Mašinsko učenje

Programski jezik Python i mašinsko učenje

Nemanja Maček

Sadržaj

- Uvodne napomene
- NumPy
- SciPy
- Prva (jednostavna aplikacija)
- Zaključne napomene

Uvodne napomene

Osnovni koraci.

- Čitanje podataka i čišćenje,
- analiza i razumevanje ulaznih podataka,
- analiziranje kako najbolje predstaviti podatke algoritmu učenja,
- izbor pravog modela i algoritma učenja, i
- pravilno merenje performansi.

Uvodne napomene

Šta nam je potrebno?

- Python,
- biblioteke za matematičke operacije i vizualizaciju.

Kako se (ne)pravilno uvozi NumPy?

```
>>> import numpy
>>> numpy.version.full_version
1.8.1
```

- Da li je sledeći način uvoza OK?

```
>>> from numpy import *
```

- Ne, „pregazićete“ standardni array Python paket.

Pravilan način upotrebe.

```
>>> import numpy as np
>>> a = np.array([0,1,2,3,4,5])
>>> a
array([0, 1, 2, 3, 4, 5])
>>> a.ndim
1
>>> a.shape
(6,)
```

- Upravo smo kreirali niz na sličan način kao što bi kreirali listu u Pythonu.
- Međutim NumPy sadrži dodatne informacije o nizu, odnosno matrici.
- U ovom slučaju, radi se o jednodimenzionalnom nizu od 6 elemenata.

Transformacija u dvodimenzionalnu matricu.

```
>>> b = a.reshape((3,2))
>>> b
array([[0, 1],
       [2, 3],
       [4, 5]])
>>> b.ndim
2
>>> b.shape
(3, 2)
```

Šta će se desiti sa originalnim nizom ako izmenimo vrednost kopije?

```
>>> b[1][0] = 77
>>> b
array([[ 0, 1],
       [77, 3],
       [ 4, 5]])
>>> a
array([ 0, 1, 77, 3, 4, 5])
```

- Vrednost elementa originalnog niza se menja ukoliko izmenite vrednost element kopije.
- Drugim rečima, kopija je zavisna od originala i obrnuto.
- Šta trebamo da uradimo ako želimo da se vrednost originalnog niza ne menja prilikom izmene vrednosti elementa kopije, odnosno ukoliko želimo nezavisnu kopiju?

Kreiranje nezavisne kopije matrice.

```
>>> c = a.reshape((3,2)).copy() # korišćenjem funkcije copy() kreira se nezavisna kopija
>>> c
array([[ 0, 1],
       [77, 3],
       [ 4, 5]])
>>> c[0][0] = -99
>>> a
array([ 0, 1, 77, 3, 4, 5])
>>> c
array([[-99, 1],
       [ 77, 3],
       [ 4, 5]])
```

Poređenje nekih operacija nad matricama sa operacijama nad listama.

```
>>> d = np.array([1,2,3,4,5])
>>> d*2 # množenje svakog elementa
array([ 2, 4, 6, 8, 10])
>>> d**2 # stepenovanje svakog elementa
array([ 1, 4, 9, 16, 25])
```

- Ove operacije se razlikuju od operacija nad listama u Pythonu.

```
>>> [1,2,3,4,5]*2
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
>>> [1,2,3,4,5]**2
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

Pristupanje elementima niza.

```
>>> a
array([ 0, 1, 77, 3, 4, 5])
>>> a[np.array([2,3,4])]
array([77, 3, 4])
```

- Propagacija uslovnih operatora elementima niza.

```
>>> a>4 # vraća bulovski niz u kome je vrednost True za sve elemente niza koji su veći od četiri
array([False, False, True, False, False, True], dtype=bool)
>>> a[a>4] # vraća niz koji sadrži sve elemente početnog niza koji su veći od četiri
array([77, 5])
```

Pristupanje elementima niza.

- Izmena vrednosti elemenata koji zadovoljavaju dati uslov.

```
>>> a
array([ 0, 1, 77, 3, 4, 5])
>>> a[a>4] = 4 # vraća niz u kome su vrednosti elemenata većih od 4 postavljene na 4
>>> a
array([0, 1, 4, 3, 4, 4])
```

Rad sa nepostojećim vrednostima.

- Nepostojeće vrednosti se često javljaju prilikom čitanja podataka iz datoteke.

```
>>> c = np.array([1, 2, np.NAN, 3, 4]) # pravićemo se da je ovo pročitano iz tekstualne datoteke
>>> c
array([ 1., 2., nan, 3., 4.])
>>> np.isnan(c)
array([False, False, True, False, False], dtype=bool)
>>> c[~np.isnan(c)]
array([ 1., 2., 3., 4.]) # u ovom slučaju ne prikazuje nepostojeću vrednost
>>> np.mean(c[~np.isnan(c)]) # računa srednju vrednost svih vrednosti sem nepostojeće
2.5
```

Šta ne možemo da radimo?

- NumPy nizovi nisu fleksibilni kao liste, odnosno moraju da sadrže elemente istog tipa.
- Ukoliko pokušamo da kreirako niz elemenata različitog tipa, NumPy će elemente konvertovati na najrazumniji način u isti tip.

```
>>> a = np.array([1,2,3])
>>> a.dtype
dtype('int64')
>>> np.array([1, "stringy"])
array(['1', 'stringy'], dtype='<U7') # NumPy niz stringova
>>> np.array([1, "stringy", set([1,2,3])])
array([1, stringy, {1, 2, 3}], dtype=object) # NumPy niz objekata
```

Čemu služi SciPy?

- SciPy sadrži algoritme za obradu struktura podataka koje generiše NumPy.
- Primeri su algoritmi za rad sa matricama, linearna algebra, optimizacija, klasterovanje, FFT itd.
- Radi lakšeg rada NumPy prostor imena je dostupan preko SciPy.
- Ovo se može proveriti na sledeći način:

```
>>> import scipy, numpy
>>> scipy.version.full_version
0.14.0
>>> scipy.dot is numpy.dot
True
```

Koje toolbox-ove SciPy sadrži?

- Neki od toolbox-ova koje SciPy sadrži su:
  - cluster – hijerarhijsko klasterovanje, k-means,
  - constants – fizičke i matematičke konstante, metode konverzije,
  - fftpack – Furijeova transformacija,
  - io – ulaz-izlaz,
  - interpolate – linearna i kubna interpolacija, itd,
  - optimize – optimizacija,
  - signal – obrada signala,
  - stats – statistički toolbox.
- Ove i druge toolbox-ove ćemo obrađivati u ovom i narednim predavanjima.

Prva (jednostavna) aplikacija

Postavka problema.

- Kao primer uzećemo hipotetički Web start-up, MLaaS, koji prodaje usluge korišćenja algoritama mašinskog učenja preko HTTP protokola.
- Pod pretpostavkom da kompanija vremenom postaje uspešnija, javlja se potreba za „pojačanjem“ infrastrukture kako bi se uspešno opsližili svi Web zahtevi.
  - Ne želimo da rezervišemo previše resursa jer bi to bilo previše skupo.
  - Sa druge strane, izgubićemo novac, ako ne rezervišemo dovoljno resursa kako bi opslužili sve dolazne zahteve.
- Postavlja se sledeće pitanje: kada će trenutne infrastruktura koja može da obradi 100.000 zahteva po satu postati nedovoljna?
- Drugim rečima, želimo unapred da znamo kada je potrebno da se zakupe dodatni serveri u oblaku kako bi mogli da opslužimo sve zahteve a da pri tome ne plaćamo servere koji su ne iskorišćeni.

Prva (jednostavna) aplikacija

Skup podataka.

- Skup podataka je smešten u datoteci `ch01/data/web_traffic.tsv` (tzv. *tab-separated value*).
- Svaka linija u datoteci sadrži redni broj sata u kome je meren broj zahteva i broj zahteva u tom satu.
- Primer nekoliko linija datoteke:

```
1	2272
2	Nan
3	1386
4	1365
5	1488
6	1337
7	1883
8	2283
```

Prva (jednostavna) aplikacija

Uvoz podataka.

- Podaci se uvoze SciPy funkcijom `genfromtxt()`.
- Iz priloženog se vidi da je uvežena matrica dvodimenzionalna i da sadrži 743 redova.

```
>>> import scipy as sp
>>> data = sp.genfromtxt("web_traffic.tsv", delimiter="\t")
>>> print(data[:5])
[[ 1.00000000e+00 2.27200000e+03]
 [ 2.00000000e+00 nan]
 [ 3.00000000e+00 1.38600000e+03]
 [ 4.00000000e+00 1.36500000e+03]
 [ 5.00000000e+00 1.48800000e+03]
>>> print(data.shape)
(743, 2)
```

Prva (jednostavna) aplikacija

Predprocesiranje i čišćenje podataka.

- Za SciPy, dvodimenzionalnu matricu je pogodnije transformisati u dva vektora veličine 743 elementa – vektor *x*, koji sadrži redni broj sata, i vektor *y*, koji sadrži broj zahteva po satu.
- Nakon toga, potrebno je proveriti da li postoje NAN vrednosti u vektoru *y*, i ukoliko postoje, očisititi ih izrazom `~sp.isnan(y)`.

```
x = data[:,0]
y = data[:,1]
>>> sp.sum(sp.isnan(y))
8
>>> x = x[~sp.isnan(y)]
>>> y = y[~sp.isnan(y)]
```

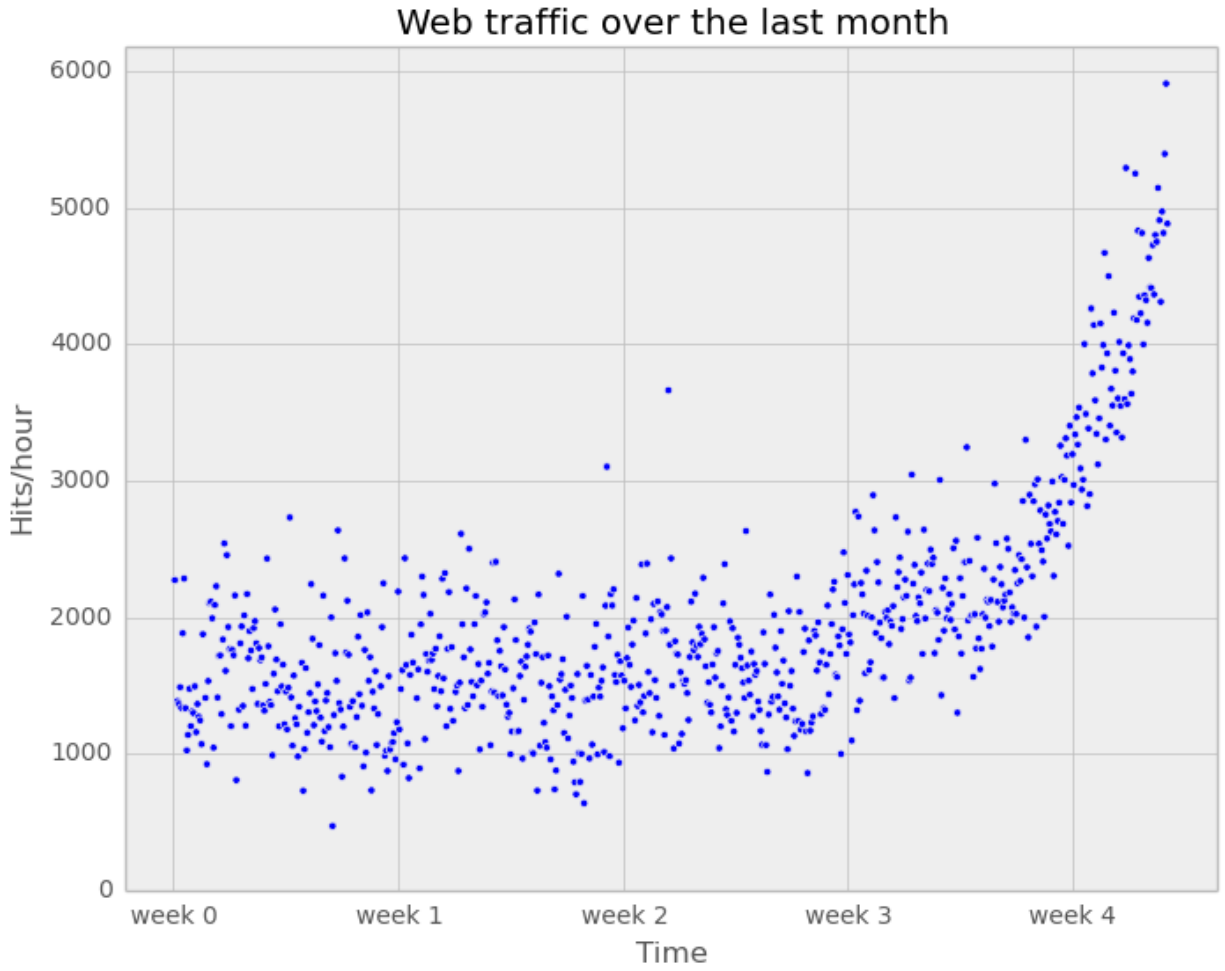
Prva (jednostavna) aplikacija

Vizualizacija podataka po nedeljama.

- Koristićemo matplotlib pyplot paket.

```
>>> import matplotlib.pyplot as plt
>>> plt.scatter(x, y, s=10) # crtaj (x,y) tačke sa veličinom tačke 10
>>> plt.title("Web traffic over the last month")
>>> plt.xlabel("Time")
>>> plt.ylabel("Hits/hour")
>>> plt.xticks([w*7*24 for w in range(10)], ['week %i' % w for w in range(10)])
>>> plt.autoscale(tight=True)
>>> plt.grid(True, linestyle='-', color='0.75') # crtaj prozirnu mrežu
>>> plt.show()
```

Prva (jednostavna) aplikacija



Prva (jednostavna) aplikacija

Odabir modela i algoritma učenja.

- Pre nego što krenemo da obučavamo modele, potrebno je da definišemo grešku aproksimacije.
- Ova greška nam pomaže pri odabiru modela.
- U našem slučaju, definisaćemo je kao sumu kvadrata udaljenosti predviđenih i stvarnih vrednosti.

```
def error(f, x, y):
  return sp.sum((f(x)-y)**2)
```

Prva (jednostavna) aplikacija

Najjednostavniji model – prava linija.

- Kao najjednostavniji model koristićemo pravu liniju.
- U tom slučaju se problem svodi na smeštanje linije u grafikon tako da je greška aproksimacije najmanja.
- To se može učiniti korišćenjem SciPy `polyfit()` funkcije.
- Za date vrednosti *x* i *y* i red polinoma (prava linija ima red *d*=1), pronalazi se model koji ima najmanju grešku aproksimacije:

```
fp1, residuals, rank, sv, rcond = sp.polyfit(x, y, 1, full=True)
```

Prva (jednostavna) aplikacija

Najjednostavniji model – prava linija.

- Funkcija `polyfit()` vraća parametre modela `fp1`.
- Ukoliko postavimo vrednost `full=True`, dobićemo dodatne informacije, poput greške aproksimacije, koja nam je od značaja.

```
>>> print("Model parameters: %s" % fp1)
Model parameters: [ 2.59619213 989.02487106]
>>> print(residuals)
[ 3.17389767e+08]
```

- Ovo znači da je optimalna prava linija zadata na sledeći način:

$$f(x) = 2{,}59619213 \times x + 989{,}02487106$$

Prva (jednostavna) aplikacija

Najjednostavniji model – prava linija.

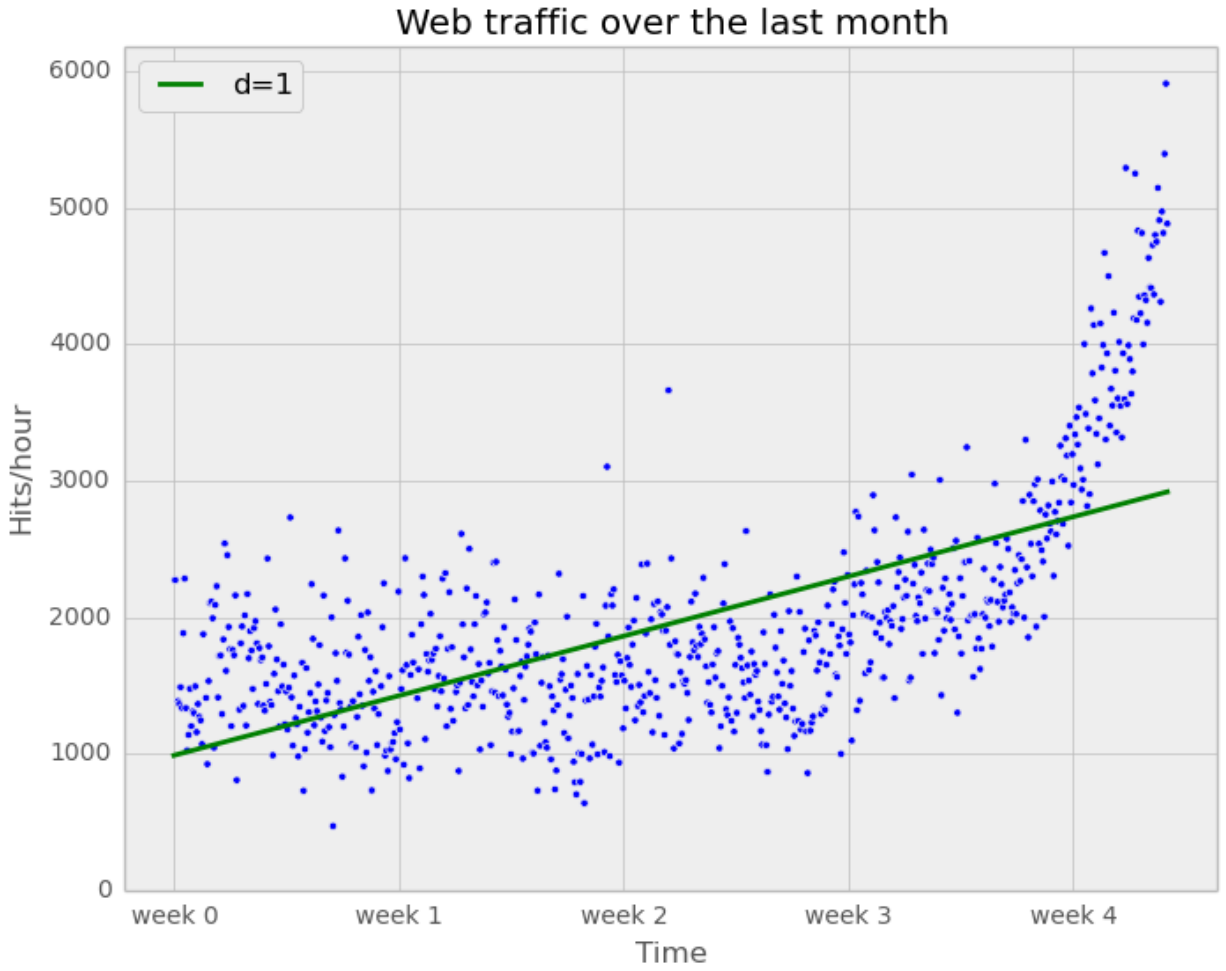
- Koristeći `polyf1d()` kreiraćemo funkciju modela na osnovu parametara.

```
>>> f1 = sp.poly1d(fp1) # prosleđuju se parametri fp1 koji su izlaz funkcije polyfit()
>>> print(error(f1, x, y))
317389767.34
```

- Sada možemo da koristimo funkciju `f1()` za iscrtavanje prvog obučenog modela.
- Uz prethodne instrukcije za iscrtavanje (v. str. 21), jednostavno dodajemo sledeći kod:

```
fx = sp.linspace(0,x[-1], 1000) # generiše X-vrednosti za crtanje
plt.plot(fx, f1(fx), linewidth=4)
plt.legend(["d=%i" % f1.order], loc="upper left")
```

Prva (jednostavna) aplikacija



Prva (jednostavna) aplikacija

Najjednostavniji model – prava linija.

- Šta možemo da zaključimo na osnovu grafikona?
  - Prve četiri nedelje su dobro aproksimirane.
  - Velika greška nastaje nakon četvrte nedelje.
  - Šta je razlog pojave velike greške?

Prva (jednostavna) aplikacija

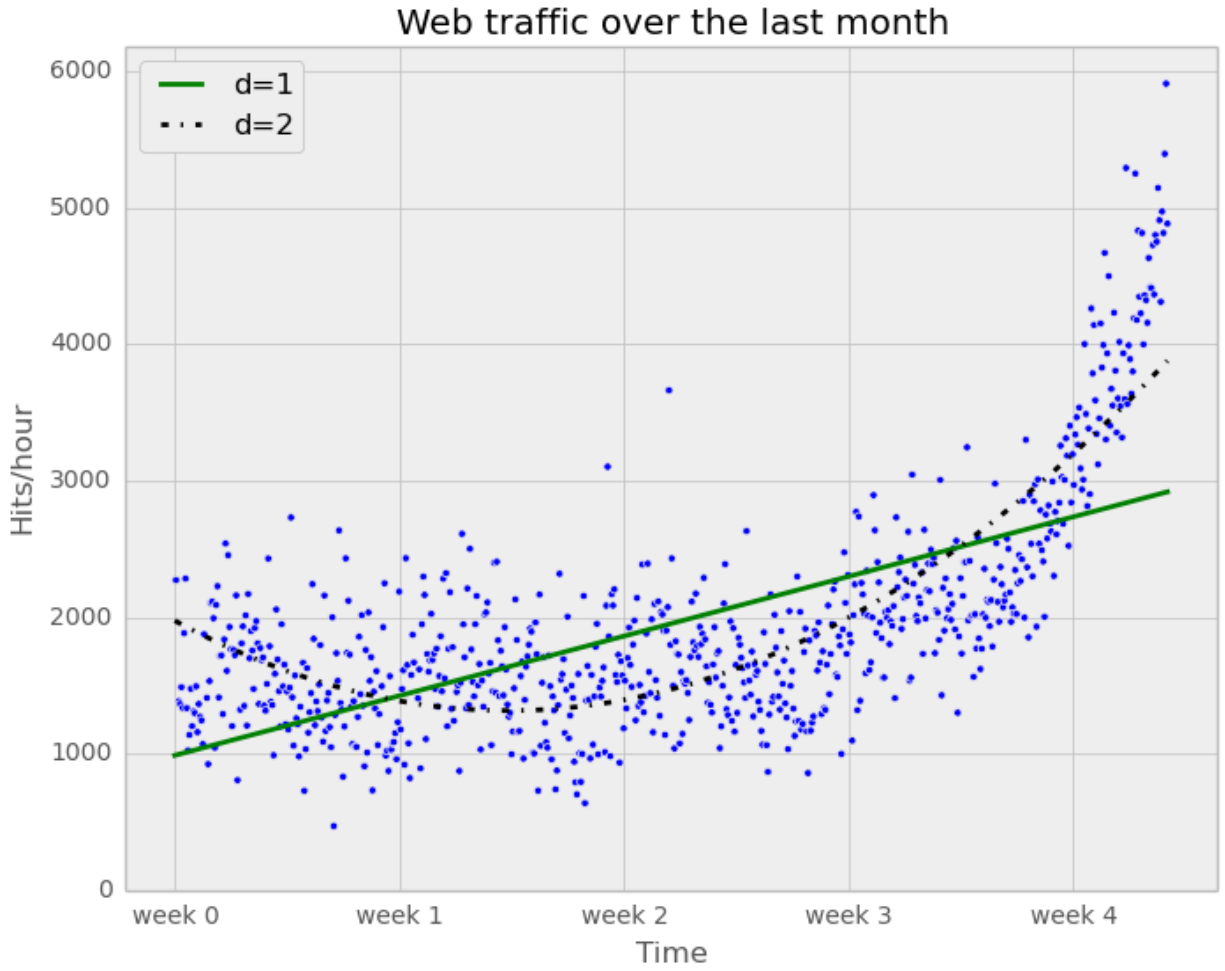
Složeniji model – polinom stepena dva.

- Postavićemo vrednost polinoma na dva u SciPy `polyfit()` funkciji kako bi proverili da li će kriva linija bolje aproksimirati podatke.

```
>>> f2p = sp.polyfit(x, y, 2)
>>> print(f2p)
array([ 1.05322215e-02, -5.26545650e+00, 1.97476082e+03])
>>> f2 = sp.poly1d(f2p)
>>> print(error(f2, x, y))
179983507.878
```

- Dobijamo funkciju: $f(x) = 0{,}0105322215 \times x^2 - 5{,}26545650 \times x + 1974{,}76082$.
- Kakav grafik možemo da očekujemo u ovom slučaju?

Prva (jednostavna) aplikacija

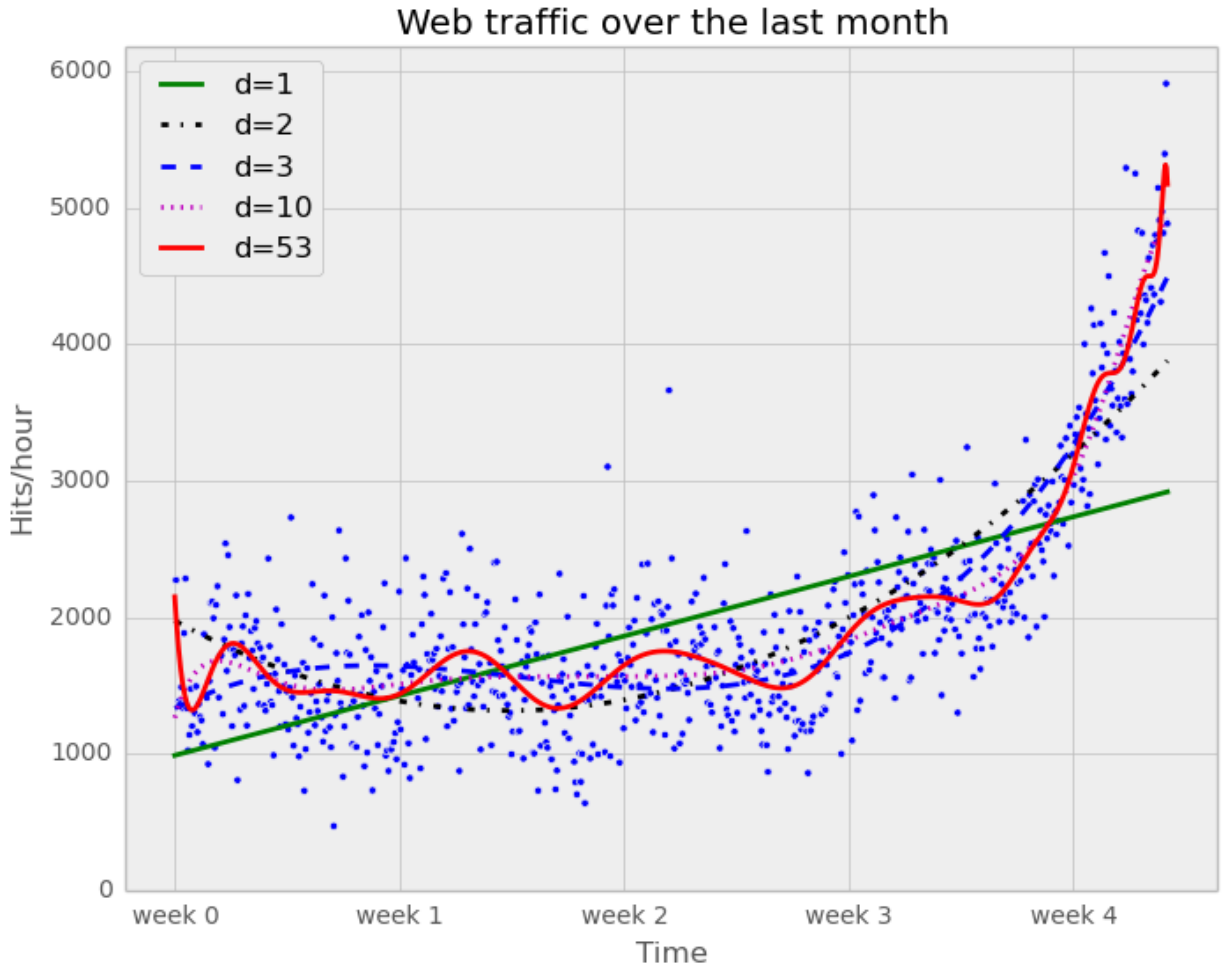


Prva (jednostavna) aplikacija

Složeniji model – polinomi većeg stepena.

- Šta zaključujemo?
  - Povećanje složenosti modela daje bolje rezultate.
- Logično, pokušaćemo da povećamo stepen na npr. 3, 10 i 100.
- Šta možemo da očekujemo?

Prva (jednostavna) aplikacija



Prva (jednostavna) aplikacija

Složeniji model – polinom većeg stepena.

- Sa prethodne slike se vidi da je sistem zaustavio povećanje stepena za *d*=53.
- Šta se zapravo desilo?
  - Zbog numeričkih greška, `polyfit()` ne može da izvrši dobro prilagođenje podacima sa stepeom 100.
  - Dodatno, funkcija je odredila da kriva definisana polinomom stepena 53 dobro aproksimira ulazne podatke.

Prva (jednostavna) aplikacija

Složeniji model – polinom većeg stepena.

- Na osnovu grešaka takođe možemo da primetimo da kriva većeg stepena (složeniji model) bolje aproksimira podatke, odnosno da se bolje prilagođava podacima.

```
Error d=1: 317,389,767.339778
Error d=2: 179,983,507.878179
Error d=3: 139,350,144.031725
Error d=10: 121,942,326.363461
Error d=53: 109,318,004.475556
```

- Međutim, uočite na grafiku krivu definisanu polinomom reda 53.
- Isuviše složen model se preterano prilagođava ulazim podacima, što znači da se, osim procesu koji je opisan ulaznim podacima, prilagođava i šumu.
- Ovaj efekat se zove prenaučenost (engl. *overfitting*).

Prva (jednostavna) aplikacija

Kako možemo postupiti dalje?

- U ovom trenutku imamo nekoliko mogućnosti za nastavak priče:
  - (1) da odaberemo jedan od dovoljno prilagođenih polinomijalnih modela,
  - (2) da odaberemo klasu složenijih vrsta krivih linija,
  - (3) da ponovo analiziramo podatke i krenemo ispočetka.
- Od obučenih modela prvi je očigledno previše jednostavan, dok su polinomijalni modeli reda 10 i 53 očigledno prenaučeni. Modeli reda 2 i 3 se takođe nedovoljno prilagođavaju podacima. Dakle, prvo rešenje otpada.
- Drugo rešenja takođe otpada zato što nemamo argumente koji bi podržali odabir određene složenije klase krivih linija.
- U ovom trenutku možemo da izvedemo zaključak da nismo dobro razumeli prirodu naših podataka.

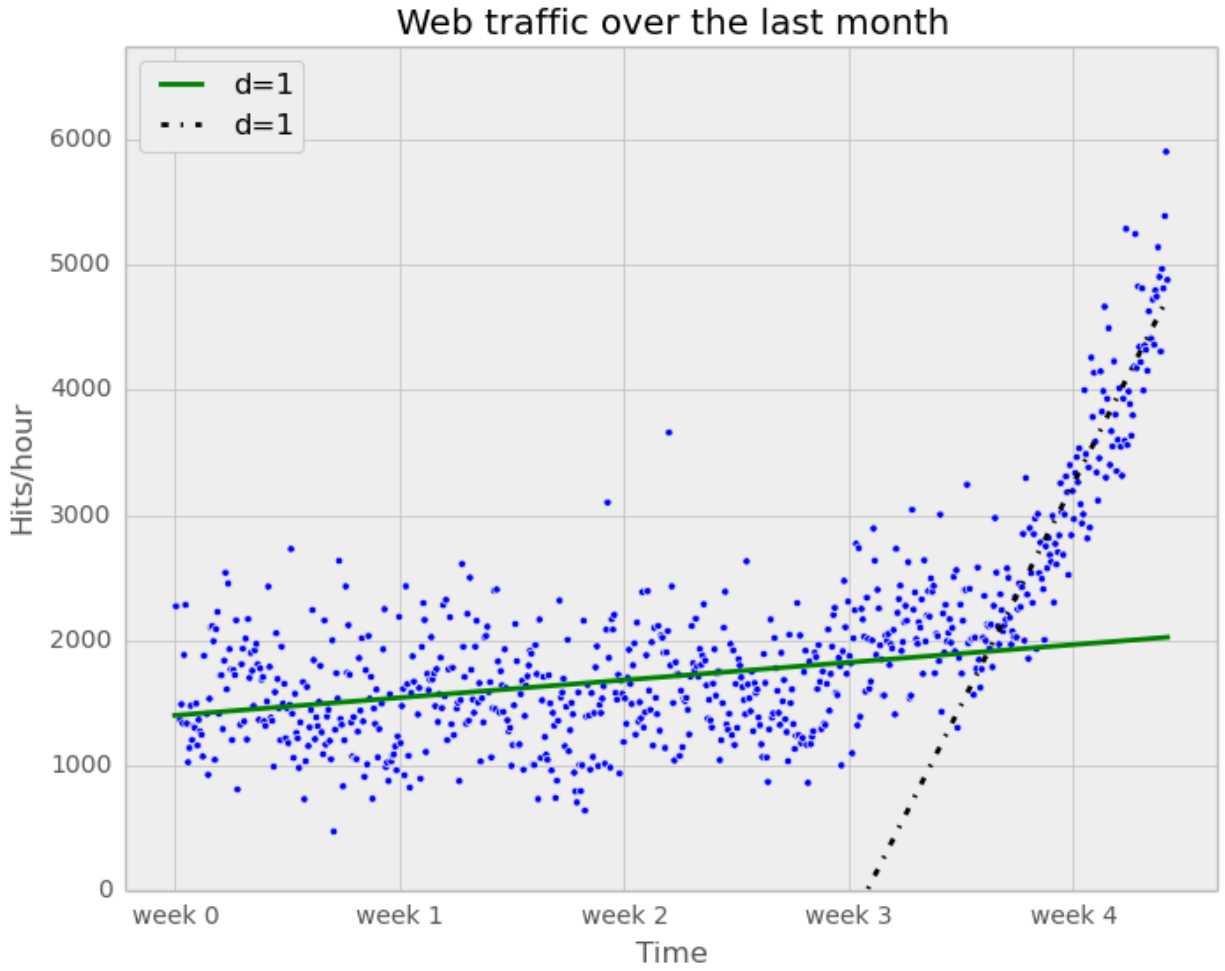
Prva (jednostavna) aplikacija

Korak unazad: ponovna analiza podataka.

- Na grafiku se može uočiti tačka preloma između nedelja 3 i 4.
- Podelićemo podatke i „obučiti“ dve linije koristeći dan nedelju 3,5 kao tačku podele.

```
inflection = 3.5*7*24 # računanje tačke preloma u satima
xa = x[:inflection] # podaci pre tačke preloma
ya = y[:inflection]
xb = x[inflection:] # podaci nakon tačke preloma
yb = y[inflection:]
fa = sp.poly1d(sp.polyfit(xa, ya, 1))
fb = sp.poly1d(sp.polyfit(xb, yb, 1))
fa_error = error(fa, xa, ya)
fb_error = error(fb, xb, yb)
print("Error inflection=%f" % (fa_error + fb_error))
Error inflection=132950348.197616
```

Prva (jednostavna) aplikacija

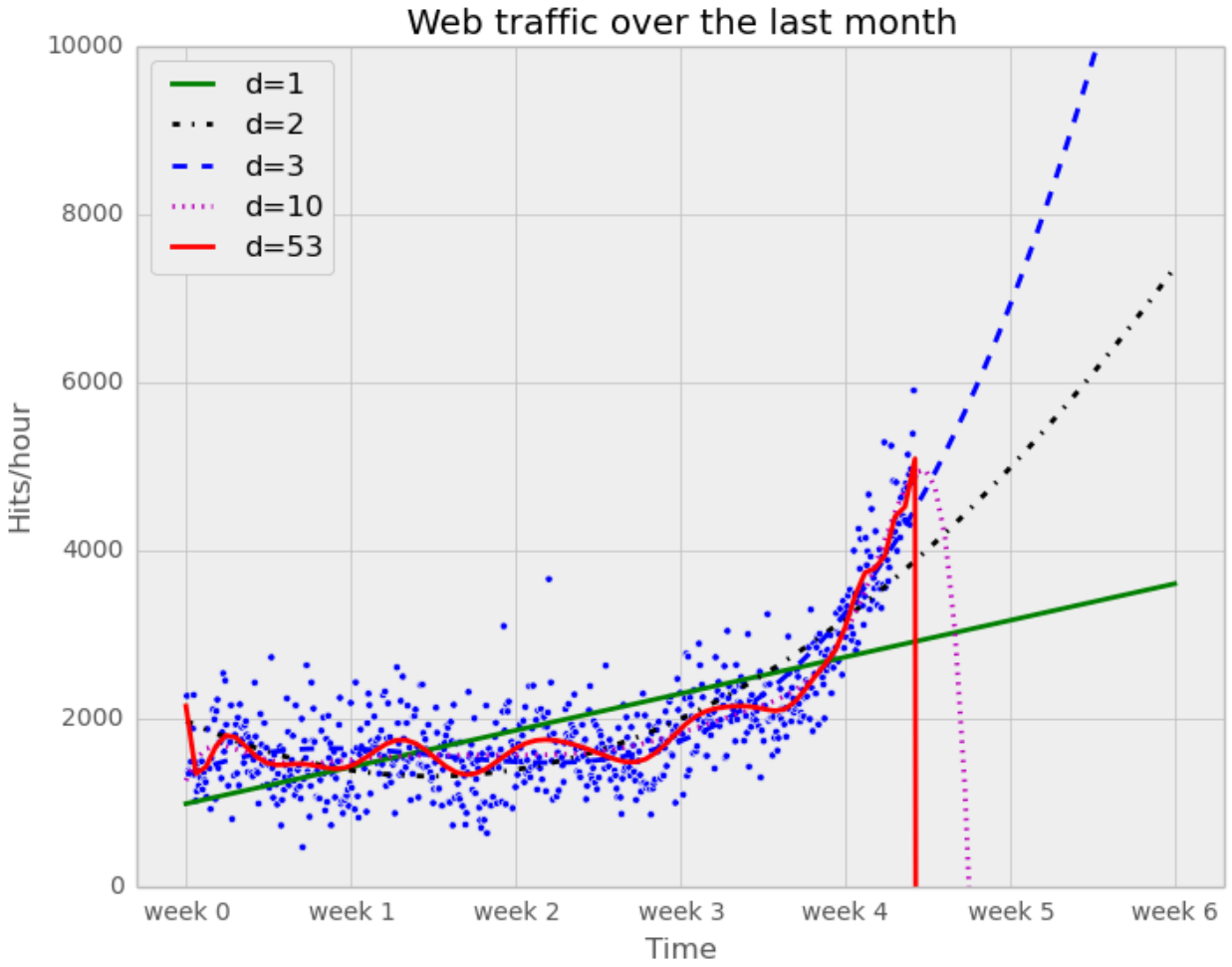


Prva (jednostavna) aplikacija

Zašto verujemo više dvema pravim linijama nego polinomima višeg reda?

- Očigledno, kobinacija dve prave linije se bolje prilagođava podacima nego bilo koji model koji smo do sada pomenuli.
- Međutim, kombinovana greška aproksimacije je veća nego greška aproksimacije polinoma višeg reda.
- Postavljamo sledeće pitanje: zašto verujemo više dvema pravim linijama od kojih se jedna bolje prilagođava u zadnjoj nedelji nego polinomima višeg reda?
- Odgovor na pitanje je sledeći: pretpostavljamo da će podatke u budućnosti bolje aproksimirati.
- Ukoliko iscrtamo model predviđanja „budućnosti“, grafik (v. str. 39) ukazuje na činjenicu da smo u pravu.

Prva (jednostavna) aplikacija



Prva (jednostavna) aplikacija

Šta se u slučaju predviđanja „budućnosti“ desilo?

- Modeli krivih definisani polinomima reda 10 i 53 su očigledno beskorisni za predviđanje zato što su prenaučeni. Drugim rečima, nemaju mogućnost generalizacije.
- S druge strane, modeli nižeg stepena se ne prilagođavaju dovoljno podacima (engl. *underfitting*).

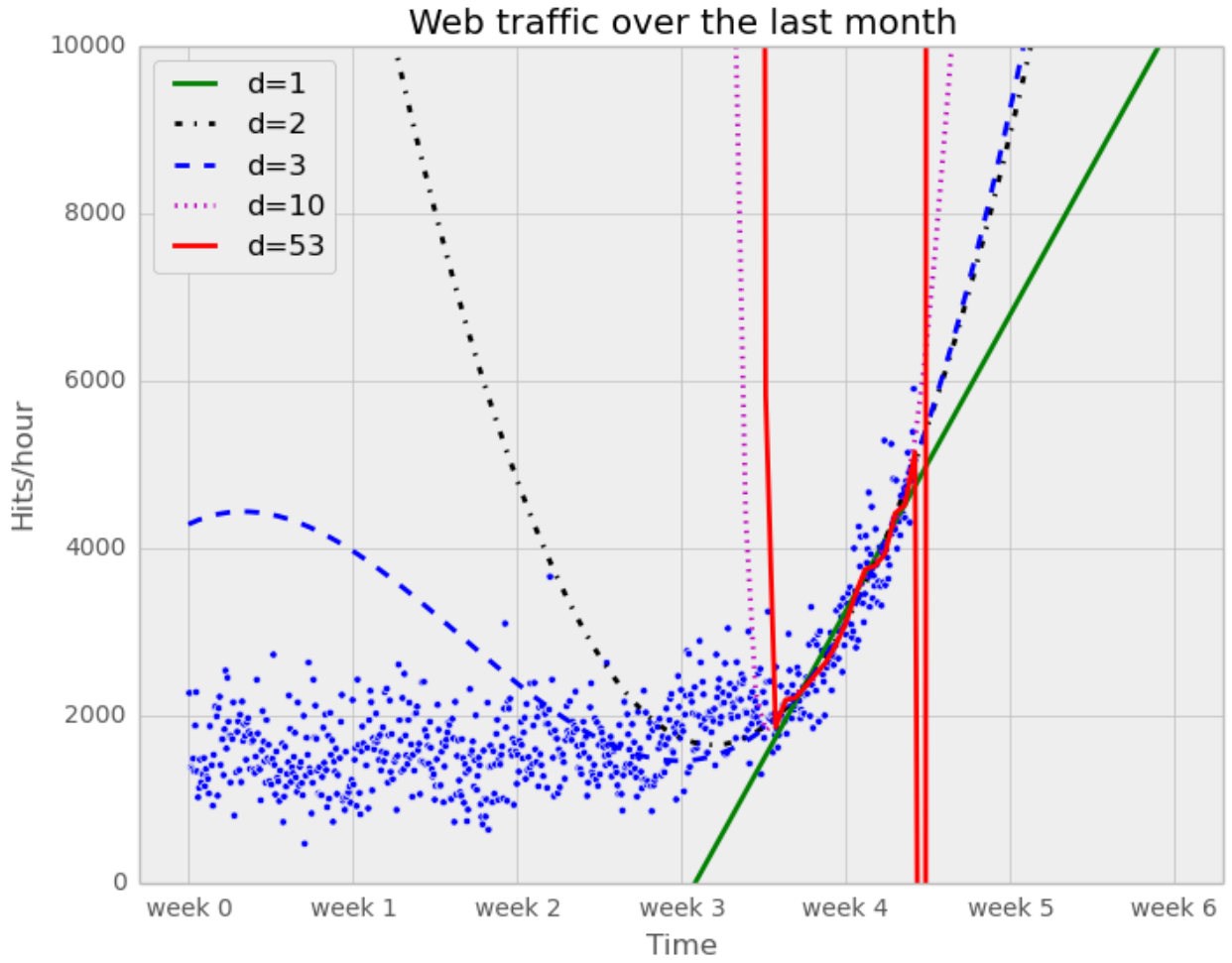
Prva (jednostavna) aplikacija

Ispitivanje modela viših stepena na podatke u poslednjoj nedelji.

- Na osnovu grešaka modela viših stepena obučenih podacima počev od nedelje 3,5 pa nadalje, može se zaključiti da je najsloženiji model ujedno i najpogodniji.
- Međutim, model reda 53 ni u ovom slučaju ne može da predvidi „budućnost“ jer je prenaučen.
- Sa „psihodeličnog grafikona“ na str. 42 se vidi koliko je prenaučenost složen problem.
- Napomena: u ovom slučaju greška je računata nakon tačke preloma.

```
Error d=1: 22,143,941.107618
Error d=2: 19,768,846.989176
Error d=3: 19,766,452.361027
Error d=10: 18,949,339.348539
Error d=53: 18,300,702.038119
```

Prva (jednostavna) aplikacija



Prva (jednostavna) aplikacija

Obuka i testiranje.

- Kada bi imali neke podatke „iz budućnosti“ koje bi mogli da iskoristimo da izmerimo tačnost naših modela, onda bi mogli da ocenimo izbor modela samo na osnovu greške aproksimacije.
- Međutim, mi takve podatke nemamo.
- „Sličan“ efekat se može postići ukoliko izdvojimo jedan deo podataka, a model obučimo ostatkom.
- Nakon toga, računamo grešku aproksimacije koristeći podatke koji su izdvojeni (odnosno, nisu korišćeni prilikom obuke).
- Na taj način može se dobiti realnija slika o tome kako će se model ponašati u budućnosti.

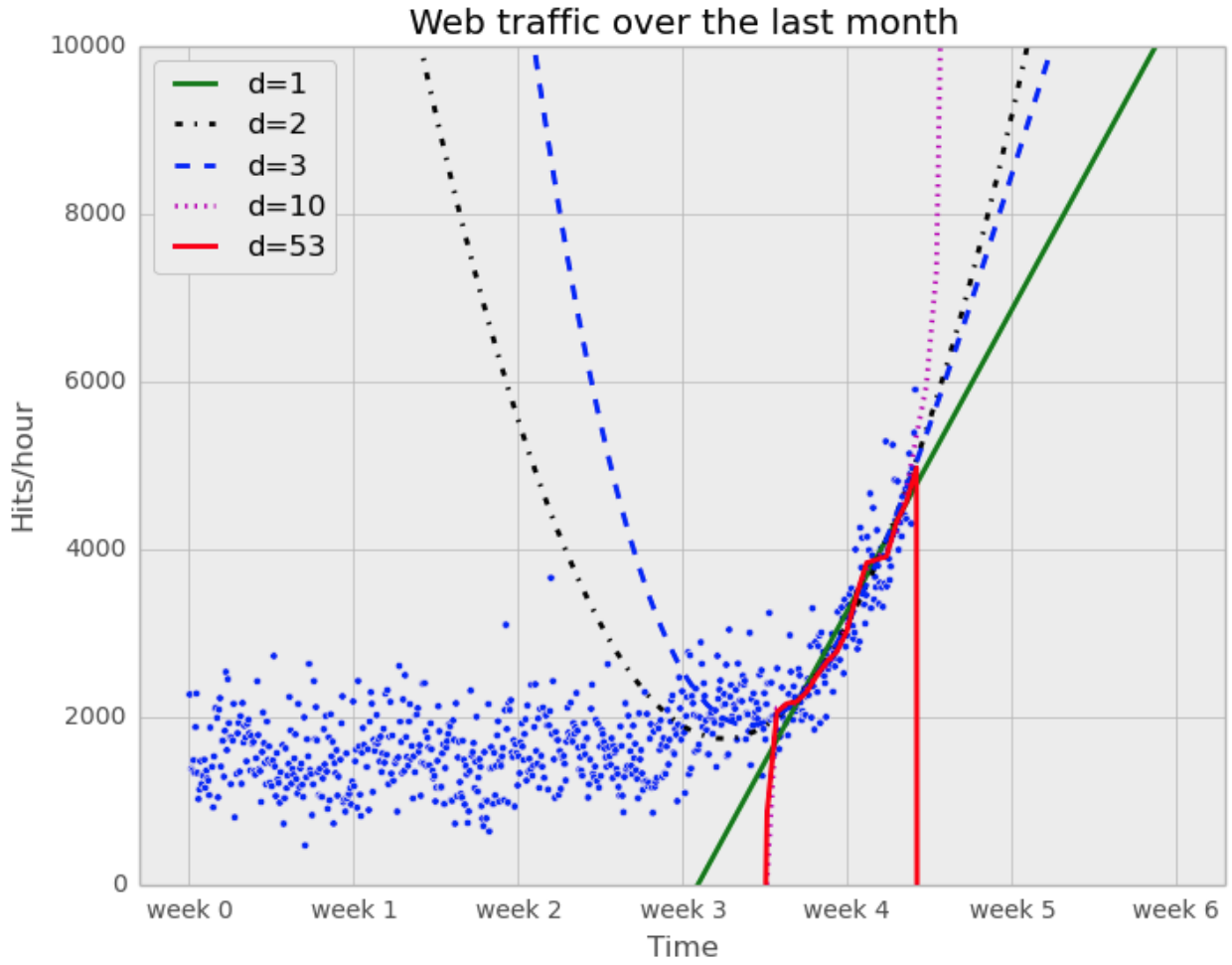
Prva (jednostavna) aplikacija

Obuka i testiranje.

```
Error d=1: 6397694.386394
Error d=2: 6010775.401243
Error d=3: 6047678.658525
Error d=10: 7037551.009519
Error d=53: 7052400.001761
```

- Model sa stepenom 2 ima najmanju grešku u testiranju, odnosno grešku koja se meri korišćenjem podataka koji su izdvojeni prilikom obuke.
- To ukazuje na šansu da neće doći do „iznenađenja“ prilikom obrade podataka „iz budućnosti“.

Prva (jednostavna) aplikacija



Prva (jednostavna) aplikacija

Odgovor na postavljeno pitanje.

- Formirali smo model za koji mislimo da najbolje opisuje proces koji je opisan podacima.
- Sledeći zadatak je da saznamo kada će naša infrastruktura dostići 100.000 zahteva po satu.
- Drugim rečima, moramo da izračunamo kada će funkcija (kriva definisana polinomom drugog reda) dostići vrednost 100.000.
- Jedan od načina je oduzimanje te vrednosti od polinoma, i računanje nule.
- Za to koristimo funkciju `solve()` SciPy optimize modula, kojoj zadajemo početnu poziciju x_0.
- Pošto imamo 743 zapisa u podacima od kojih svaki odgovara jednom satu, postavićemo početnu poziciju na malo veću vrednost (na primer, x_0 =800).
- Neka je `fbt2` polinom stepena dva.

Prva (jednostavna) aplikacija

Odgovor na postavljeno pitanje.

```
>>> fbt2 = sp.poly1d(sp.polyfit(xb[train], yb[train], 2))
>>> print("fbt2(x)= \n%s" % fbt2)
fbt2(x)=
       2
0.086 x - 94.02 x + 2.744e+04
>>> print("fbt2(x)-100,000= \n%s" % (fbt2-100000))
fbt2(x)-100,000=
       2
0.086 x - 94.02 x - 7.256e+04
>>> from scipy.optimize import fsolve
>>> reached_max = fsolve(fbt2-100000, x0=800)/(7*24)
>>> print("100,000 hits/hour expected at week %f" % reached_max[0])
```

Prva (jednostavna) aplikacija

Odgovor na postavljeno pitanje.

- Očekuje se da će infrastrukturi početi da pristižu 100.000 zahteva po satu u nedelji 9.616071, odnosno da model predviđa da će infrastruktura biti dovoljna još mesec dana.
- Naravno, treba uzeti u obzir da su ovo jednostavni modeli i da određena neizvesnost našeg predviđanja postoji.

Zaključne napomene

1. Uvek imajte na umu činjenicu da ćete veliki deo svog vremena provesti analizirajući vaše podatke, pokušavajuči da razumete njihovu prirodu, prečišćavati ih, predprocesirati itd. Ovaj mali eksperiment je trebao da Vam naznači da je Vaš fokus na podacima, a ne na algoritmima.
2. Izvedite eksperiment ispravno – nikada nemojte da koristite test podatke za obuku modela! Ovu početničku grešku prave mnogi, od kojih neki nisu početnici.

- Algoritam korišćen u ovom izlaganju nije najsjajnija stvar u mašinskom učenju. Složenije algoritme ćemo raditi u narednim predavanjima. Algoritam je odabran s ciljem da vam ne skrene pažnju sa prethodno pomenutih bitnih napomena.

Literatura

- Beleške pripremljene prema knjizi – Luis Pedro Coelho, Willi Richert (2015): „Building Machine Learning Systems with Python, Second Edition“. Packt Publishing.

Hvala na pažnji

Pitanja su dobrodošla.