класификације логистичком регресијом у 3 или више класа

Други пример класификације логистичком регресијом показује како се врши класификација у три, или више класа. Обзиром да скуп података *iris* (секција 8.3.2.1) садржи три класе – три врсте ириса, искоришћен је у овом примеру. Скуп је подељен на два подскупа – за обучавање модела 100 записа, (70%) и проверу (евалуацију) модела – 50 записа (30%). Као и за само две класе, као класификатор искоришћен је *LogisticRegression* модел из модула *sklearn.linear_model* (Слика 8.65).

```
model = LogisticRegression(multi_class='multinomial')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Слика 8.65: конструкција и обучавање модела класификатора логистичком регресијом за +3 класа

Обучавање и предикција коришћењем обученог модела је идентична као што је описано у претходној секцији. За евалуацију модела искоришћена је функција *accuracy_score*, при чему је добијена прецизност модела од 95%:

8.4. Закључак

У системима машинског учења и вештачке интелигенције коришћење података се највећим делом своди на решавање проблема њихове класификације и предикције. Објашњене су најзначајније методе, технике и модели за предикцију, кластеровање и класификацију. Као најстарији, објашњени су модели који се заснивају на регресији. Регресија омогућава моделовање линеарних и нелинеарних зависности излазне (зависне) промењиве и једне или више независних промењивих (својстава) у циљу бинарне класификације и предикције. У другом делу описано је кластеровање као техника која решава проблем груписања података када не постоји претходно знање о природи и дистрибуцији података. Применом одговарајућег алгоритма за кластеровање потпуно непознате податке је могуће груписати у неименоване, али смислене скупове зване кластерима. Након откривања природе и дистрибуције података, класификација омогућава, као резултат обраде података њихово груписање у класе (категорије) – предефинисане (именоване) скупове који одражавају заједничка својства података који тим скуповима припадају.

8.5. Питања

1. Која је намена Пирсоновог коефицијента корелације?
2. У ком случају се користи вишеструка линеарна регресија?
3. Која је разлика између експоненцијалног и степеновaног модела код нелинеарне регресије?
4. Која је практична примена хармоничног опадајућег модела?
5. Која је намена коефицијента детерминације и код које врсте нелинеарне регресије се користи?
6. Која је разлика између K-means и Mean-shift алгоритама за кластеровање?
7. Које су врсте хијерархијског кластеровања?
8. У ком случају се користи кластеровање засновано на расплинутој логици?
9. Који су основни концепти класификације векторима подршке?
10. Како функционише алгоритам K-најбλижих суседа?
11. Која је предност класификације насумичним стаблима у односу на класификацију стаблом одлучивања?
12. Која се функција користи код класификације логистичком регресијом?

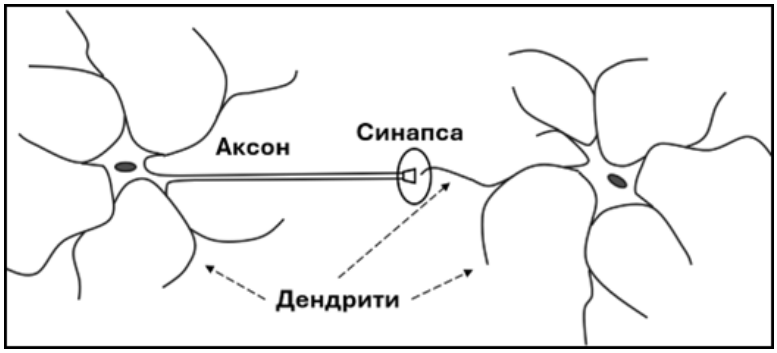
9. Вештачке неуронске мреже

Циљ поглавља је упознавање са моделима вештачких неуронских мрежа – како се обучавају и користе, како се конструишу модели различитих типова неуронских мрежа, примењивост модела на решавању различитих типова проблема класификације и предикције, као и прилагођавање модела специфичним форматима улазних података и очекиваним резултатима.

Вештачке неуронске мреже (енгл. *Artificial Neural Networks - ANN*) представљају технологију вештачке интелигенције; конкретније то су модели машинског учења који се састоје од мреже узајамно повезаних вештачких неурона са циљем учења из података и прављења предикција.

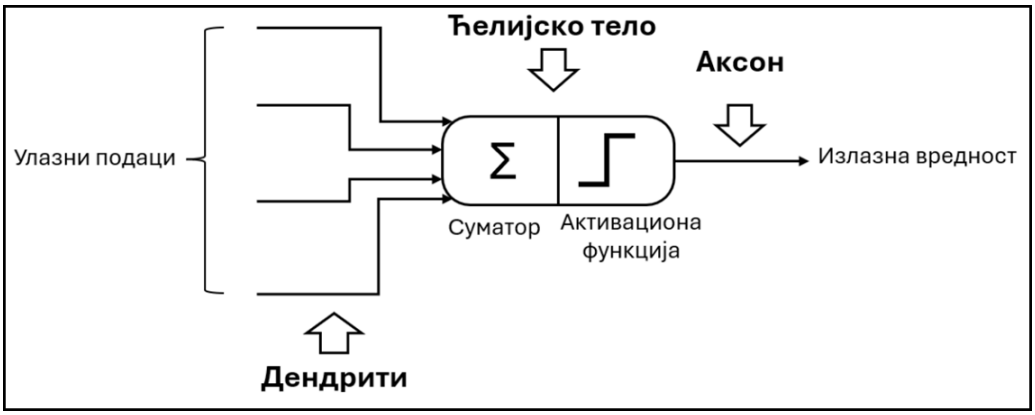
9.1. Модел неурона

Структура вештачких неуронских мрежа је инспирисана мозгом живих бића. Мозак живих бића састоји се од нервних ћелија које се називају неурони (Слика 9.1). Поред ћелијског једра и тела, неурони садрже и посебне *израштаје*, којима остварују узајамну електрохемијску везу. Ти израштаји се називају *дендрити* и *аксони*. Аксони су ћелијски израштаји који емитују електричне импулсе из неурона. Дендрити су ћелијски израштаји који примају електричне импулсе из других неурона. Електрохемијска веза између аксона једног неурона и дендрита другог неурона се назива *синапса*. Већина неурона садрже само један аксон и већи број дендрита. Људски мозак садржи мрежу од ~ 86 милијарди неурона који су узајамно повезани са ~100 трилиона синапси. То практично значи да је сваки неурон повезан са ~1000 суседних неурона.



Слика 9.1: синаптичка веза два природна неурона

При моделовању неурона (Слика 9.2), дендрити су разматрани као улази у неурон, пошто омогућавају пријем електричних импулса из других неурона. На сличан начин аксон је разматран као излаз из неурона, пошто емитује сигнал ка другом неурону. Само тело неурона је разматрано као процесор који сумира улазне сигнале (податке) из дендрита и на основу њих, генерише неки излазни сигнал – излазну вредност, који се шаље путем аксона даље кроз неуронску мрежу. У приказу, вештачки неурон генерише дискретне излазне вредности. То могу да буду и друкчије вредности што се дефинише избором активационе функције неурона. На пример, ако се користе сигмоидна функција, или функција гаусове расподеле добиће се континуалне вредности на излазу неурона.



Слика 9.2: модел вештачког неурона

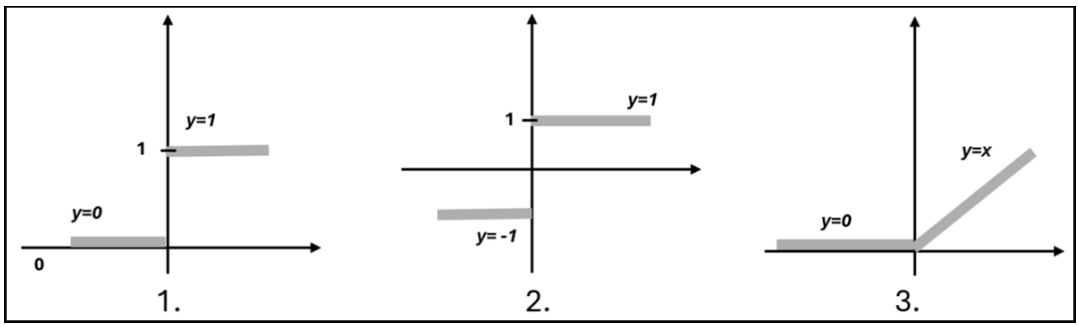
Жива бића су у стању да уче на основу искустава и то траје од рођења до краја живота. На нивоу природног неурона и неуронске мреже, учење се одвија процесом понављања. Код животиња, то су понављање радњи, промена понашања под утицајем искустава (страх од ватре или воде), а код људи то су много сложенији мисаони процеси на вишим нивоима апстракције, као што су решавање проблема, рачунање, или памћење великог броја различитих ствари које нису искуствене (текст песме, улоге, називи и садржаји уметничких дела, алфа-нумеричке вредности важне за посао и живот итд.).

9.2. Активационе функције неурона

Активационе функције вештачких неурона омогућавају подешавање како неурона, тако и читаве неуронске мреже према конкретном задатку предикције или класификације у циљу добијања што веће прецизности

модела. Као што је већ поменуто, постоје различите активационе функције вештачких неурона. У овој секцији ће бити описане најважније од њих. Подела активационих функција неурона је на линеарне и нелинеарне. Најпознатије линеарне активационе функције неурона су (Слика 9.3):

1. Дискретна функција *са прагом* – постоје две дискретне вредности {0,1}
2. Дискретна функција – постоје две дискретне вредности {-1,1}
3. Комбинована функција, са прираштајем – позната као ReLU функција (од енгл. *Rectified Linear Unit*)



Слика 9.3: линеарне активационе функције вештачких неурона

Најпознатије нелинеарне активационе функције неурона су (Слика 9.4):

1. Сигмоидна функција – образује *S* криву у интервалу [0,1] и израчунава се према изразу 9.1:

$$f(x) = \frac{1}{1 + e^{-x}} \qquad \text{И.9.1}$$

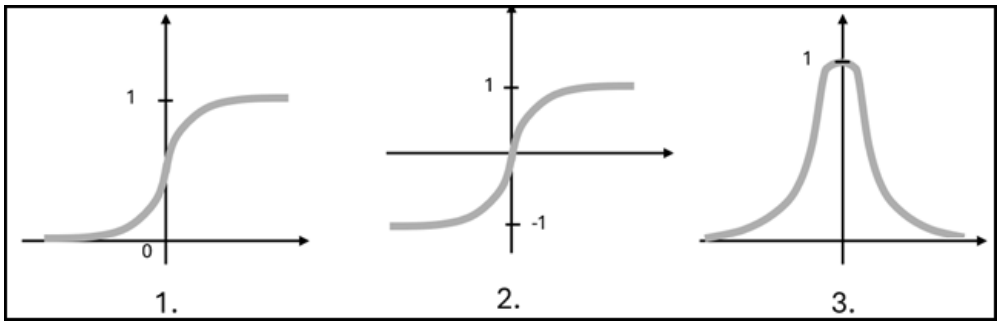
2. Хиперболична тангенс функција (*tan-h*) – образује *S* криву у интервалу [-1,1] и израчунава се према изразу 9.2:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \qquad \text{И.9.2}$$

3. Гаусова функција – образује звонасту криву (Гаусово звоно), тако да је максимум вредности функције за средњу вредност улазног параметра и израчунава се према изразу 9.3:

$$f(x) = e^{-x^2} \qquad \text{И.9.3}$$

Код свих активационих функција средина вредности улазне промењиве је у координатно почетку (вредност 0).

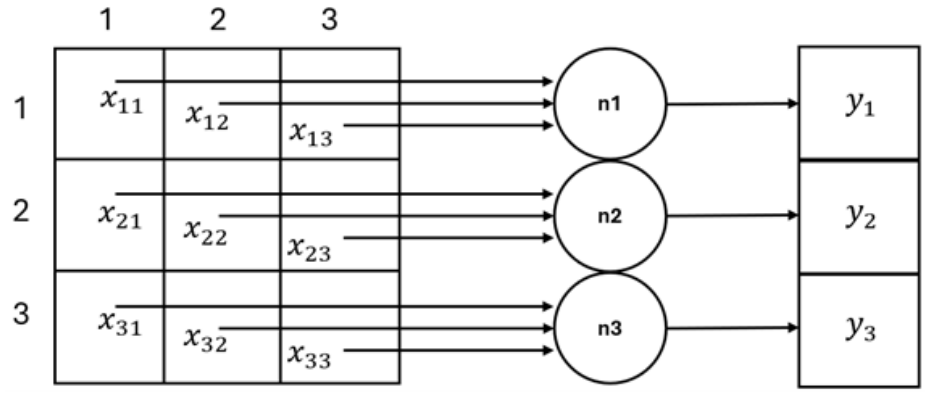


Слика 9.4: нелинеарне активационе функције неурона

Вредност излазне промењиве за централну вредност улазне промењиве код сигмоидне функције је 0.5, код *tan-h* функције је 0, док је код Гаусове функције максимум (вредност 1).

9.3. Неурони за интерпретацију правила - препознавање обрасца

Упркос замисли да вештачки неурон искључиво представља градивну јединицу неуронске мреже, постоје корисни начини удруживања неурона при чему се не гради мрежни модел. На следећем примеру представљена су 3 неурона (n_1, n_2 и n_3) који узајамно нису повезани али делују као целина (Слика 9.5). Сваки неурон има по три улаза, на које се у примеру доводе дискретни сигнали са фото осетљивих елемената представљени матрицом 3x3 елемента. Тако први неурон прима сигнале првог реда елемената, други другог и трећи неурон трећег реда елемената ($x_{11}, \ldots, x_{33}$). Излазне функције неурона су дискретне и означене као y_1, y_2, y_3.



Слика 9.5: систем за препознавање образаца од 3 неурона

Ако су задате таблице истинитости (Слика 9.6) за одређивање понашања неурона онда осам колона могу да се разматрају као дефиниције правила (секција 3.1) која у премиси (условном делу) имају три вредности повезане конјункцијом (x_{i1},x_{i2} и x_{i3}) и дају (у акционом делу правила) излазне y_i вредности. У том случају три неурона представљају систем за извршење правила.

n1

X11	0	0	0	0	1	1	1	1
X12	0	0	1	1	0	0	1	1
X13	0	1	0	1	0	1	0	1
Y1	0	0	1	1	0	0	1	1

n2

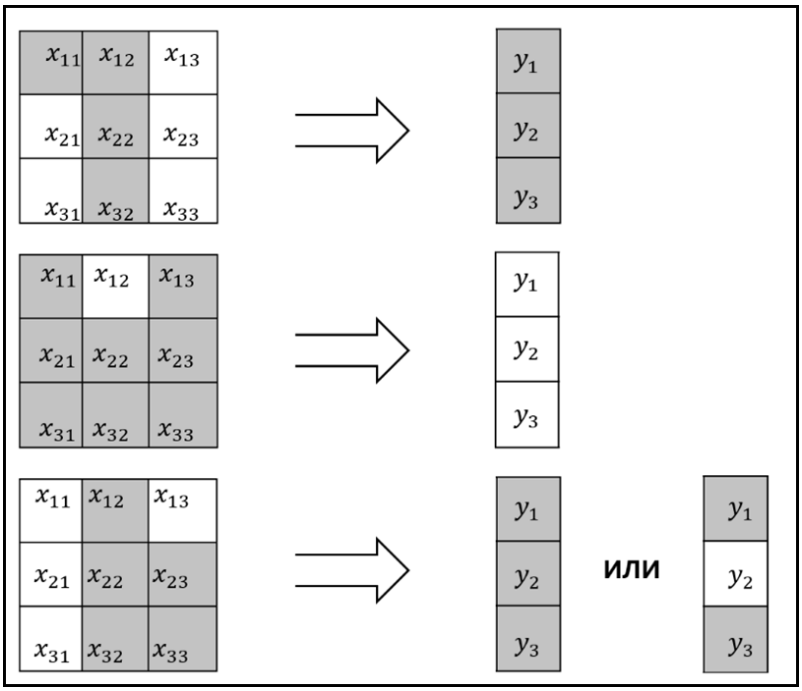
X21	0	0	0	0	1	1	1	1
X22	0	0	1	1	0	0	1	1
X23	0	1	0	1	0	1	0	1
Y2	1	1/0	1	1/0	1/0	0	1/0	0

n3

X31	0	0	0	0	1	1	1	1
X32	0	0	1	1	0	0	1	1
X33	0	1	0	1	0	1	0	1
Y3	1	0	1	1	0	0	1	0

Слика 9.6: таблице истинитости за три неурона

На следећој илустрацији (Слика 9.7) приказан је рад система за препознавање образаца. Без обзира на то да ли се тамна поља интерпретирају као тачне или нетачне вредности (1 или 0), систем ће генерисати исправне резултате. То значи да се директном и инверзном логиком добијају исти резултати.



Слика 9.7: рад система од 3 неурона за препознавање образаца

Интересантно је да неурон n_2 генерише вредности повезане дисјункцијом (0/1). То значи да неурон n_2 може да произведе два алтернативна излазна резултата. Ова особина је интересантна у случајевима да се препознају обрасци у сегментима (деловима) веће целине, тако да су потребне различите интерпретације у зависности од ширег контекста. Систем на тај начин постаје прилагодљив на захтеве које намећу нове ситуације (контексти) у којима се врше препознавања.

9.4. Класификација вештачких неуронских мрежа

Постоје многобројне класификације вештачких неуронских мрежа, које се могу поделити у две велике групе: мреже са једносмерним простирањем сигнала и мреже са повратном спрегом.

У мреже са једносмерним простирањем сигнала спадају:

1. Једнослојни перцептрон (секција 9.4.1),
2. Мреже са базом радијалних функција (секција 9.4.2),
3. Вишеслојни перцептрон (секција 9.4.3) и
4. Конволуционе мреже (секција 9.5).

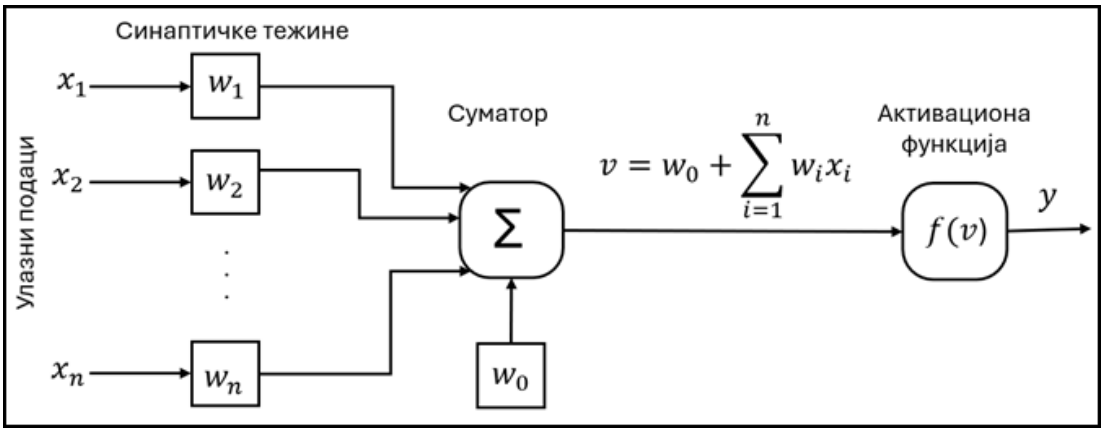
Најзначајније мреже са повратном спрегом су:

1. Рекурентне мреже (секција 9.4.4),
2. Мреже дугорочне и краткорочне меморије (секција 9.4.5) и
3. Мреже рекурентних јединица са капијама (секција 9.4.6).

9.4.1. Једнослојни перцептрон

Неурон који има могућност бинарне класификације (у две класе) на основу вишеструких улазних сигнала (података) назива се једнослојни перцептрон (енгл. *Single Layer Perceptron - SLP*). Модел једнослојног перцептрона за дефинисање улазних вредности, поред података садржи и синаптичке тежине. Синапсе су откривене још крајем 19. века. Средином прошлог века је утврђено да се понављањем светлосних и звучних подражаја мењају синаптичке тежине у појединим делова мозга сисара, на основу чега су синаптичке тежине доведене у директну везу за процесом учења. Кроз понављање, електрична проводности синапси неурона у циљној области мозга у се побољшавала. Да би се то постигло у моделу вештачког неурона дендритима су, поред улазних вредности (x_i) додате и синаптичке тежине w_i које могу да се мењају (Слика 9.8). Модел тако сумира производе улазних вредности и вредности синаптичких тежина ($w_i x_i$). Тој суми је додата још вредност фактора w_0 (енгл. *bias*) којом се коначна сума v може додатно подешавати ради постизања што бољих резултата процеса обучавања. Коначна излазна (y) вредности неурона добија се као резултат активационе функције $f(v)$, којој се као параметар прослеђује вредности v.

Једнослојни перцептрон тако има два режима рада: режим обучавања (тренинга), у коме се могу мењати синаптичке тежине w_i и режим коришћења (експлоатације), када то више није могуће. Вредност w_0 фактора је најчешће предефинисана још при изградњи модела вештачке неуронске мреже, дакле пре почетка процеса обучавања.



Слика 9.8: комплетан модел једнослојног перцептрона

Једнослојни перцептрон има употребну вредност у решавању задатака бинарне класификације. За вишеструке (>2) класификације, које се врше на основу вишеструких критеријума и са комплексним структурама података потребно је удруживање неурона у сложеније структуре које се називају вештачким неуронским мрежама.

9.4.1.1. Обучавање модела једнослојног перцептрона

Обучавање модела једнослојног перцептрона је итеративни процес у коме се на основу скупа већ класификованих података (означених 1, или 0) врше промене синаптичких тежина w_i, како би се минимизирала грешка класификације (Израз 9.4). Нова вредност синаптичке тежине w_i' добија се корекцијом постојеће вредности w_i на основу разлике очекиване (тачне) вредности y_T и добијене вредности класификацијом y_P. Величина промене нове синаптичке тежине w_i' је одређена фактором η који се још назива и *брзина учења*. У случају да разлика $y_T - y_P$ не постоји други члан у изразу има вредност 0, тако да је $w_i' = w_i$. Иначе, разлика $y_T - y_P$ има вредност -1, или 1, што значи да дефинише тренд промене (предзнак производа), док величину промене одређује брзина учења η. Број итерација може да буде предефинисан, или условљен стагнацијом напредовања у процесу обучавања ($w_i' \cong w_i$).

$$w_i' = w_i + \eta(y_T - y_P) \qquad \text{И.9.4}$$

Поред синаптичких тежина, у случају потребе (немогућност задовољења циља учења), кроз процес обучавања може се мењати и вредност фактора w_0, на исти начин као што је то већ описано.

9.4.1.2.Пример имплементације једнослојног перцептрона у *Python*-у

За конструкцију једнослојног перцептрона искоришћена је класа *Sequential* из модула *keras* (Слика 9.9). Модел је намењен бинарној класификацији (подаци се сврставају у две класе). У моделу се дефинише један слој класе *Dense* модула *keras.layers*, такав да садржи само један неурон (први параметар), који има сигмоидну функцију као активациону функцију (параметар *activation*) и који има два улаза (параметар *input_shape*).

```
model = Sequential([
    Dense(1, activation='sigmoid', input_shape=(2,))
])
```

Слика 9.9: конструкција једнослојног перцептрона

Након конструкције, потребно је да се модел пре обучавања конфигурише (Слика 9.10). За ту намену користи се метода *compile*, којој се прослеђује параметар *optimizer* који одређује начин ажурирања синаптичких тежина у току процеса обучавања, затим параметар *loss*, који одређује функцију губитка која упоређује стварне и предвиђане вредности у току обучавања и трећи, опциони параметар *metrics*, који одређује начин праћења процеса обучавања кроз итерације. Може да буде више критеријума праћења. У конкретном случају то је *accuracy* (тачност). Обзиром да се модел у примеру користи за бинарну класификацију, као функција губитка изабрана је *binary_crossentropy*.

```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Слика 9.10: припрема модела једнослојног перцептрона за обучавање

Након претходна два корака, модел је спреман за обучавање. У примеру је коришћен скуп података синтетички генерисан на исти начин као и за претходно описане класификаторе (секција 8.3.1.2). Обучавање започиње позивом методе *fit* (Слика 9.11), којој се прослеђују подаци за обуку (параметри *X_train* и *y_train*), дефинише се максималан број итерација у којима ће модел обучавати на задатим подацима (параметар *epochs*), дефинише се колико узорака ће се узети пре ажурирања параметара модела (параметар *batch_size*), дефинише се број узорака који ће се

користити за валидацију модела у току процеса обучавања (параметар *validation_split*). Последњи приказан параметар (*verbose*) одређује шта ће се приказивати на излазном панелу у току процеса обучавања (0 – без приказа, 1 – комплетан приказ и 2 – концизан приказ). Функција *fit* враћа резултат процеса обучавања кроз итерације, који може да се сачува ради касније анализе (промењива *history*).

```
history = model.fit(X_train, y_train,
                    epochs=200,
                    batch_size=16,
                    validation_split=0.1,
                    verbose=0)
```

Слика 9.11: обучавање модела једнослојног перцептрона

Евалуација модела (Слика 9.12) врши се методом *evaluate*, којој се прослеђује подскуп података за проверу (*X_test*) са очекиваним вредностима (*y_test*).

```
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy: {accuracy:.2f}")
print(f"Test loss: {loss:.2f}")
```

Слика 9.12: евалуација модела једнослојног перцептрона

Научен и евалуиран модел се користи позивом функције *predict* (Слика 9.13), којој се прослеђују подаци за предикцију. Резултат предикције је објекат која садржи матрицу, коју чине вектор излаза из перцептрона (вектор неурона у излазном слоју) и вектор предикција за улазне вредности података. Обзиром да у конкретном случају постоји само један неурон и 20 предикција добиће се матрица са 1x20 предикција. Коришћењем *numpy.array* методе, из објекта се издваја матрица (промењива *predikcije_matrica*), из које се елиминише једна димензија методом *reshape* којој се прослеђује као параметар -1 и добија се вектор предикција. Предикције (промењива *predikcije_vektor*) имају вредности у интервалу [0,1] и представљају вероватноће у којој мери узорци припадају првој класи. Како би добили вектор бинарних резултата, код којих ће 0 означавати припадност првој, а јединица другој класи, врши се поређење са задатим прагом који је у примеру 0.5 и резултат се методом *astype* конвертује у цео број (линија 61). Добијени вектор бинарних вредности (промењива

binarne_vrednosti) предсдтавља резултат предикције. У примеру је још проверена и тачност добијене предикције коришћењем *accuracy_score* функције (секција 8.3.1.2.1), која пореди векторе стварних класа и класа добијених предикцијом.

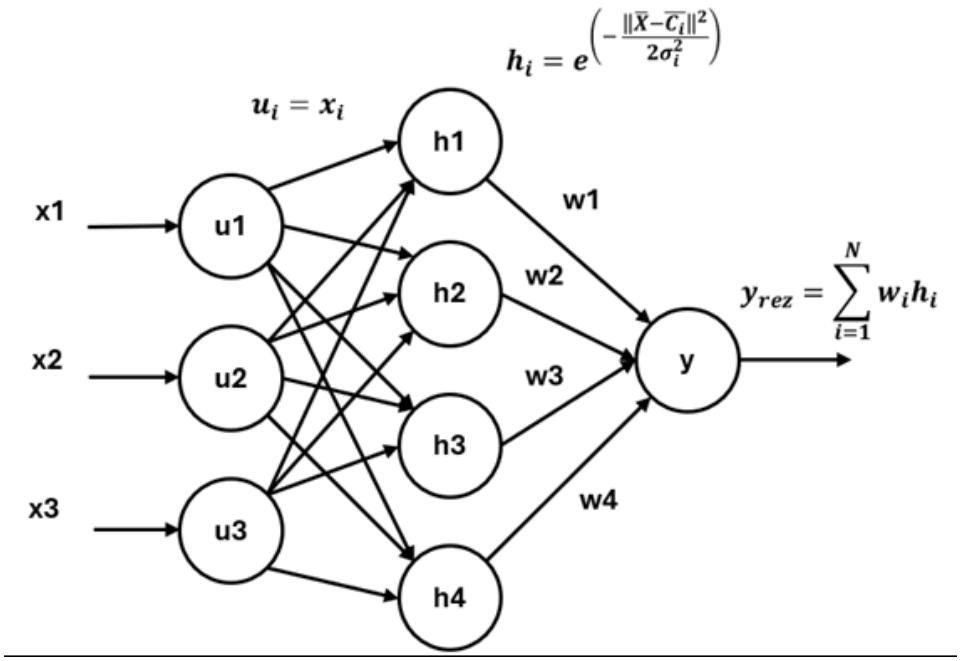
```
58  predikcije = model.predict(podaci_za_predikciju)
59  predikcije_matrica = numpy.array(predikcije)
60  predikcije_vektor=predikcije_matrica.reshape(-1)
61  binarne_vrednosti= (predikcije_vektor > 0.5).astype(int)
62  accr = accuracy_score(y_test,binarne_vrednosti)
63  print(f'Tacnost predikcije je {accr}')
```

Слика 9.13: коришћење модела једнослојног перцептрона

Једнослојни перцептрон може се имплементирати као кориснички дефинисана класа и без ослањања на Python модуле. Класа која има све описане методе перцептрона, која може да се обучава и даје могућност имплементације специфичног понашања. На тај начин се обезбеђује платформска независност, компактност и високе перформансе решења који могу да обрађују синхроно велике количине података.

9.4.2. Мреже на бази радијалних функција

Мреже на бази радијалних функција (енгл. *Radial Basis Function – RBF*) су трослојне вештачке неуронске мреже са храњењем у напред, што значи да се приликом коришћења мреже извршава једносмеран процес – од улаза према излазу. *RBF* мрежа се састоји (Слика 9.14) од улазног слоја неурона, којих има колико и улазних својстава. Неурони улазног (*u1, u2* и *u3*) слоја имају задатак да прослеђују улазне податке (*x1, x2* и *x3*) неуронима скривеног слоја (*h1, h2, h3* и *h4*) без трансформације. Сваки неурон скривеног слоја има онолико улаза колико има својстава, односно неурона у улазном слоју. Неурони скривеног слоја (*h1, h2, h3* и *h4*) обрађују улазне податке коришћењем неке радијалне функције, назване тако што је излазна вредност зависи од удаљености улазног податка од неке мере централне тенденције (нпр. центроида). У излазном слоју модела мреже на бази радијалних функција се налази само један неурон *y*, који користи податке из скривеног слоја, трансформисане на бази радијалне функције и помножене синаптичким тежинама w_i како би се добила резултујућа вредности y_{rez} као резултат сумирања. Та сума представља финални резултат рада читаве мреже на бази радијалне функције.



Слика 9.14: структура вештачке неуронске мреже на бази радијалне функције

Као радијална функција је најчешће коришћена Гаусова функција (секција 9.2), код које је (Израз 9.5) у експоненту количник квадрата еуклидских дистанци вектора улазних вредности $\bar{X}$ и вектора центроида (за свако својство) $\bar{C}_i$ за неурон h_i и у чијем имениоцу фигурише фактор σ_i који регулише опсег Гаусове функције (Гаусовог звона) за неурон h_i.

$$h_i = e^{\left(-\frac{\|\bar{X}-\bar{C}_i\|^2}{2\sigma_i^2}\right)} \qquad \text{И.9.5}$$

Вештачке неуронске мреже на бази радијалних функција су најједноставније вишеслојне мреже са храњењем у напред. Упркос једноставности, врло су значајне јер се често користе за функционалне апроксимације, а захваљујући једноставној структури, дају високе перформансе како у фази обучавања, тако и у фази експлоатације, што је значајно и у решавању проблема регресије и класификације.

9.4.2.1. Пример имплементације вештачке неуронске мреже на бази радијалне функције у Python-у

У примеру су дати синтетички генерисани подаци кретања акција на берзи током времена (Слика 9.15). Вредност независне промењиве *X* је вектор од 100 вредности на истој дистанци у интервалу [0,10]. Вредност зависне

промењиве *y* је добијена на основу синуса вектора *X* чијим елементима су додате насумично генерисане вредности из нормалне (Гаусове) дистрибуције са стандардном девијацијом 0.1 (параметар *scale*). Након тога су подаци подељени на подскуп за обуку (80%) и евалуацију модела (20%).

```
X = np.linspace(0, 10, 100).reshape(-1, 1)
y = np.sin(X).ravel() + np.random.normal(scale=0.1, size=X.shape[0])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Слика 9.15: синтетички генерисани подаци за обуку и евалуацију

За модел вештачке неуронске мреже на бази радијалне функције дефинисана је класа названа *RBFMreza* (Слика 9.16) која се конструише прослеђивањем броја улаза (параметар *br_ul_svojstava*), броја неурона у скривеном слоју, броја неурона у излазном слоју и вредности *сигма* фактора (стандардне девијације Гаусове расподеле).

```
rbf_mreza = RBFMreza(br_ul_svojstava=X_train_scaled.shape[1],
                     vel_skrivenog_sloja=10, br_izlaza=1, sigma=1.0)
```

Слика 9.16: конструкција мреже на бази радијалне функције

За обучавање модела дефинисана је *fit* функција којој се прослеђују подскупови података независне и зависне промењиве за обучавање (Слика 9.17). У дефиницији функције најпре се одређују центроиди (приватна функција *_daj_centroide*), затим се одређују вредности активационе функције за сваки неурон скривеног слоја (приватна функција *_izracunaj_aktivaciju*) и, коришћењем скаларног производа два вектора (промењиве *activation* и *y*) израчунавају се синаптичке тежине између неурона скривеног и излазног слоја.

```
rbf_mreza.fit(X_train_scaled, y_train)
```

```
def fit(self, X, y):
    self.centri = self._daj_centroide(X)
    activation = self._izracunaj_aktivaciju(X)
    self.tezine = np.dot(np.linalg.pinv(activation), y)
```

Слика 9.17: Функција за обучавање RBF мреже

Приватна функција *_izracunaj_aktivaciju* (Слика 9.18) као параметар прихвата објекат *X* који садржи улазне податке и израчунава вредности

активација за све неуроне скривеног слоја применом Гаусове функције (секција 9.2) и коришћењем центроида и *сигма* вредности (стандардне девијације Гаусове расподеле). Активације се враћају као истоимени низ (локална промењива *aktivacije*).

```
def _izracunaj_aktivaciju(self, X):
    aktivacije = np.zeros((X.shape[0], self.vel_skrivenog_sloja))
    for i in range(self.vel_skrivenog_sloja):
        aktivacije[:, i] = np.exp(-np.linalg.norm
                                  (X - self.centri[i], axis=1) ** 2 / (2 * self.sigma ** 2)
                                  )
    return aktivacije
```

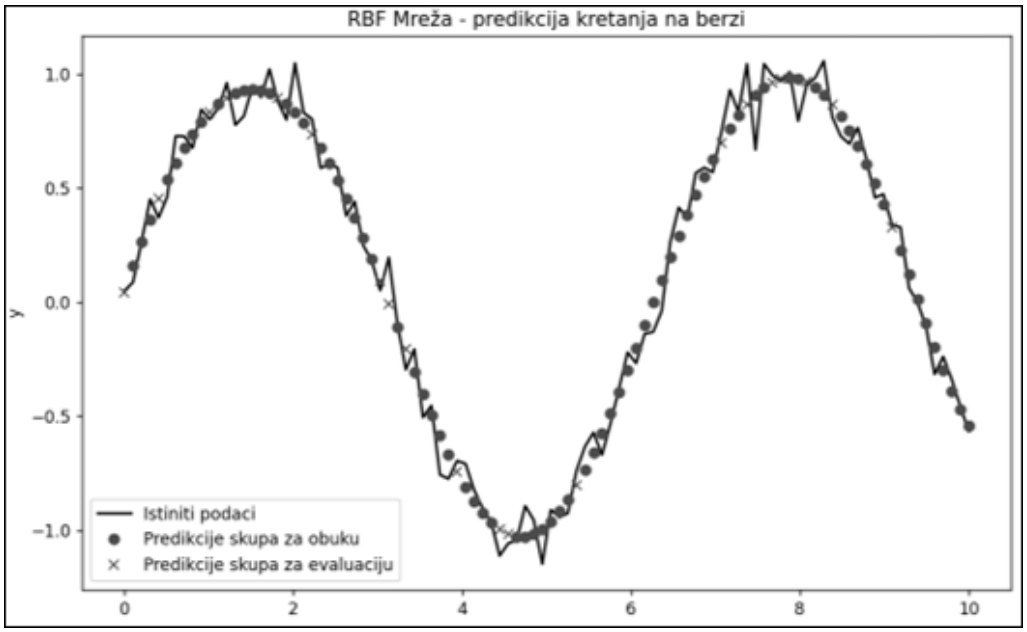
Слика 9.18: имплементација активационе функције као Гаусове функције код RBF мреже

Функција *_izracunaj_aktivaciju* се користи и приликом коришћења неуронске мреже (Слика 9.19). Као параметар функција прихвата објекат *X* који садржи улазне податке, рачуна активације и као коначан резултат враћа скаларни производ активација неурона скривеног слоја и тежина синаптичких тежина који ти неурони имају према излазном неурону.

```
def predict(self, X):
    aktivacije = self._izracunaj_aktivaciju(X)
    rezultat = np.dot(aktivacije, self.tezine)
    return rezultat
```

Слика 9.19: имплементација предикције RBF мреже

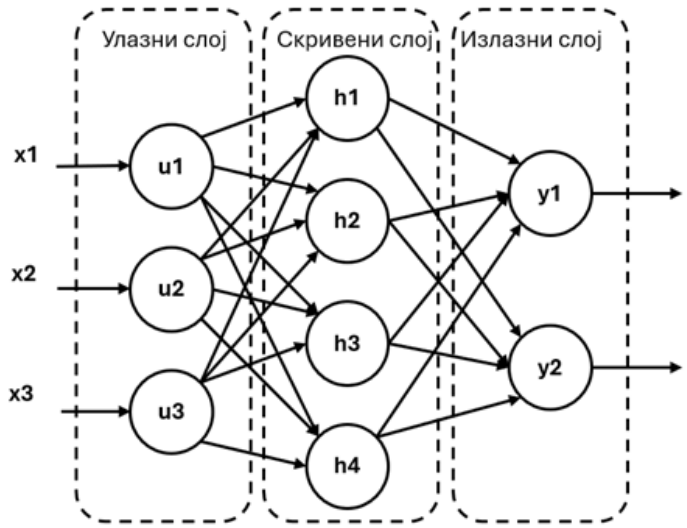
Визуелизација резултата рада модела за предикцију коришћењем класе *RBFMreza* приказан је на следећој слици (Слика 9.20). Обзиром да су подаци синтетички генерисани синусном функцијом, а затим модификовани рандом функцијом којој су добијена одступања од 0.1 од оригиналних (на слици означени као *истинити подаци*), може се закључити да су предикције RBF мреже заправо апроксимација синусне функције, која и представља основу у дистрибуцији података. Доказ томе је синусни облик добијене криве коју чине тачке, као предикције подскупа за обуку (тренинг) и крстићи, као предикције подскупа података за евалуацију (тест).



Слика 9.20: резултат коришћења RBF мреже из примера

9.4.3. Вишеслојни перцептрон

Вишеслојни перцептрони (енгл. *Multi-Layer Perceptron – MLP*) су вештачке неуронске мреже које имају више неурона организованих у слојеве (Слика 9.21) тако да постоји барем један улазни и један излазни слој. Поред тога мреже могу да имају један или више скривених слојева.



Слика 9.21: Структура вишеслојног перцептрона

Вишеслојни перцептрони се још називају и мреже са *храњењем у напред* (енгл. *Feedforward Neural Networks - FNN*) пошто се пропагација сигнала (података) врши од улаза према излазу из мреже, односно у улазни слој, кроз скривене слојеве до излазног слоја, који даје резултат класификације, или предикције.

9.4.3.1. Обучавање вишеслојног перцептрона

Вишеслојни перцептрони се обучавају (као и сви остали представљени модели за класификацију) на основу скупа већ класификованих података. За разлику од класификације, која представља процес који се врши само у напред, обучавање се врши и *у напред* и у *назад*. Обучавање се врши такозваним *алгоритмом за учење у назад* (енгл. *Back Propagation Algorithm*). Овај алгоритам има три фазе: у првој се иницијално постављају синаптичке тежине неурона у суседним слојевима. Затим се кроз мрежу пуштају подаци за обуку (фаза пропагације *у напред*). Након добијања вектора предикција за скуп улазних података, рачунају се одступања од очекиваних вредности, на основу којих се коригују синаптичке тежине *у назад* – од излазног према улазном слоју.

Због сложене структуре, обучавање вишеструког перцептрона је сложен итеративни процес обзиром да је сваки неурон вишеструко повезан са неуронима из суседних слојева, као и чињенице да сви слојеви утичу на резултате које мрежа као целина даје. Основни принцип је исти као у случају једноструког перцептрона (секција 9.4.1.1.1.), који се своди на упоређивање вредности које се добијају на излазу из мреже са очекиваним (тачним) вредностима. Разлика ове две вредности представља величину грешке, која се отклања тако што се *у назад* коригују вредности свих синаптичких тежина између слојева мреже. Процес се изводи у предефинисаном броју итерација, или када више нема напретка у минимизацији одступања (грешке).

Код вишеслојног перцептрона као функција губитка (грешка) узима се средња квадратна грешка (енгл. *Mean Square Error – MSE*) која се израчунава (Израз 9.6) за n предикција и n узорака у скупу за обучавање. При томе сваки узорак може да садржи вредности једног, или више својстава. Y_i представља вектор вредности предикција, док $\hat{Y}_i$ представља вектор очекиваних (тачних) вредности. Да би се добила средња вредност,

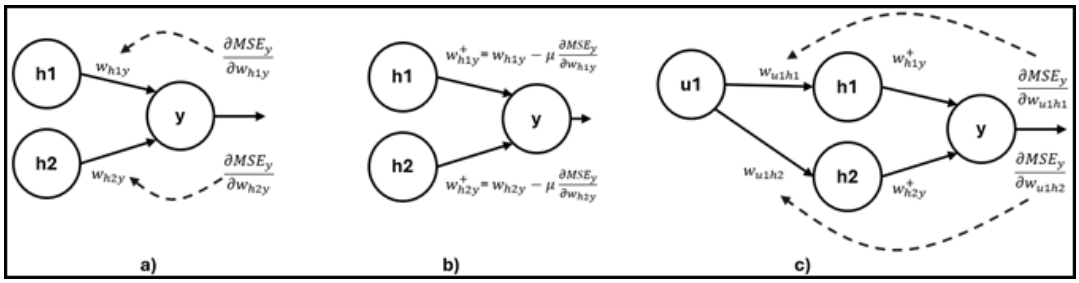
сума квадрата разлика ова два вектора се дели са бројем узорака у скупу за обучавање (n).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \qquad \text{И.9.6}$$

Вредност средње квадратне грешке се користи за корекцију синаптичких тежина *у назад*. Пошто су неурони излазног слоја једино повезани са неуронима претходног слоја (на пример, последњи скривени слој у мрежи), онда се методом парцијалних извода доводи у везу вредност грешке са одабраном синаптичком тежином. Коригована вредност неке синаптичке тежине w^+ (Израз 9.7) се добија тако што се њена постојећа вредност w умањује производом фактора μ (познат као *брзина учења*) и парцијалног извода средње квадратне грешке по вредност w.

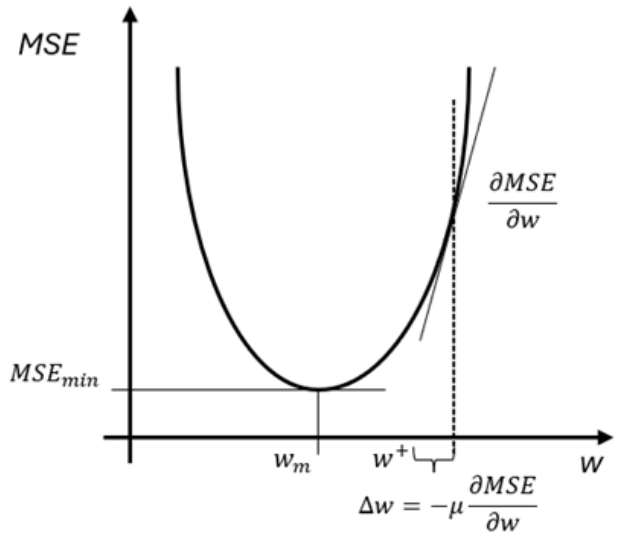
$$w^+ = w - \mu\frac{\partial MSE}{\partial w} \qquad \text{И.9.7}$$

У следећем примеру (Слика 9.22), на левој страни (*a*) представљен је неурон излазног слоја *y* који је повезан за два неурона скривеног слоја (*h1* и *h2*), тако да се средња квадратна грешка на *y* неурону доводи у везу са две синаптичке тежине (w_{h1y} и w_{h2y}). Кориговане вредности синаптичких тежина w_{h1y}^+ и w_{h2y}^+ добијају се парцијалним изводима средње квадратне грешке MSE_y по w_{h1y} и w_{h2y} (*b*). Након тога процес се понавља према улазном слоју (у назад). На слици (*c*) је због прегледности, приказан само један неурон (*u1*) улазног слоја који има две синапсе према неуронима скривеног слоја (*h1* и *h2*), чије су тежине (w_{u1h1} и w_{u2h2}). За њихову корекцију се такође користе парцијални изводи средње квадратне грешке MSE_y , али по w_{u1h1} и w_{u2h2}. Пошто MSE_y није у директној вези са w_{u1h1} и w_{u2h2}, користи се принцип уланчавања парцијалних извода. Важно је да у прорачунима парцијалних извода *у назад* не учествују кориговане вредности тежина у ланцу, већ оригиналне. У конкретном примеру то су вредности w_{h1y} и w_{h2y}.



Слика 9.22: обучавање мреже корекцијом синаптичких тежина у назад

Парцијални извод средње квадратне грешке се још назива и градијент (Слика 9.23). Као што је речено, процес обучавања се врши у итерацијама са циљем да се добије минимална средња квадратна грешка. Кроз итерације градијент се *спушта* (енгл. *Gradient descent*). Како парцијални извод одређује стрмину (*MSE*) функције (тангента), алгоритам за учење користи ту чињеницу како би нашао минимум средње квадратне грешке откривањем тачке у којој функција мења тенденцију (тангента паралелна са *x* осом).



Слика 9.23: Обучавање спуштом градијента средње квадратне грешке

Вредност брзине учења μ треба да буде усклађена са осталим факторима у алгоритму обучавања (природа података, величина скупа података, синапичке тежине мреже). Мале вредности μ доводе до малих промена синаптичке тежине, тако да захтевају велики број итерација процеса обучавања да би се пронашао стварни минимум средње квадратне грешке. Обрнуто је у случају великих вредности μ. Поред тога, велике вредности μ

могу да доведу до немогућности проналажења стварног минимума средње квадратне грешке (велики *кораци* који могу да доведу до *прескакања* вредности w_m). Насупрот томе, мале вредности μ имају већу вероватноћу проналажења стварног минимума средње квадратне грешке, односно одговарајуће вредности тежине w_m.

9.4.3.2. Пример имплементације вишеслојног перцептрона у Python-у

У примеру је имплементиран трослојни перцептрон (мрежа) за потребе класификације. Као скуп података су коришћени подаци из базе *iris* из јавно доступне библиотеке *sklearn.datasets*, који садржи 150 записа три врсте биљке ирис, одређених се четри својства (секција 8.3.2.1). Овај скуп је подељен функцијом *train_test_split* (модул *sklearn.model_selection*) у подскуп за обучавање мреже од 120 записа (80%) и подскуп за проверу мреже од 30 записа (20%).

Обзиром да мрежа треба да класификује податке у 3 класе, избор је да садржи 3 неурона у излазном слоју. За остале слојеве изабрано је 10 неурона у улазном слоју и 10 неурона у скривеном слоју (Слика 9.24). Као модел искоришћена је класа *Sequential* из *keras.models* библиотеке. Моделу су додавани *Dense* слојеви, код којих се формирају потпуне конекције неурона суседних слојева. Као активациона функција улазног и скривеног слоја искоришћена је *ReLU* функција (секција 9.2), док је као активациона функција излазног слоја искоришћена *softmax* функција. Функција *softmax* трансформише вектор резултата у расподелу вероватноћа између тих резултата. Обзиром да излазни слој има 3 неурона, који производе вектор од 3 резултата, коришћењем *softmax* функције се тај вектор резултата трансформише у дистрибуцију вероватноћа тако да је њихова сума један, а коначни резултат је класа чији резултат има највећу вероватноћу. Улазни слојо има специфициран *input_shape* параметар, како би мрежа била прилагођена формату улазних података (сваки запис има 4 вредности својстава)

```
model = Sequential()
model.add(Dense(10, input_shape=(4,), activation='relu', name='ulazni sloj'))
model.add(Dense(10, activation='relu', name='skriveni sloj'))
model.add(Dense(3, activation='softmax', name='izlazni sloj'))
```

Слика 9.24: конструкција модела вишеслојног перцептрона

Позивом методе *summary* 3на моделу добија се извештај о структури модела и повезаности неурона у слојевима (Слика 9.25). Пошто улазни и скривени слој имају по 10 неурона, њихов излазни формат (*Output Shape*) је 10. Исто тако излазни слој има 3 неурона па је излазни формат мреже 3. Број параметара улазног слоја је 50, што значи да има 4 својства која се доводе на 10 неурона, укупно 40 и 10 за вредност w_0 (*bias*) факторе неурона (секција 9.4.1.1). На исти начин се могу тумачити параметри за остале слојеве: скривени слој има 110 параметара (10x10 – на сваки неурон скривеног се доводе подаци из 10 неурона улазног слоја и 10 фактора w_0); излазни слој има 33 фактора (3x10 – на сваки неурон излазног се доводе подаци из 10 неурона улазног слоја и 3 фактора w_0 – за сваки неурон излазног слоја по један).

Neural Network Model Summary:
Model: "sequential"

Layer (type)	Output Shape	Param #
ulazni sloj (Dense)	(None, 10)	50
skriveni sloj (Dense)	(None, 10)	110
izlazni sloj (Dense)	(None, 3)	33

Total params: 193 (772.00 B)
Trainable params: 193 (772.00 B)
Non-trainable params: 0 (0.00 B)

Слика 9.25: сумарни извештај о моделу вишеслојног перцептрона

Вредност *None* код излазног формата указује да је број узастопних записа који се захватају у процесу обучавања варијабилан и да ће се оптимизовати према перформансама платформе на којој се модел обучава. Укупно модел садржи 193 параметара који се обучавају и који имају меморијско заузеће од 772 бајта, што значи да су вредности параметара кодирани са по 4 бајта (што одговара нумеричким типовима *int* и *float*). У случају проблема са перформансама модела, сумарни извештај је користан за сагледавање комплексности модела и евентуалне интервенције на његовој структури.

Након конструкције, модел се позивом функције *compile* конфигурише (Слика 9.26) за процес обучавања (секција 9.4.1.1.2). Као функција губитка користи се *categorical_crossentropy*, типична за неуронске мреже и класификације у више од две класе.

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Слика 9.26: конфиīурисање модела за īроцес обучавања

Модел је спреман за обучавање (Слика 9.27). Методи *fit* прослеђују се подаци за обуку (параметри *X_train* и *y_train*), дефинише се 200 итерација у којима ће се модел обучавати на задатим подацима (параметар *epochs*) и дефинише се ажурирање параметара модела на сваких пет узастопних записа података (параметар *batch_size*). Параметром *verbose* са вредношћу 0 одређује се да на излазном панелу у току процеса обучавања неће бити приказа детаља (што одузима додатно процесно време) у току обучавања.

```
model.fit(train_x, train_y, verbose=0, batch_size=5, epochs=200)
```

Слика 9.27: обучавање модела у 200 иīерација

Након обучавања модела, добијена је тачност од 93%, уз добијену функцију губитка од 0.167, што представља задовољавајуће резултате, који се могу даље унапређивати. Један од начина који неће нарушити стабилност модела је фактор μ – брзина учења (секција 9.4.1.2.1). Овај фактор може експлицитно да се подеси код *Adam* оптимизатора приликом конфигурације модела пре обучавања (Слика 9.28). У конкретном случају фактор μ је подешен на 10^{-3} што ће омогућити већу вероватноћу проналажења правог минимума функције губитка.

```
optimizer = Adam(learning_rate=0.001)
```

Слика 9.28: īодешавање брзине учења

Нови објекат *optimizer* се затим прослеђује *compile* функцији, ради поновне конфигурације модела. Након поновног обучавања, на моделу је добијена тачност од 100% и функција губитка од само 0.078, што представља знатно унапређене резултате евалуације модела.

Након обучавања модел може да се користи у класификацији. Као код већине класификатора, користи се метода *predict*, којој се прослеђује као улазни податак низ са вредностима 4 својства и у формату који се користио за обуку (Слика 9.29).

```
nazivi_klasa = ["Iris-setosa", "Iris-versicolor", "Iris-virginica"]

predikcije = model.predict(np.array([[5.4, 3.4, 1.5, 0.4]]))
indeks_klase = np.argmax(predikcije)

print(predikcije)
print(f'Na osnovu podataka vrsta biljke je {nazivi_klasa[indeks_klase]}')
```

Слика 9.29: коришћење модела вишеструког перцептрона у класификацији биљака

Обзиром да су у излазном слоју 3 неурона и да је коришћена *softmax* активациона функција, модел ће као резултат дати вектор (промењива *predikcije*) од три излазне вредности чија је сума један (Слика 9.30). Коришћењем *argmax* функције добија се индекс највеће вредности у вектору (промењива *indeks_klase*). У конкретном случају то је прва класа. Тај индекс се даље користи за добијање назива класе, како би се формирао податак који је читљив и разумљив људима.

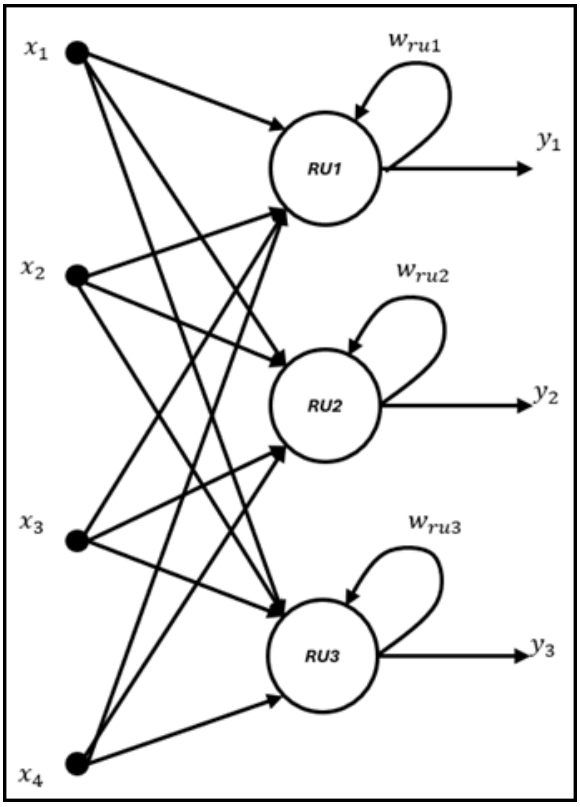
Слика 9.30: резултат класификације моделом вишеструког перцептрона

Како би се унапредили резултати класификатора, неретко није довољно само променити брзину учења, већ су потребне интервенције над осталим параметрима модела. Боље перформансе модела могу да се добију и избором одговарајућег броја итерација у процесу учења, или променом активационе функције у слојевима мреже. Промене не треба да буду велике и треба да се врше само на једном параметру. Након промене се понављају фазе конструкције, конфигурисања, учења и евалуације модела. Ако промене и даље не доводе до унапређења перформанси модела, треба размотрити промене у структури модела. Ако ни ове промене не доведу до резултата, постоје две могућности: промена модела за класификацију, или унапредити скупо података за обуку и евалуацију модела.

9.4.4. Рекурентне неуронске мреже

Рекурентне мреже (енгл. *Recurrent neural networks - RNN*) су вештачке неуронске мреже које у структури садрже повратне спреге (Слика 9.31) како

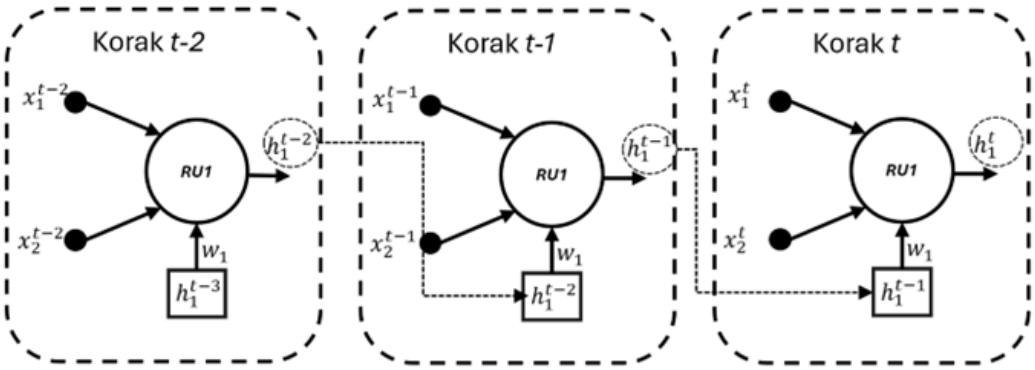
би се обрађивали секвенцијални подаци као што су временске серије података, обрада текста или природног говора. Неурони у рекурентној неуронској мрежи се називају рекурентним јединицама (енгл. *Recurrent unit* – *RU*). Рекурентна јединица садржи такозвано *скривено стање* – меморију која се ажурира преко повратне спреге, која такође представља синапсу, има тежину w_{ruj}, при чему *j* представља индекс неурона. На тај начин у одређеној мери врши се *памћења* података који су претходили онима који су тренутно на улазу *RU*.



Слика 9.31: Рекурентна неуронска мрежа

Због повратне спреге рекурентна јединица ради у итеративном режиму. Код рекурентних мрежа једна итерација се представља једним *кораком* обраде улазних података. На следећем примеру су приказана 3 корака (итерације) рекурентне јединице *RU1*(Слика 9.32). Ако је актуелни корак *t* онда је скривено стање рекурентне јединице посматрано кроз време као што је на приказу дато. Види се да је у израчунавању стања h_1^t поред улазних података x_1^t и x_2^t утицало и претходно стање h_1^{t-1} *RU1* јединице. На исти начин је на израчунавање стања h_1^{t-1}, поред улазних података x_1^{t-1} и x_2^{t-1}

утицало и претходно стање h_1^{t-2}. На израчунавање стања h_1^{t-2}, поред улазних података x_1^{t-2} и x_2^{t-2} утицало и претходно стање h_1^{t-3}. Овај процес се назива *одмотавање у назад кроз време* (енгл. *Unwrapping*, или *Unrolling Backpropagation through time – BPTT*). Одмотавањем рекурентне мреже се добија аспект вештачке неуронске мреже са храњењем у напред, тако да се начин обучавања код мреже са храњењем у напред (секција 9.4.1.2.1) може применити и на рекурентне мреже .



Слика 9.32: „Одмотавање“ рада рекурентне јединице

Општи израз за скривено стање h_j^t рекурентне јединице j у кораку t (Израз 9.8) добија се као резутлат активационе функције f_a којој се као параметар прослеђује не само фактор w_0 (*bias*) и сума производа синаптичких тежина w_i и улазних вредности x_i^t (i-ти улаз од укупно N улаза, у кораку t), већ садржи и производ претходно запамћеног стања h_j^{t-1} (у претходном кораку $t-1$) и синаптичке тежине повратне спреге w_j рекурентне јединице j.

$$h_j^t = f_a\left(\sum_{i=1}^{N} w_i \cdot x_i^t + w_j \cdot h_j^{t-1} + w_0\right) \qquad \text{И.9.8}$$

Код рекурентних неуронских мрежа, као активационе функције најчешће се користе сигмоидна, хиперболички тангенс и комбинована функција са прираштајем (*ReLU*) (секција 9.2).

9.4.4.1 Пример имплементације рекурентне неуронске мреже у Python-у

У примеру, за имплементацију рекурентне неуронске мреже коришћен је као модел – објект класе *Sequential* из модула *keras* и као рекурентни слој – објекат класе SimpleRNN из модула *keras.layers* (Слика 9.33). Рекурентни слој садржи 50 неурона који *ReLU* функцију користе као активациону. Као

скуп података за обуку и евалуацију модела синтетички су генерисани подаци, 1000 узорака са по 1 својством, који су груписани у временским серијама које садржи 10 корака (на пример, у неком практичном примеру то би било 10 година). Из тог разлога је примарна *временска* димензија као улазни формат података (први параметар промењиве *input_shape*). Обзиром да овако конструисана мрежа има 50 излаза, ради демонстрације додат је излазни слој са једним неуроном како би била добијена једна излазна вредност. Ако би се модел користио за вишеструку класификацију, број неурона у излазном слоју би био већи.

```
model = keras.Sequential([
    layers.SimpleRNN(50, activation='relu', input_shape=(X_train.shape[1], 1)),
    layers.Dense(1)
])
```

Слика 9.33: Конструкција модела рекурентне неуронске мреже

Позивом методе *summary* добија се извештај о структури и параметрима модела (Слика 9.34). Види се да има 50 рекурентних јединица (неурона). Број параметара за скривени рекурентни слој је 2600. У тај број улази:

1. 50 тежина за улазне синапсе – постоји једно својство које се доводи на улазе 50 рекурентних јединица
2. 2500 (50x50) – матрица рекурентних тежина која повезује претходно скривено стање и текуће скривено стање за сваку рекурентну јединицу и
3. 50 фактора w_0 (*bias*) – за сваку рекурентну јединицу.

У излазном слоју се налази 51 параметар, од којих је 50 тежина за синапсе из рекурентних јединица скривеног слоја и један за фактор w_0 (bias).

Model: "sequential"

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 50)	2,600
dense (Dense)	(None, 1)	51

Total params: 2,651 (10.36 KB)
Trainable params: 2,651 (10.36 KB)
Non-trainable params: 0 (0.00 B)

Слика 9.34: сумарни извештај о моделу рекурентне неуронске мреже

Након конструкције, модел се позивом функције *compile* конфигурише (Слика 9.35) за процес обучавања (секција 9.4.1.1.2). Као функција губитка у овом случају је коришћена средња квадратна грешка (*MSE*) (секција 9.4.1.2.1). Обучавање модела се извршава позивом функције *fit* у 50 итерација (*epochs*), при чему се 90% од скупа података користи за обучавање, док се за евалуацију користи 10% података. За евалуацију је, уместо тачности (*accuracy*) коришћени губици (промашаји у предикцији). Након евалуације на основу 100 узорака добијена је вредност 0.0004 која представља средњу квадратну грешку. Ова вредност је постигнута захваљујући малој стандардној девијацији и периодичности података из узорка.

```
model.compile(optimizer='adam', loss='mean_squared_error')
history = model.fit(X_train, y_train, epochs=50,
                    batch_size=32, validation_split=0.1, verbose=0)
loss = model.evaluate(X_test, y_test, verbose=0)
print(f"Evaluacija - MSE: {loss:.4f}")
```

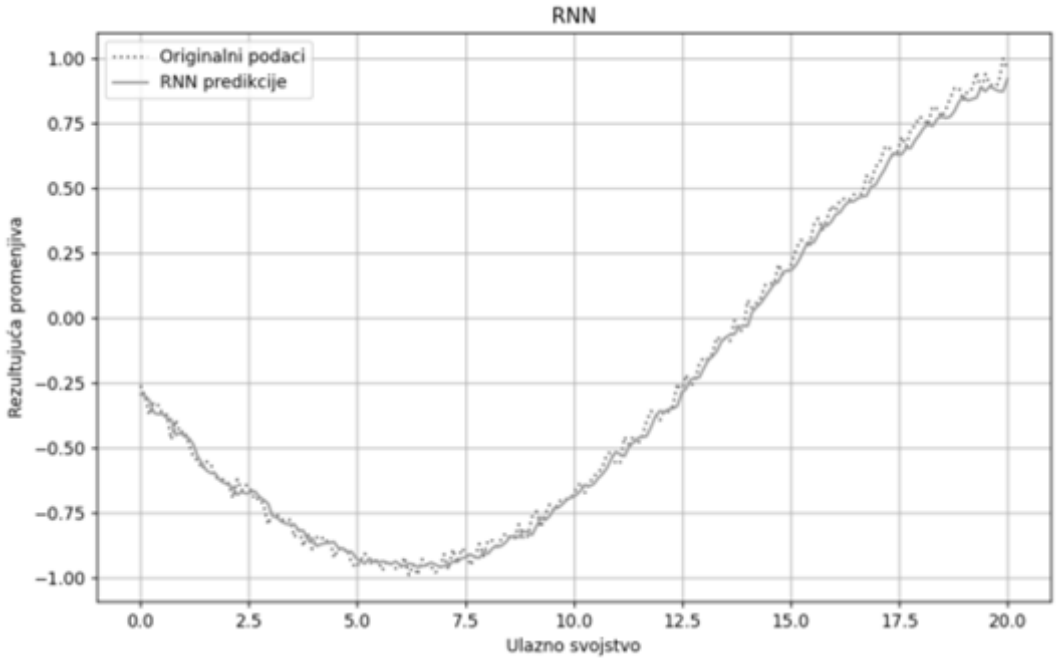
Слика 9.35: конфигурисање, обучавање и евалуација модела рекурентне мреже

Модел је након евалуације припремљен за коришћење. Као код већине класификатора, користи се метода *predict*, којој је прослеђен вектор улазних података који се користио за обуку (Слика 9.36). Добијене предикције и очекиване вредности су затим трансформисане из нормализованих у оригиналне вредности ради приказа.

```
predikcije = model.predict(X_test)
```

Слика 9.36: коришћење модела рекурентне неуронске мреже

Приказ 100 стварних података и 100 резултата добијених предикцијом уз помоћ модела рекурентне неуронске мреже (Слика 9.37) указује на мале разлике између података (реда су 10^{-2}). Оно што се још види на дијаграму је да су трендови стварних и података добијених предикцијом врло слични.



Слика 9.37: поређење предикција и стварних вредности података из примера

У представљеном примеру модел рекурентне мреже има архитектуру *један према више* обзиром да се на улаз доводе временске серије (секвенце од 10 узастопних података) и један излазни вектор података. Поред наведене, постоје још и *један према више, више према један и више према више* архитектуре.

9.4.4.2. Архитектуре рекурентних неуронских мрежа

Модели један према један су модели без секвенци података (без временских серија), тако да функционишу као мреже са храњењем у напред. Немају значајну практичну примену, али се понекад користе за провере и упоређивања са другим врстама модела за предвиђање и класификацију.

Модели један према више имају један улазни вектор и производи вредности на више излаза (више – димензиони простор података). Улазни вектор представља иницијалну секвенцу, или контекст који модел даље користи на пример, за генерисање текста, звука, за вишеструка обележавања на слици као улазном податку.

Модели више према један имају више-димензиони формат улазних података (на пример временске секвенце за више својстава) и један

излазни резултат. Погодни су за предикције временских серија, за препознавање сентимента на основу текста (улазне секвенце речи), или на основу садржаја говора.

Модели више према више имају више-димензиони формат улазних и излазних података. Користе се за сложене трансформације (на великом броју својстава) временских серија. На пример за обраду слике, видеа или звука (избацивање шума, додавање различитих ефеката и сл.)

9.4.5. Мреже дугорочне и краткорочне меморије

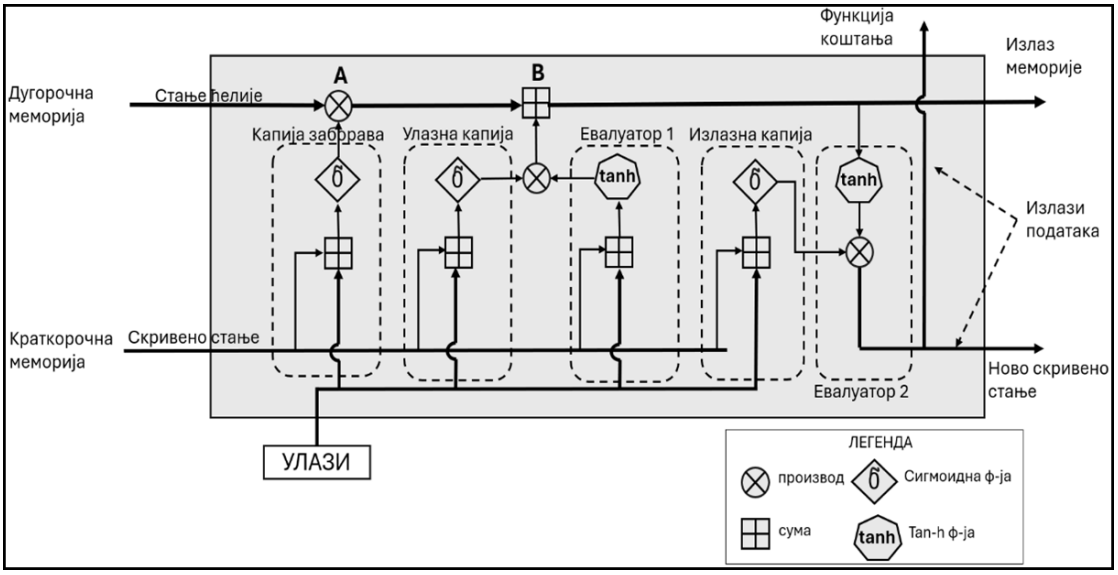
У примени рекурентних неуронских мрежа може се појавити проблем са градијентом учења изазваног избором начина регулације синаптичких тежина мреже. Ако су вредности тежине веће од 1, може доћи до хиперболичног раста градијента учења за дугачке временске серије. Обрнуто, ако су вредности тежине у интервалу [0,1], код дугачких временских серија може доћи до добијања вредности блиских нули, или потпуног ишчезавања градијента учења. Ради превазилажења овог проблема, у дизајниране су посебне мреже дугорочне и краткорочне меморије (енгл. *Long-Short Term Memory - LSTM*) .

Мреже дугорочне и краткорочне меморије (у даљем тексту *LSTM* мреже) представљају специфичан тип рекурентних неуронских мрежа којима се постижу унапређења за откривање и учење дугорочних узајамних зависности великог броја својстава у временским серијама (представљених секвенцијалним подацима). Рекурентна јединица код *LSTM* мреже назива се *ћелија* (Слика 9.38) и она садржи 2 компоненте: *дуго̄рочну меморију* и *крат̄корочну меморију*. Дугорочна меморија је одговорна за чување стања ћелије. Дугорочна меморија садржи суматоре и мултипликаторе и нема тежинске факторе. С друге стране, краткорочна меморија садржи два тежинска фактора као регуларна рекурентна јединица: за улазни податак и за повратну синапсу ћелије. Краткорочна меморија је одговорна за чување скривеног стања чија се вредност сумира са улазном вредношћу у свим елементима ЛСТМ структуре. У структури ЛСТМ се издвајају компоненте: *ка̄пија заборава*, *улазна ка̄пија, излазна ка̄пија* и два *евалуа̄тора*.

Капија заборава, улазна и излазна капија имају потпуно исту структуру, али различиту намену. Изузев другог евалуатора, све остале компоненте

сумирају улазне податке са скривеним стањем ћелије и затим их обрађују сигмоидном функцијом као излазном активационом функцијом компоненте. Евалуатори процесирају (*процењују*) податке на основу хиперболичке тангенс функције.

Капија заборава „одлучује" на основу улазног податка да ли ће текуће стање ћелије бити занемарено (анулирано), или ће утицати на процес тренирања ћелија у текућем кораку (мултипликатор А). Улазна капија на основу улазних података прави производ са излазном вредношћу евалуатора 1 и том вредношћу (суматором Б) се ажурира стање ћелије. Вредности из излазне капије се процењује у евалуатору 2 тако што се текуће стање ћелије обрађује *tan-h* функцијом а затим се множењем те две вредности добија новог скривено стање ћелије. Ново скривено стање ћелије је уједно и вредности функције трошкова. Нова вредност стања ћелије и нова вредност скривеног стања се користе заједно са улазном вредношћу у следећој итерацији у процесу обучавања модела. Вредностима функције трошкова *LSTM* ћелија процењује се обученост модела. Вредности функције трошкова се користе и у корекцији тежинских фактора на улазима ћелије (скривено стање и улазни подаци) за све капије.



Слика 9.38: структура ћелије LSTM мреже

Намена сигмоидне и *tan-h* функције је корективна у смислу да спречавају нагле промене излазних података у случају великих варирања улазних података. Без обзира на вредности улазног податка, *tan-h* функција даје

излазну вредност не мању од минус један у не већу од један. При томе, нелинеарност функције делује тако да улазне податаке у близини координатног почетка само пресликава на излаз ћелије без слабљења, док остале податке слаби више уколико је улазни податак даљи од координатног почетка. Сигмоидном функцијом се регулише у ком проценту се податак памти у дугорочну меморију тако што се мале промене памте у већем проценту од великих. Другим речима, сигмоидна функција компензује варирања која могу дестабилизовати модел мреже.

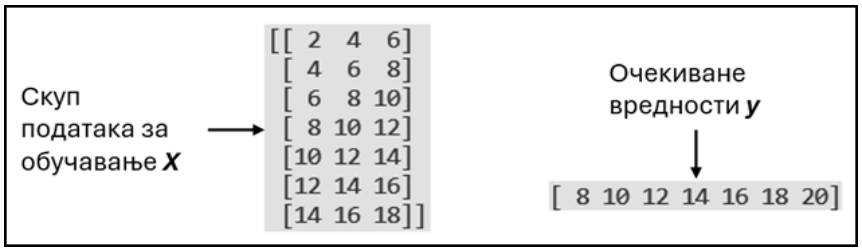
9.4.5.1. Пример имплементације LSTM мреже у Python-у

За имплементацију LSTM неуронске мреже у *Python*-у коришћен је као модел – објект класе *Sequential* из модула *keras* у који је као рекурентни слој угњежден објекат класе *LSTM* из модула *keras.layers* (Слика 9.39). Рекурентни слој садржи 50 неурона који *ReLU* функцију користе као активациону. Параметар *input_shape* временску димензију узима као примарну. LSTM мрежа је у модел угњеждена као скривени слој који има 50 излаза, тако да је додат један *Dense* објекат као излазни слој који садржи један неурон како би добили једну излазну вредност. За конфигурацију мреже (функција *compile*) је коришћен *adam* оптимизатор (секција 9.4.1.1.2) и средња квадратна грешка (*MSE*) (секција 9.4.1.2.1) као функција губитка.

```
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(X.shape[1], 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

Слика 9.39: конструкција модела LSTM мреже

Као скуп података за обуку модела (објекат *X*) синтетички су генерисани подаци груписани у 7 временских серија које садрже податке за 1 својство (Слика 9.40). Очекиване вредности за сваку временску серију су дате у посебном вектору (објекат *y*).



Слика 9.40: изглед скупа података за обуку

Обучавање се врши позивом методе *fit* којој се прослеђују подаци за обучавање и очекиване вредности. У примеру је коришћено обучавање у 200 епоха (Слика 9.41).

```
model.fit(X, y, epochs=200, verbose=0)
```

Слика 9.41: обучавање модела LSTM мреже

Евалуација модела би се вршила на потпуно исти начин као у примеру модела рекурентне мреже (секција 9.4.2.1.1), са напоменом да би се морао генерисати већи скуп података, из ког би се издвојио део података намењен евалуацији. Коришћење модела се врши позивом методе *predict* којој се прослеђује временска секвенца за предикцију (Слика 9.42). Као улазни податак коришћен је вектор који, за разлику од скупа за обучавање, садржи реалне бројеве код којих су вредности суседа на различитим дистанцама. На пример, дистанца прва два податка (6.5 и 7.1) је 0.6, док је дистанца између друга два податка већа (0.9).

```
ulazna_serija = np.array([6.5, 7.1, 8])
ulazna_serija = ulazna_serija.reshape(1, ulazna_serija.shape[0], 1)
predikcija = model.predict(ulazna_serija, verbose=0)
```

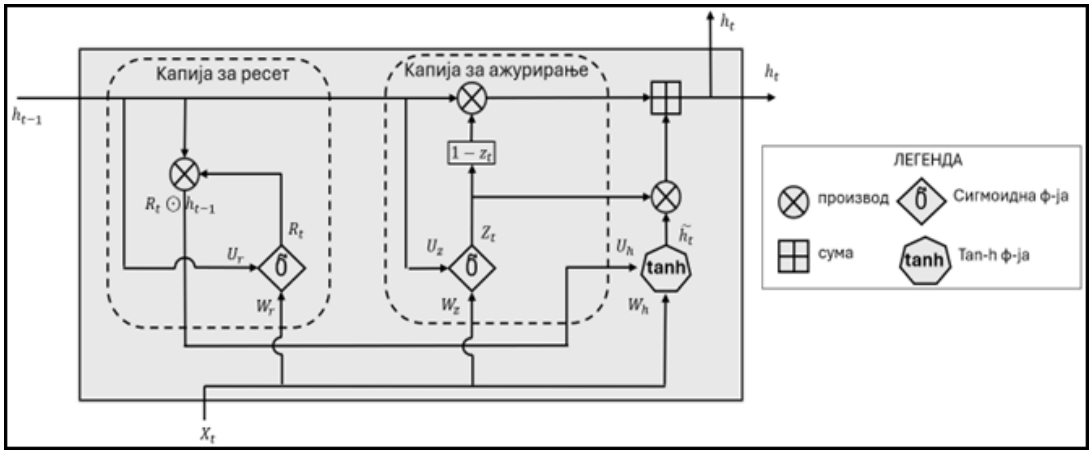
Слика 9.42: коришћење модела LSTM мреже за предикцију

Добијен резултат предикције за наведене улазне вредности је 9.54, што значи да је предвиђање да је 9.54 вредност која ће се појавити у случају временске серије (6.5, 7.1, 8).

9.4.6. Мреже рекурентних јединица са капијама

Мреже рекурентних јединица са капијама (енгл. *Gated Recurrent Units - GRU*) представљају посебан тип неуронских мрежа са повратном спрегом. Рекурентне јединице са капијама садрже две компоненте (Слика 9.43):

капију за ресет и капију за ажурирање. Капија за ресет одређује умањује (инхибира) претходно стање јединице h_{t-1}, који се затим прослеђује *tan-h* функцији како би се генерисао кандидат за ново стање $\widetilde{h_t}$. Капија за ажурирање на основу улазних података одређује део $\widetilde{h_t}$ који ће се пренети у ново стање јединице h_t. Основа за функционисање рекурентне јединице са капијама су матрице тежина $W_r, W_z,\ W_h, W_{r0}, W_{z0}, W_{h0}, U_r, U_z, U_h$ и вектори x_t, h_t, h_{t-1}. Матрице тежина улазних синапси (W) имају димензију $N \times M$ где је N број својстава вектора улазних података и M број GRU јединица у рекурентном слоју мреже. Матрице тежина скривених синапси (U) имају димензију $M \times M$ где је M број *GRU* јединица у рекурентном слоју мреже. Вектор x_t има елементата колико има својстава, а вектори скривених стања h имају елемената колико има GRU јединица у рекурентном слоју мреже.



Слика 9.43: структура рекурентне јединице са капијама

Функција капије за ресет R_t у тренутку (кораку) *t* је сигмоидна функција (Израз 9.9), која као параметар има суму производа вектора улазних вредности x_t и синаптичких тежина улаза W_r, затим производа синаптичких тежина јединица скривеног слоја U_r (у кораку *t-1*) и претходног стања јединица h_{t-1} и фактора W_{r0} (*bias* капије).

$$R_t = \sigma(W_r x_t + U_r h_{t-1} + W_{r0}) \qquad \text{И.9.9}$$

Аналогно капији за ресет, функција капије за ажурирање Z_t у тренутку (кораку) t се може описати формулом (Израз 9.10) у којој доминира сигмоидна функција која као параметар има суму производа вектора улазних вредности својстава x_t и синаптичких тежина улаза јединица

скривеног слоја W_z, производа јединица скривеног слоја U_z (у кораку *t-1*) и претходног стања јединице h_{t-1} и фактора W_{z0} (*bias* капије).

$$Z_t = \sigma(W_z x_t + U_z h_{t-1} + W_{z0}) \qquad \text{И.9.10}$$

Вектор кандидата за ново стање $\widetilde{h_t}$ добија се *tan-h* функцијом (Израз 9.11) која као параметар има суму производа вектора улазних вредности x_t и синаптичких тежина улаза јединица скривеног слоја w_h, производа синаптичких тежина улаза јединица скривеног слоја U_h (у кораку *t-1*) са Хадамардовим [1]производом ($\odot$) елементата вектора претходних стања јединица скривеног слоја h_{t-1} и вектора вредности функције капије за ресет R_t и фактора W_{h0} (*bias*).

$$\widetilde{h_t} = tanh(W_h x_t + U_h(R_t \odot h_{t-1}) + W_{h0}) \qquad \text{И.9.11}$$

На крају, ново стање h_t представља суму (Израз 9.12) Хадамардовог производа комплемента вектора вредности функције капије за ажурирање Z_t и вектора претходних стања јединице h_{t-1} и Хадамардовог производа вектора Z_t и вектора кандидата за ново стање $\widetilde{h_t}$.

$$h_t = (1 - Z_t) \odot h_{t-1} + Z_t \odot \widetilde{h_t} \qquad \text{И.9.12}$$

Мреже рекурентних јединица са капијама се такође користе за обраду секвенца (серија) података. Захваљујући једноставнијој структури, мреже рекурентних јединица са капијама су брже за обучавање и ефикасније од *LSTM* мрежа за једноставније задатке предикције.

9.4.6.1. Пример имплементације мреже рекурентних јединица у Python-у

За имплементацију мреже рекурентних јединица у *Python*-у коришћен је као модел – објект класе *Sequential* из модула *keras* у који је као рекурентни слој угњежден објекат класе *GRU* из модула *keras.layers* (Слика 9.44). Рекурентни слој садржи 50 неурона. Параметар *input_shape* временску димензију узима као примарну. Пошто је мрежа у модел угњеждена као скривени слој који има 50 излаза, да је додат један *Dense* објекат као излазни слој који садржи један неурон како би добили једну излазну вредност. За конфигурацију мреже (функција *compile*) је коришћен *adam* оптимизатор (секција 9.4.1.1.2) и средња квадратна грешка (*MSE*) (секција

[1] Хадамардов производ множи 2 матрице истих димензија тако што се множе елементи на истим позицијама, дајући матрицу истих димензија као резултат

9.4.1.2.1) као функција губитка. Као скуп података за обуку модела (објекат *X*) синтетички су генерисани подаци коришћењем синусне функције – 980 записа временских серија величине 20, што значи да има 20 вредности једног својства у једној серији. Очекиване вредности (980) за сваку временску серију су дате у посебном вектору (објекат *y*).

```
model = Sequential([
    GRU(50, input_shape=(velicina_serije, 1)),
    Dense(1)
])
model.compile(optimizer=Adam(learning_rate=0.01), loss='mse')
model.fit(X, y, epochs=20, batch_size=16, verbose=1)
```

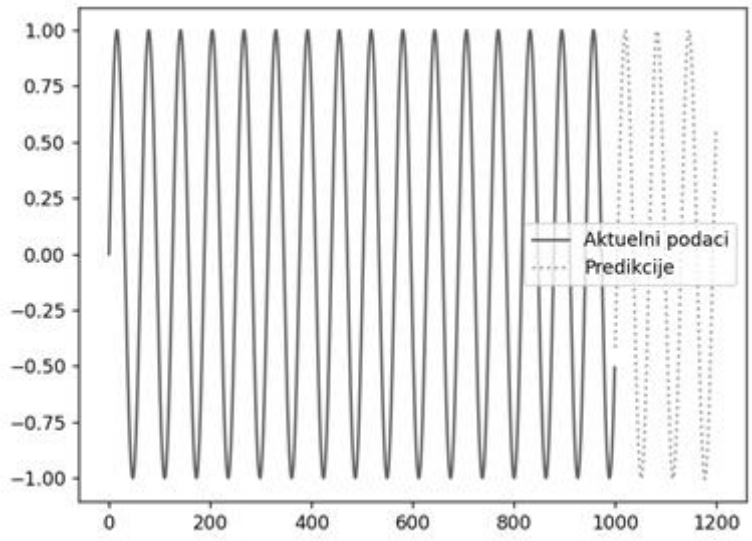
Слика 9.44: конструкција и конфигурација модела мреже рекурентних јединица

Евалуација модела би се вршила на потпуно исти начин као у примеру модела рекурентне мреже (секција 9.4.2.1.1). Коришћење модела се врши позивом методе *predict* којој се прослеђује временска секвенца за предикцију (Слика 9.45). Као улазни податак коришћен је вектор од 20 улазних вредности, коме је најпре прилагођен формат *predict* функцији.

```
x_formatiran = x_vrednosti_za_predikciju.reshape((1, velicina_serije, 1))
predikcija_y = model.predict(x_formatiran, verbose=0)
```

Слика 9.45: коришћење модела мреже рекурентних јединица

Визуелизација резултата (Слика 9.46) приказује синусну функцију стварних података (пуна линија) и предикције (тачкаста линија). Може се закључити да одступања предикције нема, што је очекивано обзиром да се ради о великом скупу података за обучавање (20000) генерисаних просто-периодичном синусном функцијом.



Слика 9.46: резултат предикције наредних 200 узорака података

За коришћење модела мреже рекурентних јединица у случајевима временских серија сложенијих података, са великим бројем својстава, потребно је да се, поред евалуације модела, изврши и емпиријска провера модела на различитим скуповима стварних података.

9.5. Конволуционе мреже

Конволуционе мреже (енгл. *Convolution Neural Networks - CNN*) су вештачке неуронске мреже са храњењем у напред, специјализоване за препознавање образаца у подацима који имају мрежну структуру, као што су слике и видео записи. Њихова архитектура се зато знатно разликује него код свих претходно описаних вештачких неуронских мрежа. За *CNN* мреже се каже да представљају надградњу вишеслојног перцептрона са новим функционалним и структурним решењима којима се контролише комплексност и оптерећење обрадом визуелних података. Конволуционе мреже нашле су примену у класификацији слика, препознавању објеката на сликама и сегментацији слика, анализи видео записа и других временских серија података.

9.5.1. Појам канала при обради слике

Слика у боји се састоји од тачака (енгл. *pixels*) и свака од тачака има црвену, зелену и плаву компоненту којима гради боју тачке (енгл. *Red, Green, Blue – RGB* компоненте). Канал представља сиво скалирану репрезентацију слике

добијену од једне компоненте боје На следећем примеру (Слика 9.47) лево је оригинална слика, док су десно слике 3 канала добијене из црвене, зелене и плаве компоненте слике. Оригинална слика има 328 x 227 тачака, што представља мрежну (матричну) структуру од 74456 тачака (поља). Свака тачка има 3 компоненте боје по 1 бајт, тако да је укупан број података који треба да се обради на слици 223368 – што би представљао потребан број неурона улазног слоја вештачке неуронске мреже како би се извршила обрада слике ради класификације, сегментације, или препознавања. Декомпозицијом у канале слика у сваком каналу има три пута мању комплексност (74456 података) пошто је нијанса сиве сваке тачке описана податком од 1 бајта.

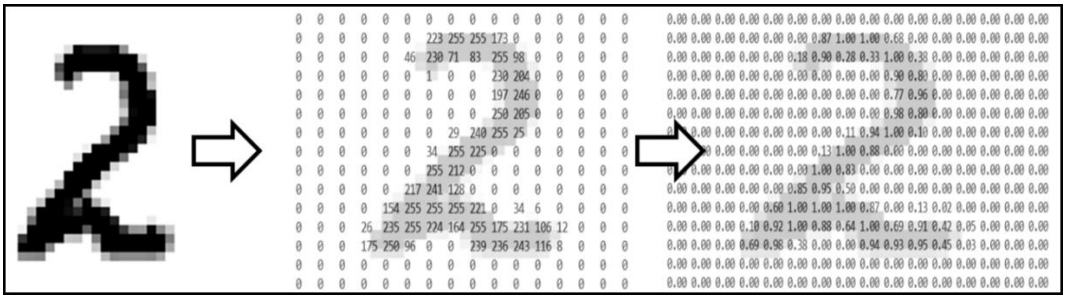
Слика 9.47: оригинална слика и каналске слике компоненти

Конволуционе мреже омогућавају да се каналске слике, иако су генерисане филтрирањем и кодирањем (сива скала 0-255), у конволуционим мрежама обрађују посебно. Бољим проучавањем каналских слика може се закључити да се међусобно разликују и да је контраст највише изражен код црвене компоненте, зелена компонента има највеће осветљење, док је плава компонента најтамнија и слабије израженим контрастом од црвене компоненте. Ове карактеристике су важне у даљој обради слике конволуционим мрежама.

9.5.2. Принцип конволуције

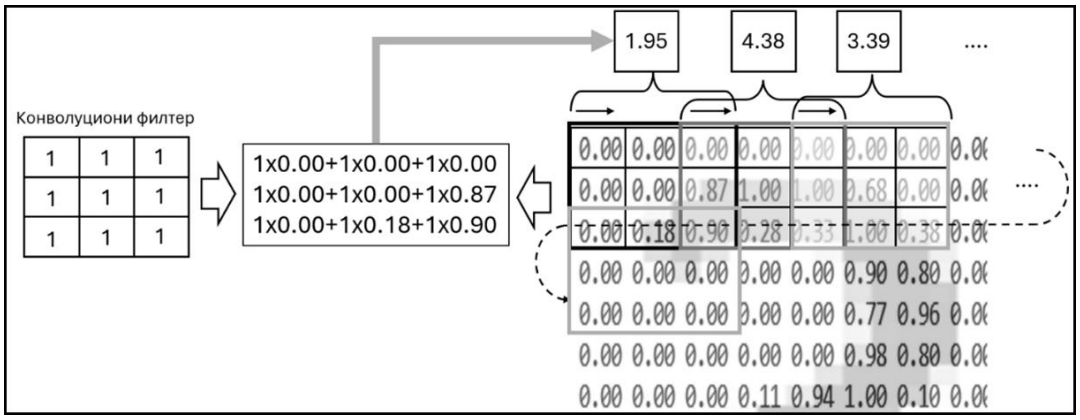
У машинском учењу, конволуција представља математичку операцију уз помоћ *малих* матрица како би се сложена мрежна структура (на пример слика) трансформисала у мање комплексну мапу својстава. Наведене *мале* матрице се називају конволуциони филтери (енгл. *kernel*). Конволуциони филтер обрађује улазни податак тако да се из њега потисне не-информациони садржај, а истакну својства битна за даљу обраду. На пример, улазни податак је слика (15x16 тачака) која садржи запис цифре 2 (двојка) и представљена је скалираним бајтовским записом сиве нијансе

тачака (Слика 9.48). Свака од 240 тачака је кодирана вредношћу у интервалу [0,255]. Да би се обезбедила очигледност, извршено је инвертовање вредности нијансе сваке тачке ($p' = 255 - p$), тако да је највећа вредност 255 – црна уместо беле и обрнуто (средња илустрација). У следеће, кораку се врши скалирање како би се вредности нашле у интервалу [0,1] и припремиле за даљу обраду (десна илустрација).



Слика 9.48: нумеричка репрезентација сиво скалиране слике цифре 2

Слика је припремљена за обраду конволуционим филтером (кернелом), матрицом 3x3 поља чије су вредности јединице (Слика 9.49). На примеру је приказан средњи горњи фрагмент слике. Обрада се своди на сумирање производа вредности кернела и тачке слике који су на истој позицији. За сваки померај кернела, обавља се 9 множења јединицом и смештање суме у поље нове матрице – мапе својстава. Резултујућа матрица својстава се још назива и *конволуција*. Кернел се помера (енгл. *slide*) у десно. Корак померања (енгл. *stride*) одређује димензију резултујуће матрице. У конкретном примеру изабран је корак 2, тако да се помера за две тачке у десно и на доле. Када кернел заврши један ред, помера се на доле за предефинисан корак и у лево, након чега наставља да обрађује нови ред тачака слике на исти начин.

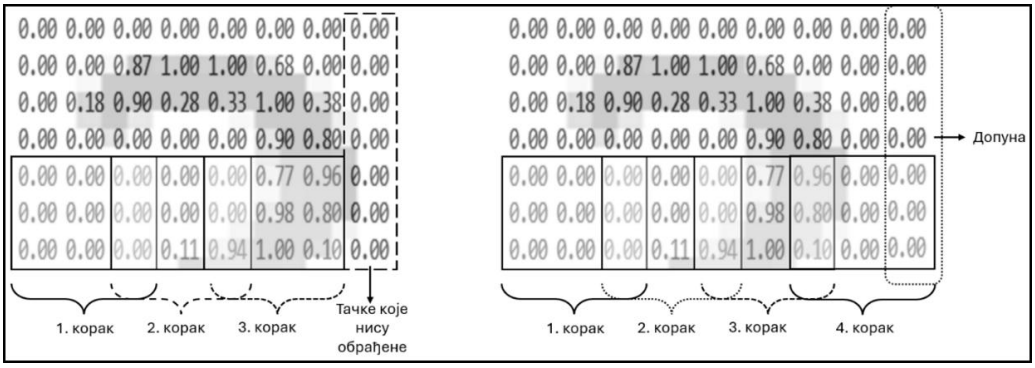


Слика 9.49: принцип рада конволуционог филтера (кернела)

Од слике која представља матрицу од 15x16 = 240 елемената, конволуционим филтером са кораком (*stride*) 2 је добијена резултујућа матрица својстава која има 7x7=49 елемента (конволуциони филтер не може делимично да захвати делове слике, односно у свим његовим пољима морају да буду тачке слике коју обрађује). Да је корак био један, онда би се добила матрица 13x14=182 елемената. То значи да без обзира на величину корака, свака конволуција је мања од слике. За велике слике уобичајено је да се користи више конволуционих филтрирања, тако да је последица описаног ефекта – смањивање (енгл. *shrinking*) сваке следеће конволуције у односу на претходну. Пошто се смањивање врши на рачун (десних и доњих) углова и ивица који су обрађени у мањем броју корака конволуционог филтера него остали делови слике/конволуције, постоји потенцијална могућност да дође до губљења корисне информације коју слика садржи.

Описан проблем је решен увођењем концепта *допуњавања* (енгл. *padding*). Допуњавањем се уводи маргина, тако што се (по потреби) слика модификује додавањем тачака како би конволуциони филтер (кернел) подједнако захватио све њене тачке, не рачунајући маргину. На примеру (Слика 9.50) је слика 7x8 тачака, док је кернел димензије 3x3 поља. Кернел обрађује слику кораком 2 што значи да ће бити у сваком реду обраде испуштена последња колона тачака. Проблем је још већи пошто и вредности у 2 претпоследње колоне губе на значају у својствима конволуције. То се дешава јер се обрађују само у једном (трећем) кораку, а као што се на слици види, имају велики значај. Да се то не би догодило, извршена је допуна једном колоном тачака неутралних вредности (0). Тиме је постигнуто да се слика обрађује у

више корака, наведене вредности се обрађују у два уместо у једном кораку, чиме конволуција није изгубила на репрезентативности својстава.



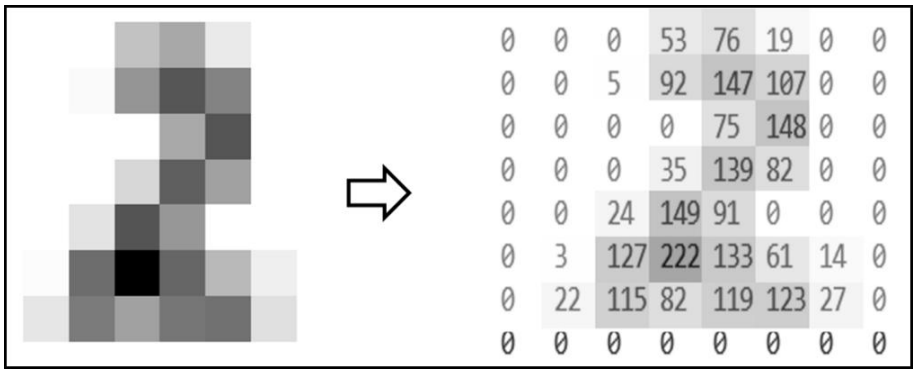
Слика 9.50: Дοпуњавање слике (енīл. padding*) пре обраде конволуционим филтером*

Као што је претходно наведено, допуњавањем се спречава и смањивање конволуције. Димензија конволуције се израчунава изразом (Израз 9.13), у коме је Dim_i^d димензија d конволуције i, Dim_u^d је димензија d слике, или претходне конволуције u, P представља величину додавања (*padding*) и S представља корак кернела (*stride*). Заградом је представљена функција заокруживања количника увек на мањи број.

$$Dim_i^d = \left\lfloor \frac{Dim_u^d - Dim_f^d + 2P}{S} \right\rfloor + 1 \qquad \text{И.9.13}$$

На пример, ако је слика била 15x16 тачака ($Dim_u^{visina} = 15$, $Dim_u^{sirina} = 16$), кернел 3x3 ($Dim_f^{visina} = 3$), корак 2 ($S = 2$), без додавања ($P = 0$), димензије резултујуће конволуције су исте ($Dim_i^{visina} = Dim_i^{sirina} = 7$).

Визуелизација резултата (Слика 9.51) показује да је, поред редукције димензија формата слике и количине података која садржи, конволуцијом добијена генерализована репрезентација броја два.



Слика 9.51: Резултат конволуције слике броја два кораком 2 и резултујућом матрицом 8x8 тачака

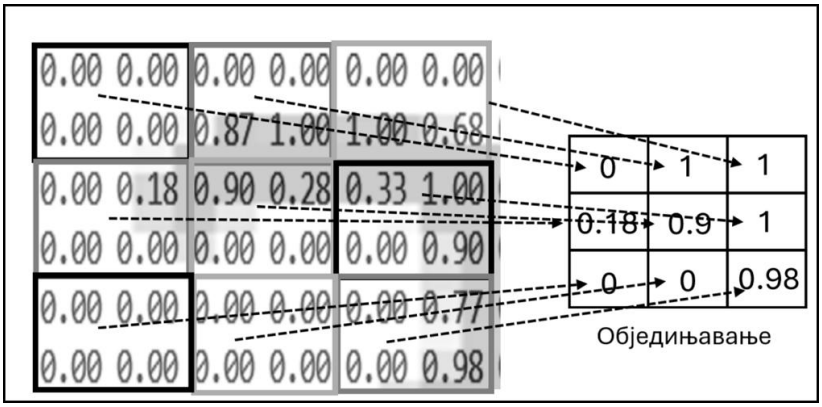
Овако добијена репрезентација омогућава да се на основу распореда и пропорција вредности елемената изврши аутоматско препознавање коришћењем обучене неуронске мреже у улози класификатора.

9.5.3. Принцип обједињавања

Поред конволуције, карактеристика конволуционих неуронских мрежа је и процес обједињавања (енгл. *pooling*). Обједињавање се врши искључиво на матрицама својстава добијеним на излазу конволуционих слојева са циљем редукције просторних димензија. Као што је у претходној секцији објашњено, није пожељно да операција конволуције врши редукцију димензија због опасности од губљења важних својстава која могу да буду одлучујућа у процесима класификације. Обједињавање као процес се врши независно, по каналима. У пракси се у конволуционим операцијама користи више конволуционих филтера (кернела) и сваки кернел генерише резултујућу матрицу својстава, која представља посебан канал. При обједињавању, сваки канал се засебно обрађује, тако да је број канала након обраде исти. Обједињавање може да буде максималном, или просечном вредношћу. За обједињавање се дефинише прозор

На следећем примеру је представљено обједињавање максимумом (Слика 9.52). Матрица својстава има димензије 6x6 поља. Димензија прозора за обједињавање је 2x2 поља. Лево је матрица својстава, добијена на излазу из конволуционог слоја, док је на десној страни добијена мапа обједињавања. Редукција димензија је очигледна и сведена је на матрицу својстава од 3x3 поља. За разлику од конволуције, код кога кернел ради у корацима тако да има преклапања између захваћених поља, код

обједињавања корак одговара димензијама прозора, тако да преклапања поља приликом обраде нема.



Слика 9.52: ӣринцийӣ обједињавања (pooling) максимумом

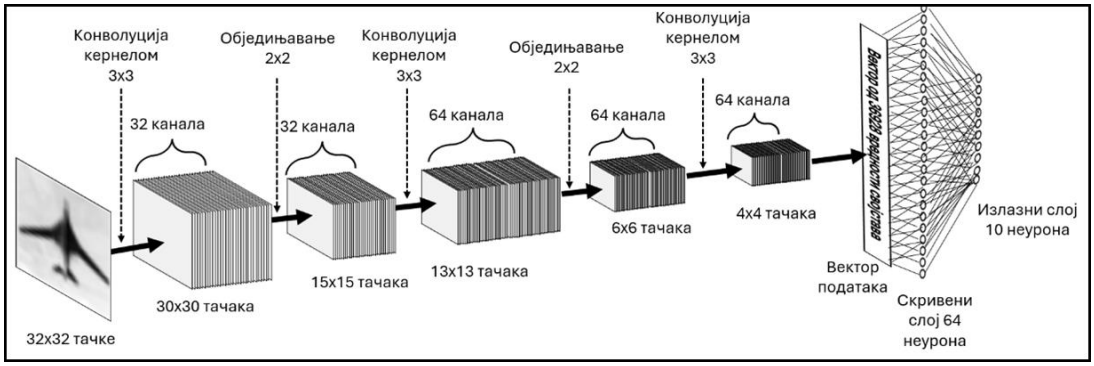
У резултујућој матрици се издвајају најзначајнија својства, било да се ради о просечној, или максималној вредности поља у прозору обједињавања.

9.5.4. Архитектура конволуционих неуронских мрежа

Велике слике (на пример, добијене коришћењем паметних телефона или камера) захтевају да се у пракси обично користи више различитих конволуционих филтера како би се издвојила својства која ће бити одређујућа у даљој обради слике. Ова чињеница одређује да конволуционе неуронске мреже садрже један, или више конволуционих слојева. Број слојева је најчешће адаптибилан, како би се избегло беспотребно процесирање малих слика. На пример, за слике које су реда 10^1 – 10^2 тачака је у већини случајева довољна једна конволуција (једно конволуционо филтрирање). За слике реда 10^6 – 10^9 тачака најчешће треба више од две конволуције.

На следећој илустрацији приказан је пример архитектуре конволуционе неуронске мреже, који садржи 3 конволуциона слоја између којих су два слоја обједињавања (Слика 9.53). Наизменичним операцијама конволуционих филтера и обједињавања, од слике 32x32 тачке дошло се до 64 канала – резултујућих (4x4) матрица својстава. Ова матрица се затим трансформише у вектор ради даљег коришћења у класификатору. У примеру је као класификатор употребљен вишеслојни перцептрон (секција 9.4.3), који се састоји од улазног слоја (који трансформише улазне податке у вектор, енгл. *flattening*), једног скривеног слоја и од излазног слоја. Број

неурона скривеног слоја (64) је усклађен са бројем канала на излазу из последњег конволуционог слоја. Излазни слој садржи 10 неурона. Тај број је усклађен са бројем класа у које подаци (слике) треба да се класификују.



Слика 9.53: пример архитектуре конволуционе неуронске мреже

9.5. Пример имплементације конволуционе неуронске мреже у *Python*-у

У следећем примеру представљена је имплементација класификатора слика на бази вишеслојне конволуционе неуронске мреже. Као скуп података коришћен је јавно доступан *Cifar10* који садржи 60000 слика у боји (3 канала), величине 32x32 тачке, при чему су слике сврстане у 10 категорија: *авион, аутомобил, птица, мачка, јелен, пас, жаба, коњ, брод* и *камион*.

За имплементацију конволуционе мреже као модел је коришћен објект класе *Sequential* из модула *keras.models*. Због комплексније структуре, модел конволуционе мреже биће објашњен у две целине. У првом делу модела (Слика 9.54) се креирају конволуциони и слојеви обједињавања. Пошто модел класификује слике, коришћен је слој *Conv2D* из библиотеке *keras.layers*. *Conv2D* слој је специјализован за обраду података са 2 димензије. *Conv2D* слој користи више конволуционих филтера (кернела). Први параметар је број филтера (32) и толико ће бити канала на излазу (секција 9.5.2, израз 9.13). Други параметар су димензије конволуционог филтера (3x3). *ReLU* (секција 9.2.) се користи као активациона функција. Параметар *input_shape* описује формат слике (32x32 тачке и 3 *RGB* канала). На излазу из првог конволуционог слоја формат конволуције биће 30x30

Као други слој који се додаје у модел је слој обједињавања (*layers.MaxPooling2D*) који врши под-узорковање смањујући димензије излазне мапе својстава. У примеру има само један параметар – величину кернела за обједињавање (2x2), чиме уједно дефинише величину корака (*2*). На излазу из слоја за обједињавање излазна мапе својстава биће дупло мања (15x15) од конволуције на улазу.

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

Слика 9.54: додавање конволуционих слојева модела класификатора

Позивом методе *summary* на моделу добија се извештај о структури слојева модела и параметрима тих слојева (Слика 9.55). Излазни формат конволуционих слојева зависи од формата податка на улазу и димензије конволуционих филтера. На основу израза (секција 9.5.2, израз 9.13), добија се величина излазног формата од 30x30 и број од 32 канала на излазу, који одговара броју конволуционих филтера. Други слој је обједињавајући. Обзиром да ради са кораком 2, излазни формат је дупло мањи од претходног (15x15) при чему се број канала на излазу не мења. Истим методама могу да се објасне излазни формат и број канала другог конволуционог слоја, након ког следи још један обједињавајући, са истим кораком (2). Слојеви обједињавања немају параметре. Број параметара конволуционог слоја N_{params} може се израчунати (израз 9.14) као производ димензија кернела (K_{d1}, K_{d2}) и броја улазних канала (N_{uk}) увећаних за један, који се затим множе са бројем излазних канала (N_{ik}).

$$N_{params} = (K_{d1} * K_{d2} * N_{uk} + 1) * N_{ik} \qquad \text{И..9.14}$$

У конкретном случају кернел је 3x3, број улазних канала је 3 (*RGB* канали слике која се обрађује), што даје 27, увећано за 1 је 28, што множи број излазних канала (32), што резултира са укупно 896 параметара. Број параметара излазног формата другог и трећег конволуционог слоја се знатно увећава иако је кернел истог формата. Експанзија величена се дешава због броја улазних канала у та два слоја (64).

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36,928

Total params: 56,320 (220.00 KB)

Слика 9.55: преглед основних података конволуционих слојева модела класификатора

Други део модела (Слика 9.56) намењен је за класификацију на основу резултујућих матрица својстава, добијених на излазу комбинације конволуционих слојева и слојева обједињавања. Излазни формат последњег конволуционог слоја (*conv2d_2*) у виду 64 канала – резултујућих матрица формата 4x4. Вишеслојни перцептрон очекује на улазу векторску структуру (1Д формат) тако да је из библиотеке *keras.layers* искоришћен је *Flatten* објекат за форматирање различитих формата улазних података у вектор. Након тог улазног слоја додати су као *Dense* објекти један скривени и излазни слој. Скривени слој садржи 64 неурона (аналогно броју канала на излазу из конволуционог блока) и *ReLU* активационом функцијом (секција 9.2). Излазни слој класификатора садржи 10 неурона – за сваку класу слика по један. Активациона функција излазног слоја је *softmax* (секција 9.4.3.2), која трансформише вектор резултата у расподелу вероватноћа тих резултата.

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

Слика 9.56: слојеви модела намењени класификацији

Након конструкције модела следи конфигурација и припрема за његово обучавање (Слика 9.57). Као оптимизатор је искоришћен *adam* (секција 9.4.1.1.2), а као функција губитка искоришћена је *SparseCategoricalCrossentropy* , која одговора када се врши класификација у више од 2 класе и у случају да је *softmax* активациона функција излазног слоја. Параметар *from_logits* има вредности *False* чиме се указује да је на вредности излазног слоја већ примењена дистрибуција вероватноћа.

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

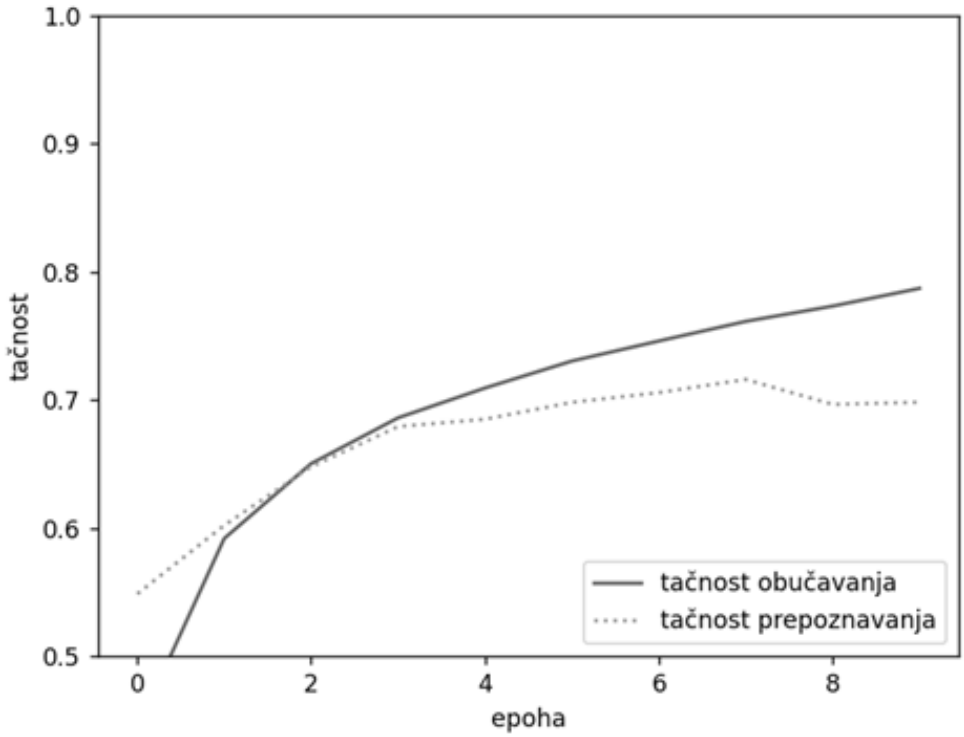
Слика 9.57: конфиīурација модела за йроцес обучавања

За обучавање модела користи се *fit* функција (Слика 9.58) којој се прослеђују слике и ознаке класа слика у скупу за обучавање (50000 слика у конкретном случају). Обучавање се врши у 10 епоха, а за валидацију модела у току обучавања прослеђује се и посебан скуп слика (10000 слика у конкретном случају) за проверу са ознакама њихових класа, тако да се добијају историјски подаци обучавања за сваку епоху у смислу тачности (*accuracy*) у односу на податке за обучавање, као и тачности (*val_accuracy*) у односу на непознате податке (*slike_za_proveru*).

```
istorija = model.fit(slike_za_obucavanje, klase_slika_za_obucavanje, epochs=10,
          validation_data=(slike_za_proveru, klase_slika_za_proveru))
```

Слика 9.58: Обучавање модела

На дијаграму историје обучавања (Слика 9.5) се може уочити да је тачности у односу на податке за обучавање знатно већа, што је очекивано. Добијена је резултујућа тачности од ~0.8 за коју се може рећи да је *добра* обзиром да је учење обављено само у 10 епоха. Као резултат процеса обучавања добијена је и вредност функција губитка од ~0.55, што указује на задовољавајућу ефикасност обучавања и да су предикције поуздане.



Слика 9.59: приказ историје обучавања модела конволуционе неуронске мреже

Након обучавања и провере (ако су резултати задовољавајући), модел може да се користи. За ту сврху користи се функција *predict*, којој се прослеђује слика која треба да се класификује. Могућа је класификација слика различитих формата, али је неопходно претходно да се трансформише у формат који се очекује на улазу у конволуциону мрежу (32x32 тачке са 3 (*RGB*) канала).

9.5. Закључак

У поглављу о вештачким неуронским мрежама су представљени и објашњени принципи и методе функционисања и обучавања различитих врста неуронских мрежа. Поред тога, представљене су примене вештачких неуронских мрежа у решавању различитих проблема. Вештачке неуронске мреже са храњењем у напред најчешће се користе за решавање проблема класификације, док су рекурентне мреже погодније за анализе података временских серија и предикције будућих трендова. Конволуционе мреже омогућавају обраду комплексних мултимедијалних података захваљујући техникама конволуције и обједињавања података, којима се из изворних података издвајају значајна својства, елиминише информациони шум и

изворни подаци се трансформишу у каналске структуре матрице својстава, које се даље прослеђују вишеслојном перцептрону ради класификације. Захваљујући слојевитој структури мреже, омогућено је *дубоко учење* на основу велике количине података, што је слично функционисању људског мозга. Обзиром на комплексност архитектуре и велики број параметара којима се обрађују подаци, хардверски ресурси представљају критичну инфраструктуру у имплементацији модела вештачких неуронских мрежа.

9.6. Питања

1. Које су активационе функције неурона?
2. Како функционише једнослојни перцептрон?
3. Која се активациона функција користи код неуронске мреже на бази радијалних функција и зашто?
4. Како се обучава вишеслојни перцептрон?
5. Објаснити функцију губитка код вишеслојног перцептрона.
6. У ком случају се користе рекурентне неуронске мреже?
7. Које активационе функције се користе у LSTM мрежама и која има је намена?
8. Које су сличности и разлике између LSTM и GRU мрежа?
9. Која је улога кернела у конволуционим неуронским мрежама?
10. Која је улога обједињавања (*pooling*) у конволуционим неуронским мрежама?
11. Шта је неопходно урадити приликом повезивања конволуционог слоја са вишеслојним перцептроном?

10. Речник кључних речи

Кључна реч	Кратко објашњење
Активациона функција	Функција која даје излазну вредност из неурона
Апроксимација	Приближавање тачној вредности
Вектор подршке	Податак који одређује маргину код класификације векторима подршке
Вишеслојни перцептрон	Вишеслојна неуронска мрежа са храњењем у напред
Дефазификација	Поступак којим се од резултујућег скупа расплинутог резоновања добија дискретна вредност
Евалуација модела	Провера обучености модела након обучавања
Фазификација	Поступак којим се дискретна вредност трансформише у расплинут скуп
Фреквенција термина	Функција која мери број појављивања одређене речи у документу
Функција губитка	Функција која описује степен непрецизности модела у класификацији
Функција припадања	Функција која описује степен припадања вредности расплинутом скупу
Хармонични опадајући модел	Модел нелинеарне регресије
Хеуристика	Скуп метода које омогућавају откривање знања и решавање проблема на бази претходног знања
Хиперболични тангенс	Активациона функција неурона заснована на тангенсу
Храњење у напред	Процес кретања података кроз вишеслојни перцептрон
Инверзна фреквенција документа	Логаритамска функција која мери важност документа у односу на критеријум претраге
Интелигенција	Ментална особина која омогућава учење из искуства, адаптирање на нове ситуације, схватање и разумевање нових ситуација и коришћења стеченог знања у решавању проблема
Једнослојни перцептрон	Неуронска мрежа од једног неурона за једноставне бинарне класификације

Капија заборава	Део *ЛСТМ* ћелије који искључује екстремне улазне вредности у процесу обучавања
Квантификатор	Омогућава представљање знања у предикатској логици
Кернел	Математичка операција за издвајање својстава из података у конволуционим мрежама (1) и за класификацију векторима подршке (2)
Кластер	Неименовани скуп података сличних на основу вредности својстава које садрже
Коефицијент детерминације	Функција која одређује да ли полиномска регресија одговара за моделовање података
Конволуција	Процес издвајање својстава из података помоћу кернела
Консеквенца	Акциони (*онда*) део правила
К-најближих суседа	Поступак класификације података на бази К-суседних података
К-средина	Поступак груписања података у К-кластера
Косинусна сличност	Функција која мери сличност два скаларна вектора базирана на косинусу
Логистичка регресија	Класификатор који користи логистичку сигмоидну функцију
ЛСТМ ћелије	Неурони рекурентних мрежа који садрже дугорочну и краткорочну меморију
Маргина	Одређује границе у процесу класификације
Модел	Алгоритам и пратећа структура у машинском учењу
Модел засићења	Модел нелинеарне регресије
Модус поненс	Начин афирмативног закључивања у логици
Модус толенс	Начин закључивања у логици одбацивањем
Надзирано обучавање	Обучавање скупом података који су већ класификовани
Наивни Бајес	Класификација на бази Бајесове теореме и једном евиденцијом догађаја
Не надзирано обучавање	Обучавање скупом података непознате природе и дистрибуције
Насумична стабла	Класификатор који садржи више стабала одлучивања
Обучавање модела	Адаптација модела коришћењем скупа података

Одмотавање	Анализа обучавања рекурентне мреже кроз временске секвенце
Пирсонов коефицијент	Одређује да ли линеарна регресија одговара за моделовање података
Померајућа средина	Метода кластеровања секвенцијалним избором података за центроиде кластера
Поклапање узорака	Метода за утврђивање образаца у подацима
Правило	Начин представљања знања у *ако – онда* облику
Премиса	Условни (*ако*) део правила
Расплинут скуп	Скуп коме подаци могу да припадну у одређеној мери
Расплинуто кластеровање	Податак може да припада у више кластера у одређеном степену
Регресивна линија	Замишљена линија која описује дистрибуцију података у узорку
Рекурентна јединица	Неурон у слоју рекурентне неуронске мреже
Резоновање	Мислити, размишљати, расуђивати и закључивати разумно
Синапса	Веза између неурона у суседним слојевима вештачке неуронске мрежа
Синаптичка тежина	Параметар за обучавање у вештачким неуронским мрежама
Степеновани модел	Модел нелинеарне регресије
Стратификација	Припрема података за обучавање модела класификатора вођењем рачуна о равномерној заступљености података по класама
Тачност	Функција која описује прецизност и поузданост модела у класификацији
Удруживање	Редуковање димензије марице својстава сабирањем суседних поља код конволуционих мрежа

11. Библиографија

[1] Stuart Russell and Peter Norvig. 2009. Artificial Intelligence: A Modern Approach (3rd. ed.). Prentice Hall Press, USA.

[2] Géron Aurélien. 2023. Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow. Sebastapol, CA: O'Reilly Media, Inc.

[3] Enric Trillas and Luka Eciolaza. 2016. Fuzzy Logic: An Introductory Course for Engineering Students (1st. ed.). Springer Publishing Company, Incorporated.

[4] Oliver Kramer. 2017. Genetic Algorithm Essentials (1st. ed.). Springer Publishing Company, Incorporated.

[5] Timo Koski and John Noble. 2009. Bayesian Networks: An Introduction (1st. ed.). Wiley Publishing.

[6] Jim Frost. (2019). Regression Analysis: An Intuitive Guide for Using and Interpreting Linear Models. Jim Publishing.

[7] Charu C. Aggarwal and Chandan K. Reddy. 2013. Data Clustering: Algorithms and Applications (1st. ed.). Chapman & Hall/CRC.

[8] Charu C. Aggarwal. 2014. Data Classification: Algorithms and Applications (1st. ed.). Chapman & Hall/CRC.

[9] Charu C. Aggarwal. 2018. Neural Networks and Deep Learning: A Textbook (1st. ed.). Springer Publishing Company, Incorporated.

[10] Salman Khan, Hossein Rahmani, and Syed Afaq Ali Shah. 2018. A Guide to Convolutional Neural Networks for Computer Vision. Morgan & Claypool Publishers.

[11] Fathi M. Salem. 2022. Recurrent Neural Networks: From Simple to Gated Architectures, Springer Publishing Company, Incorporated.

[12] Ramčilović Jesih, A., Šimić, G., Konatar, L. Brljak, Z., Šprajc, P. (2024). Energy efficiency as a driver of the circular economy and carbon neutrality in selected countries of Southern Europe: a soft computing approach. Energy, Sustainability and Society 14, 22, https://doi.org/10.1186/s13705-024-00456-1

[13] Knez, S., Šimić, G., Milovanović, A, Starikova, S., Županić, F.Ž. (2022) Prices of conventional and renewable energy as determinants of sustainable and secure energy development: regression model analysis. Energy, Sustainability and Society 12, https://doi.org/10.1186/s13705-022-00333-9

[14] Šimić G, Radovanović M, Filipović S, Mirković Isaeva O. (2021) Fuzzy logic approach in energy security quantification - "ESecFuzzy" software application, Soft Computing 25: 1-16, https://doi.org/10.1007/s00500-021-05976-y

[15] Hribar, N., Šimić, G., Vukadinović, S, Šprajc, P. (2021) Decision-making in sustainable energy transition in Southeastern Europe: probabilistic network-based model. Energy, Sustainability and Society 11, 39, https://doi.org/10.1186/s13705-021-00315-3

[16] Podbregar I, Šimić G, Radovanović M, Filipović S, Maletič D, Šprajc P. (2020) The International Energy Security Risk Index in Sustainable Energy and Economy Transition Decision Making — A Reliability Analysis, Energies, Vol. 13, Issue 14, 3691. https://doi.org/10.3390/en13143691

[17] Podbregar I, Šimić G, Radovanović M, Filipović S, Šprajc P. (2020) International Energy Security Risk Index - Analysis of the Methodological Settings, Energies, Vol. 13, Issue 12, 3234, https://doi.org/10.3390/en13123234

[18] Jovanović M, Šimić G, Čabarkapa M, Ranđelović D, Nikolić V, Nedeljković S, Čisar P. (2019) SEFRA - Web-based Framework Customizable for Serbian Language Search Applications, Acta Polytechnica Hungarica, Volume 16, Issue Number 3, http://www.uni-obuda.hu/journal/Jovanovic_Simic_Cabarkapa_Randelovic_Nikolic_Nedeljkovic_Cisar_90.pdf

[19] Goran P. Šimić, Unapređenje servisa e-vlade za naprednu pretragu, Vojnotehnički glasnik, vol. 67 br. 2 (2019), https://doi.org/10.5937/vojtehg67-20356

[20] Šimić, G., Jevremović, A., Strugarević, D. (2024). Improvement of the Teaching Process Using the Genetic Algorithm. In: Perakovic, D., Knapcikova, L. (eds) Future Access Enablers for Ubiquitous and Intelligent Infrastructures. FABULOUS 2024. Lecture Notes of the Institute for

Computer Sciences, Social Informatics and Telecommunications Engineering, vol 596. Springer, Cham. https://doi.org/10.1007/978-3-031-72393-3_7

12. Индекс појмова и имена

ЗАВРШНЕ ОДРЕДБЕ: