grebena_regresija

August 23, 2017

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn import metrics
from sklearn import model_selection
%matplotlib inline
```

In [2]:
```
# radimo sa podacima koji se ticu bejzbol igraca
# hteli bismo na osnovu svih statistika o performansama igraca da predvidimo odgovarajucu
# podaci koji se koriste su preuzeti iz knjige Introduction to Statisctical Learning
# link do knjige je: http://www-bcf.usc.edu/~gareth/ISL/
data = pd.read_csv('hitters.csv')
```

In [3]:
```
# skup sadrzi 21 atribut (ukljucujuci i platu koju zelimo da predvidimo)
# i 322 informacije koje se ticu pojedinacnih igraca
data.shape
```

Out[3]: (322, 21)

In [4]:
```
data.head(5)
```

Out[4]:

	Player	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat
0	-Andy Allanson	293	66	1	30	29	14	1	293
1	-Alan Ashby	315	81	7	24	38	39	14	3449
2	-Alvin Davis	479	130	18	66	72	76	3	1624
3	-Andre Dawson	496	141	20	65	78	37	11	5628
4	-Andres Galarraga	321	87	10	39	42	30	2	396

	CHits	...	CRuns	CRBI	CWalks	League	Division	PutOuts	Assists
0	66	...	30	29	14	A	E	446	33
1	835	...	321	414	375	N	W	632	43
2	457	...	224	266	263	A	W	880	82
3	1575	...	828	838	354	N	E	200	11
4	101	...	48	46	33	N	E	805	40

	Errors	Salary	NewLeague
0	20	NaN	A

```
# Salary takodje treba ukloniti
X = data.drop(['Player', 'League', 'Division', 'NewLeague', 'Salary'], axi
print(X)
```

```
     AtBat  Hits  HmRun  Runs  RBI  Walks  Years  CAtBat  CHits  CHmRun  \
1      315    81      7    24   38     39     14    3449    835      69
2      479   130     18    66   72     76      3    1624    457      63
3      496   141     20    65   78     37     11    5628   1575     225
4      321    87     10    39   42     30      2     396    101      12
5      594   169      4    74   51     35     11    4408   1133      19
6      185    37      1    23    8     21      2     214     42       1
7      298    73      0    24   24      7      3     509    108       0
8      323    81      6    26   32      8      2     341     86       6
9      401    92     17    49   66     65     13    5206   1332     253
10     574   159     21   107   75     59     10    4631   1300      90
11     202    53      4    31   26     27      9    1876    467      15
12     418   113     13    48   61     47      4    1512    392      41
13     239    60      0    30   11     22      6    1941    510       4
14     196    43      7    29   27     30     13    3231    825      36
16     568   158     20    89   75     73     15    8068   2273     177
17     190    46      2    24    8     15      5     479    102       5
19     127    32      8    16   22     14      8     727    180      24
20     413    92     16    72   48     65      1     413     92      16
21     426   109      3    55   43     62      1     426    109       3
23     472   116     16    60   62     74      6    1924    489      67
24     629   168     18    73  102     40     18    8424   2464     164
25     587   163      4    92   51     70      6    2695    747      17
26     324    73      4    32   18     22      7    1931    491      13
27     474   129     10    50   56     40     10    2331    604      61
28     550   152      6    92   37     81      5    2308    633      32
29     513   137     20    90   95     90     14    5201   1382     166
31     419   108      6    55   36     22      3     591    149       8
33     583   168     17    83   80     56      5    1646    452      44
34     204    49      6    23   25     12      7    1309    308      27
35     379   106     10    38   60     30     14    6207   1906     146
..     ...   ...    ...   ...  ...    ...    ...     ...    ...     ...
287    687   213     10    91   65     27      4    1518    448      15
288    368   103      3    48   28     54      8    1897    493       9
289    263    70      1    26   23     30      4     888    220       9
290    642   211     14   107   59     52      5    2364    770      27
291    265    68      8    26   30     29      7    1337    339      32
293    559   141      2    48   61     73      8    3162    874      16
294    520   120     17    53   44     21      4     927    227      22
295     19     4      1     2    3      1      1      19      4       1
296    205    43      2    24   17     20      7     854    219      12
297    193    47     10    21   29     24      6    1136    256      42
299    213    61      4    17   22      3     17    4061   1145      83
300    510   147     10    56   52     53      7    2872    821      63
```

301	578	138	1	56	59	34	3	1399	357	7
303	441	113	5	76	52	76	5	1546	397	17
304	172	42	3	17	14	15	10	4086	1150	57
306	127	32	4	14	25	12	19	8396	2402	242
307	279	69	4	35	31	32	4	1359	355	31
308	480	112	18	50	71	44	7	3031	771	110
309	600	139	0	94	29	60	2	1236	309	1
310	610	186	19	107	98	74	6	2728	753	69
311	360	81	5	37	44	37	7	2268	566	41
312	387	124	1	67	27	36	7	1775	506	6
313	580	207	8	107	71	105	5	2778	978	32
314	408	117	11	66	41	34	1	408	117	11
315	593	172	22	82	100	57	1	593	172	22
317	497	127	7	65	48	37	5	2703	806	32
318	492	136	5	76	50	94	12	5511	1511	39
319	475	126	3	61	43	52	6	1700	433	7
320	573	144	9	85	60	78	8	3198	857	97
321	631	170	9	77	44	31	11	4908	1457	30

	CRuns	CRBI	CWalks	PutOuts	Assists	Errors
1	321	414	375	632	43	10
2	224	266	263	880	82	14
3	828	838	354	200	11	3
4	48	46	33	805	40	4
5	501	336	194	282	421	25
6	30	9	24	76	127	7
7	41	37	12	121	283	9
8	32	34	8	143	290	19
9	784	890	866	0	0	0
10	702	504	488	238	445	22
11	192	186	161	304	45	11
12	205	204	203	211	11	7
13	309	103	207	121	151	6
14	376	290	238	80	45	8
16	1045	993	732	105	290	10
17	65	23	39	102	177	16
19	67	82	56	202	22	2
20	72	48	65	280	9	5
21	55	43	62	361	22	2
23	242	251	240	518	55	3
24	1008	1072	402	1067	157	14
25	442	198	317	434	9	3
26	291	108	180	222	3	3
27	246	327	166	732	83	13
28	349	182	308	262	329	16
29	763	734	784	267	5	3
31	80	46	31	226	7	4
33	219	208	136	109	292	25

```
34     126   132     66     419     46     5
35     859   803    571      72    170    24
..     ...   ...    ...     ...    ...   ...
287    196   137     89     294    445    13
288    207   162    198     209    246     3
289     83    82     86      81    147     4
290    352   230    193     337     19     4
291    135   163    128      92      5     3
293    421   349    359     352    414     9
294    106    80     52      70    144    11
295      2     3      1     692     70     8
296    105    99     71     131      6     1
297    129   139    106     299     13     5
299    488   491    244     178     45     4
300    307   340    174     810     99    18
301    149   161     87     133    371    20
303    226   149    191     160    290    11
304    579   363    406      65      0     0
306   1048  1348    819     167     18     6
307    180   148    158     133    173     9
308    338   406    239      94    270    16
309    201    69    110     300     12     9
310    399   366    286    1182     96    13
311    279   257    246     170    284     3
312    272   125    194     186    290    17
313    474   322    417     121    267    19
314     66    41     34     942     72    11
315     82   100     57    1222    139    15
317    379   311    138     325      9     3
318    897   451    875     313    381    20
319    217    93    146      37    113     7
320    470   420    332    1314    131    12
321    775   357    249     408      4     3

[263 rows x 16 columns]
```

```
In [13]: X.shape

Out[13]: (263, 16)

In [11]: # posto su skale vrednosti podataka neujednacene, ima smisla raditi normalizaciju podataka

In [12]: # delimo skup podataka na:
         # trening skup
         # validacioni skup koji koristimo za ucenje metaparametara
         # test skup na kojem ocenjujemo uspesnost modela
```

```
X_train_validation, X_test, Y_train_validation, Y_test = model_selection.t
X_train, X_validation, Y_train, Y_validation = model_selection.train_test_
```

In [13]:
```
# baseline - linearna regresija
linreg = linear_model.LinearRegression(normalize=True)
linreg.fit(X_train_validation, Y_train_validation)
Y_predicted = linreg.predict(X_test)
linreg_score = metrics.mean_squared_error(Y_test, Y_predicted)
linreg_score
```

Out[13]:
```
138551.12033732756
```

In [14]:
```
# grebena regresija sa metaparametrom 4
reg_1= linear_model.Ridge(alpha=4, normalize=True)
reg_1.fit(X_train_validation, Y_train_validation)
Y_predicted = reg_1.predict(X_test)
reg_score_1 = metrics.mean_squared_error(Y_test, Y_predicted)
reg_score_1
# na osnovu ovog rezultata zakljucujemo da ima smisla raditi regularizaciju jer smo dobili
```

Out[14]:
```
126928.00669545057
```

In [15]:
```
# grebena regresija sa metaparametrom 10^10
reg_2= linear_model.Ridge(alpha=10**10, normalize=True)
reg_2.fit(X_train_validation, Y_train_validation)
Y_predicted = reg_2.predict(X_test)
reg_score_2 = metrics.mean_squared_error(Y_test, Y_predicted)
reg_score_2
# na osnovu ovog rezultata zakljucujemo da ima smisla odrediti i precizno vrednost parame
# jer ocigledno nisu sve regularizacije dobre
```

Out[15]:
```
194839.49321934962
```

In [45]:
```
# skup vrednosti iz kojeg trazimo najbolju vrednost za parametar alfa
# imamo i male i velike vrednosti u skupu
alphas = 10**np.linspace(10, -2, 100)*0.5
```

In [46]:
```
alphas
```

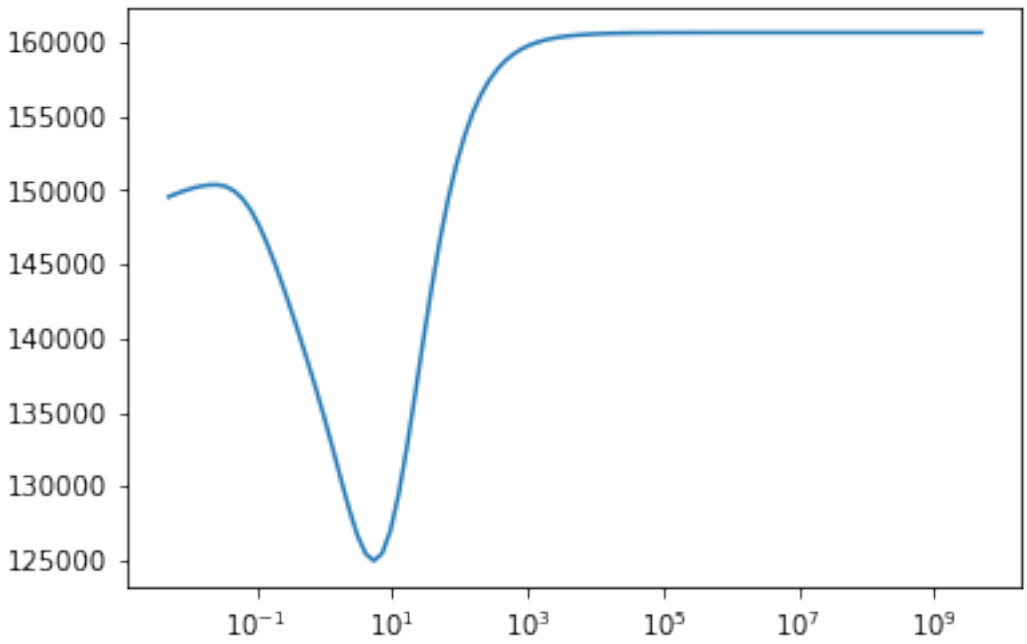
Out[46]:
```
array([  5.00000000e+09,   3.78231664e+09,   2.86118383e+09,
         2.16438064e+09,   1.63727458e+09,   1.23853818e+09,
         9.36908711e+08,   7.08737081e+08,   5.36133611e+08,
         4.05565415e+08,   3.06795364e+08,   2.32079442e+08,
         1.75559587e+08,   1.32804389e+08,   1.00461650e+08,
         7.59955541e+07,   5.74878498e+07,   4.34874501e+07,
         3.28966612e+07,   2.48851178e+07,   1.88246790e+07,
         1.42401793e+07,   1.07721735e+07,   8.14875417e+06,
         6.16423370e+06,   4.66301673e+06,   3.52740116e+06,
         2.66834962e+06,   2.01850863e+06,   1.52692775e+06,
```

```
       1.15506485e+06,   8.73764200e+05,   6.60970574e+05,
       5.00000000e+05,   3.78231664e+05,   2.86118383e+05,
       2.16438064e+05,   1.63727458e+05,   1.23853818e+05,
       9.36908711e+04,   7.08737081e+04,   5.36133611e+04,
       4.05565415e+04,   3.06795364e+04,   2.32079442e+04,
       1.75559587e+04,   1.32804389e+04,   1.00461650e+04,
       7.59955541e+03,   5.74878498e+03,   4.34874501e+03,
       3.28966612e+03,   2.48851178e+03,   1.88246790e+03,
       1.42401793e+03,   1.07721735e+03,   8.14875417e+02,
       6.16423370e+02,   4.66301673e+02,   3.52740116e+02,
       2.66834962e+02,   2.01850863e+02,   1.52692775e+02,
       1.15506485e+02,   8.73764200e+01,   6.60970574e+01,
       5.00000000e+01,   3.78231664e+01,   2.86118383e+01,
       2.16438064e+01,   1.63727458e+01,   1.23853818e+01,
       9.36908711e+00,   7.08737081e+00,   5.36133611e+00,
       4.05565415e+00,   3.06795364e+00,   2.32079442e+00,
       1.75559587e+00,   1.32804389e+00,   1.00461650e+00,
       7.59955541e-01,   5.74878498e-01,   4.34874501e-01,
       3.28966612e-01,   2.48851178e-01,   1.88246790e-01,
       1.42401793e-01,   1.07721735e-01,   8.14875417e-02,
       6.16423370e-02,   4.66301673e-02,   3.52740116e-02,
       2.66834962e-02,   2.01850863e-02,   1.52692775e-02,
       1.15506485e-02,   8.73764200e-03,   6.60970574e-03,
       5.00000000e-03])
```

In [47]:
```
# niz gresaka nasih modela
errors = []
```

In [48]:
```
# izracunavamo gresku za skvaku vrednost metaparametra alfa
for a in alphas:
    reg = linear_model.Ridge(normalize=True,  alpha=a)
    reg.fit(X_train, Y_train)
    Y_predicted = reg.predict(X_validation)
    error = metrics.mean_squared_error(Y_predicted, Y_validation)
    errors.append(error)
```

In [49]:
```
# iscrtavamo zavisnost greske od vrednosti metaparametra alfa
plt.plot(alphas, errors)
plt.xscale('log')
```




```
In [50]: # trazimo koja je to vrednost

In [51]: type(errors)

Out[51]: list

In [52]: erros = np.array(errors)

In [53]: erros.argmin()

Out[53]: 74

In [54]: alpha = alphas[74]
         alpha

Out[54]: 5.3613361100516048

In [55]: # greska modela
         errors[74]

Out[55]: 124988.0740587797

In [56]: # sada kada znamo najbolju vrednost metaparametra alfa mozemo da naucimo finalni mod
         # da damo konacnu procenu greske
         # ne zaboraviti da se u ovom koraku koriste i trening i validacioni skup

         regfin = linear_model.Ridge(alpha=5.361, normalize=True)
```