Tarea N°1 - Máquinas de Aprendizaje - ILI393

Martín Villanueva A.

Introducción

En es primera tarea, se tiene como objetivo la implementación y testeo de algoritmos para regresión lineal y regresión logística. En ambos casos la busqueda de los mejores parámetros del modelo se realiza por medio de *Gradiente Descendente* (batch y online) y *Newton-Raphson*. Para la correcta selección de los *hiperparámetros* se realiza 5-fold crossvalidation, intentando de este modo que los modelos resultantes no caigan en problemas de *overfitting*.

Parte 1 - Regresión Lineal

In [29]:

```
#alphas to try on raw data
alphas1 = np.linspace(4.0e-7, 4.5e-7, 5, endpoint=True)
#alphas to try on rescaled and normalized data
alphas2 = np.array([1.0e-3, 0.8e-3, 0.6e-3, 0.4e-3, 0.2e-3])
```

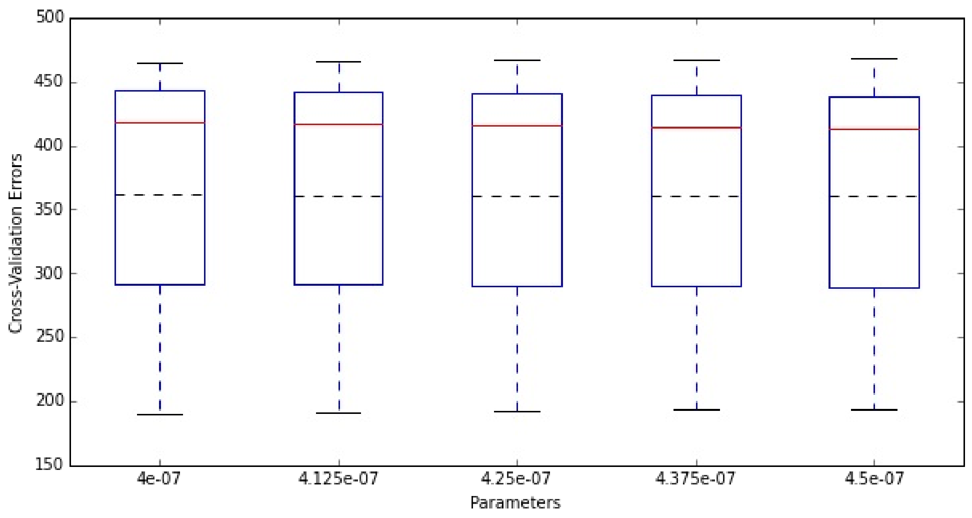
1a) Gradiente Descendente Batch

Raw data

In [42]:

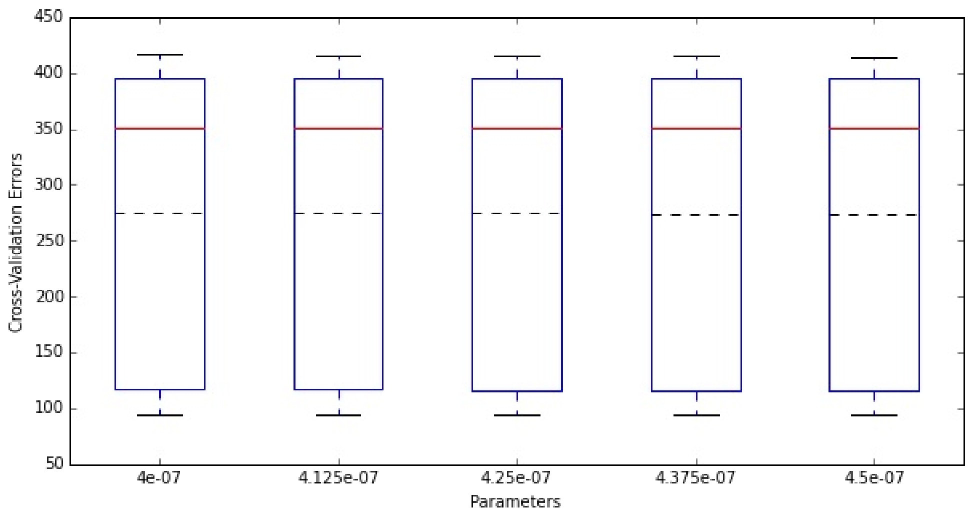
```
solve_regression(gd_batch, 'linear', params=alphas1, show=[0,14])
```

```
##############################################################
Dataset: 0
Best alpha: 4.5e-07
```

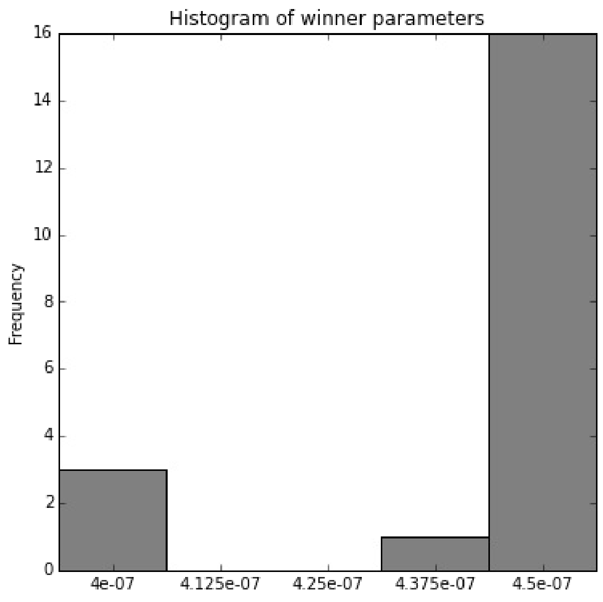


```
Training error: 291.529613576
Testing error: 224.340517783
N° iterations: 64
Beta: [ 0.00759594  0.25955161  0.02174389 -0.0171124  -0.01772088  0.01810715
  0.13438221 -0.04736688  0.12759068 -0.03385616  0.01151583  0.0052043
  0.0076039 ]
##############################################################


##############################################################
Dataset: 14
Best alpha: 4.5e-07
```



```
Training error: 236.271311927
Testing error: 424.701874
N° iterations: 53
Beta: [ 0.00555882  0.25899926  0.02076567 -0.00916217 -0.00514845  0.00739162
  0.10491754 -0.03100194  0.1036039   0.02771736  0.01037837  0.00402665
  0.0064281 ]
##############################################################
```

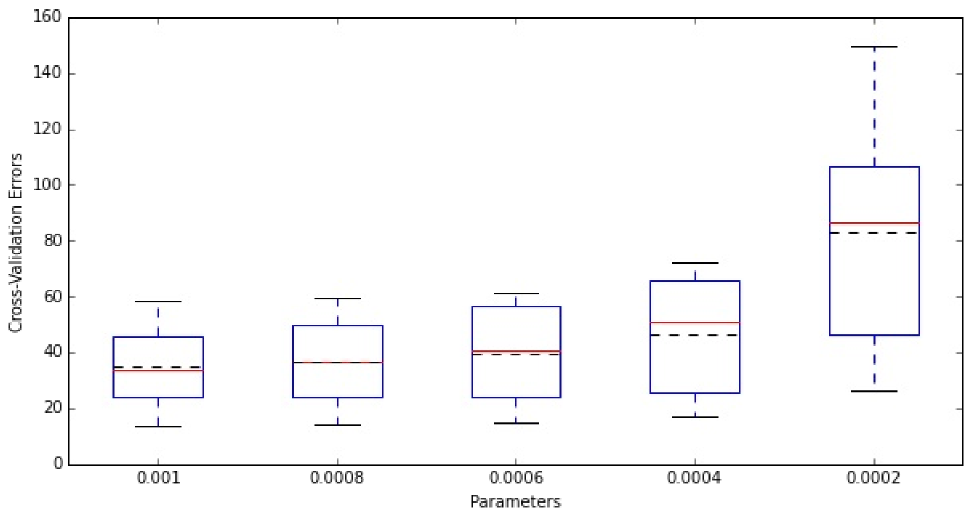


Rescaled data

In [44]:

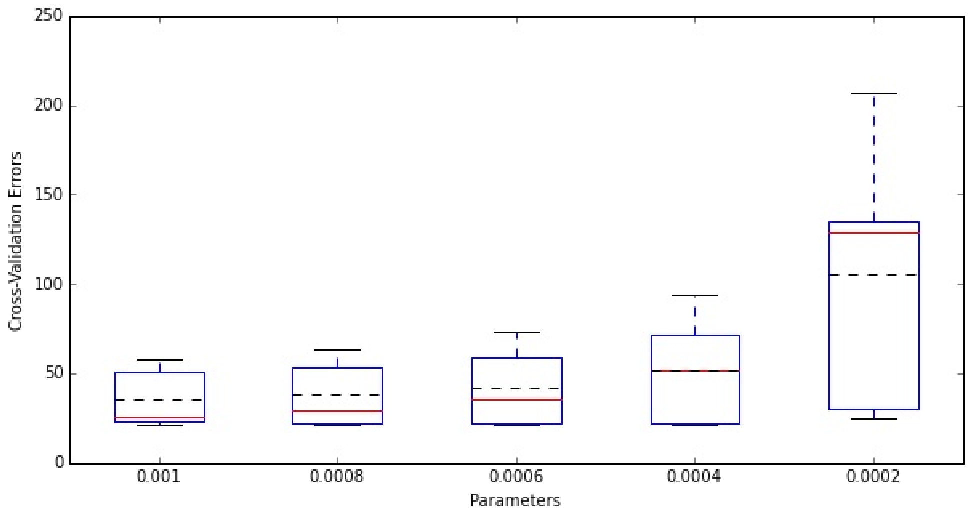
```
solve_regression(gd_batch, 'linear', params=alphas2, data_func=rescale, show=[0,14])
```

```
##############################################################
Dataset: 0
Best alpha: 0.001
```

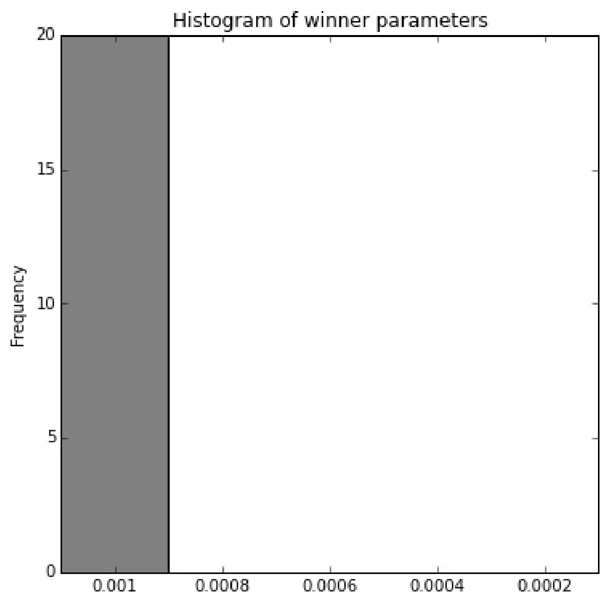


```
Training error: 20.6182739329
Testing error: 42.5328851194
N° iterations: 530
Beta: [ 39.27502848 -13.60536851  17.88310933 -16.95659792 -13.32113539
  21.00349746  14.26238428 -17.86462666  10.29693403 -11.62685618
   3.58148223   4.97282493  18.67876439]
##############################################################


##############################################################
Dataset: 14
Best alpha: 0.001
```



```
Training error: 21.9788628555
Testing error: 91.1000283764
N° iterations: 548
Beta: [ 31.86746216 -11.73989936  18.4682303  -15.58587584 -10.42352654
  16.69472148  16.37413566 -14.74326499   7.4469539   -7.23761563
   4.18365522   3.54434565  18.06400087]
##############################################################
```

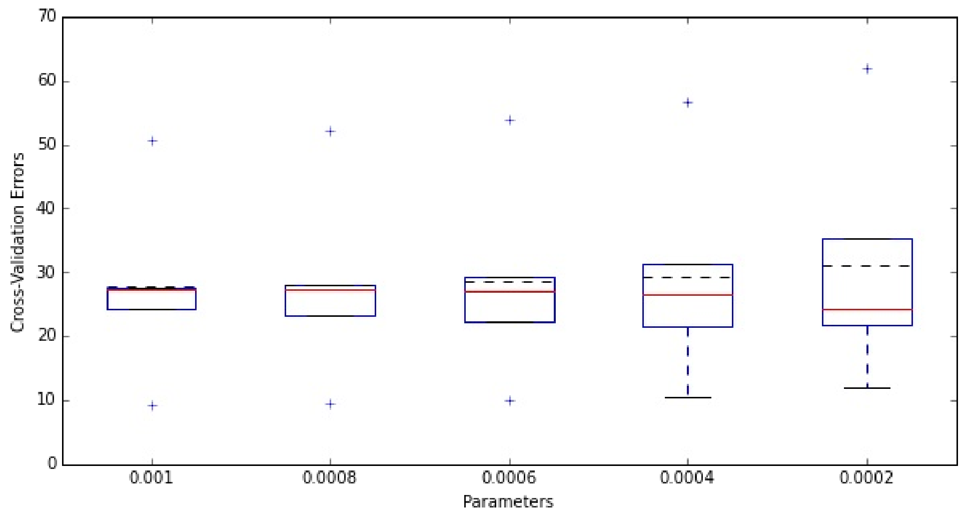


Normalized data

In [45]:

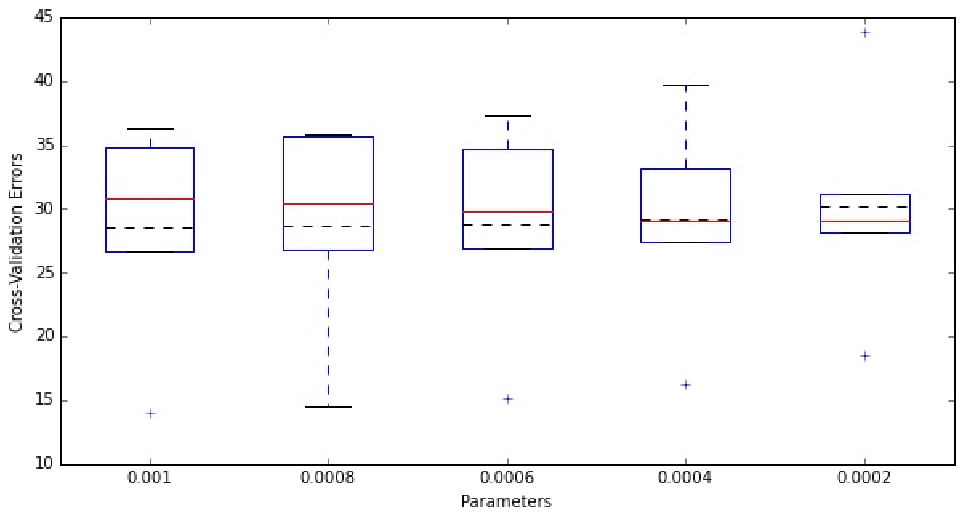
```
solve_regression(gd_batch, 'linear', params=alphas2, data_func=normalize, show=[0,14])
```

```
################################################################
Dataset: 0
Best alpha: 0.001
```

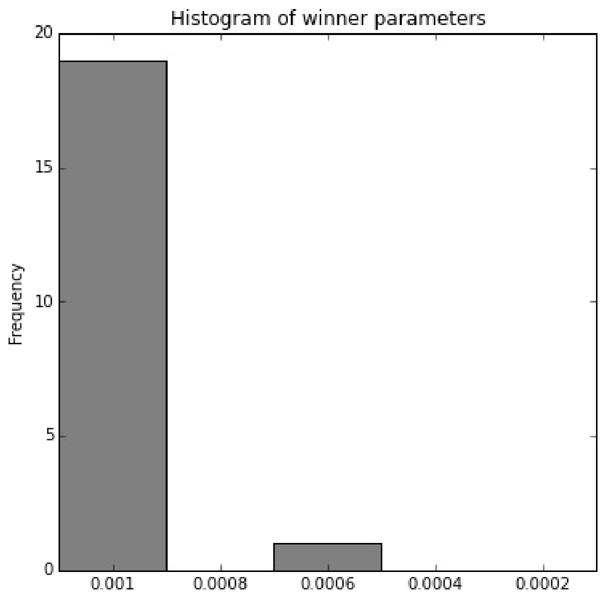


```
Training error: 13.4194733738
Testing error: 36.5826804086
N° iterations: 137
Beta: [ 42.22626947  -4.02131789   4.23055468  -3.50410785  -3.8804333
   6.72285551   3.1979057   -4.13343304  -0.61328431  -3.02687997
   1.98386787   1.43159348   3.59464352]
################################################################


################################################################
Dataset: 14
Best alpha: 0.001
```



```
Training error: 16.3098751254
Testing error: 65.7517896491
N° iterations: 96
Beta: [ 40.87319656  -3.87689122   4.66900552  -3.56542152  -3.20285147
   5.02062112   2.99775041  -3.08497395   0.46571746  -1.71364208
   1.8960333    1.21081326   3.61273617]
##############################################################
```



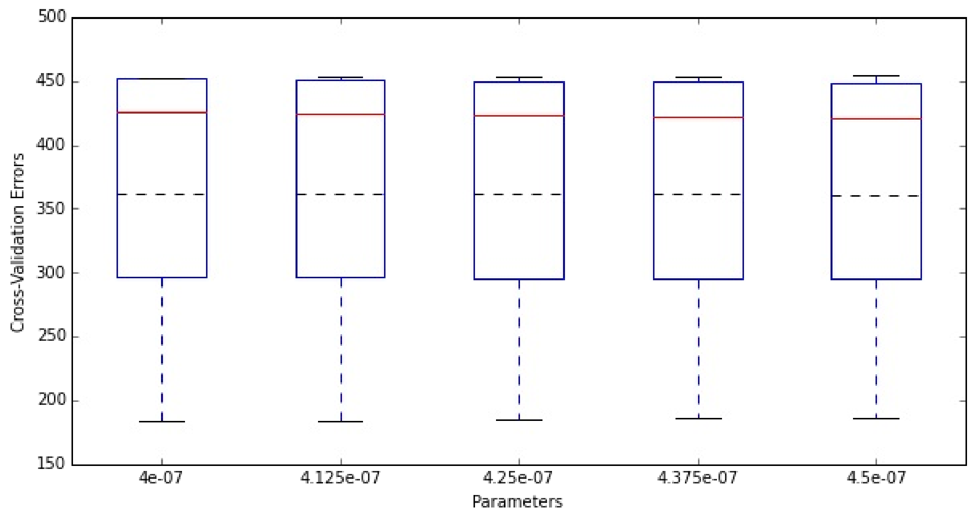
1b) Gradiente Descendente Online

Raw data

In [46]:

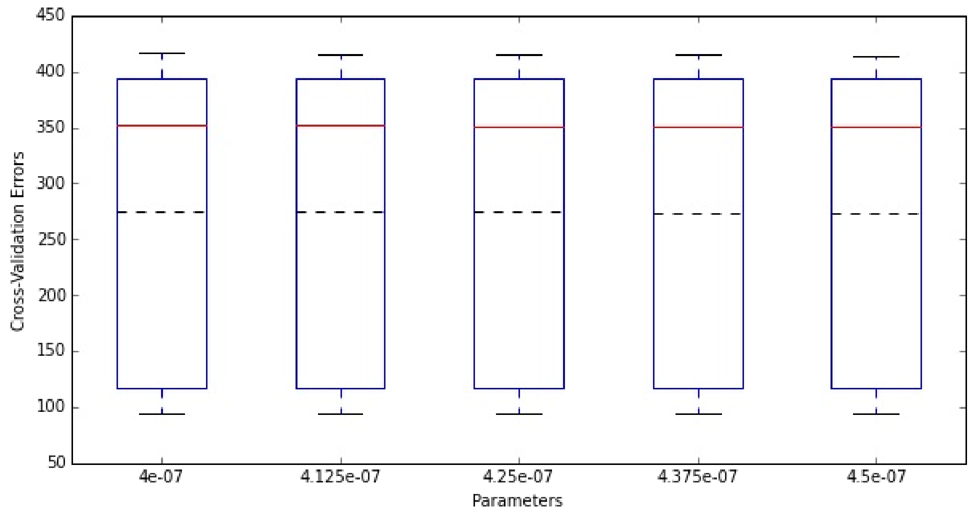
```
solve_regression(gd_online, 'linear', params=alphas1, show=[0,14])
```

```
##############################################################
Dataset: 0
Best alpha: 4.5e-07
```

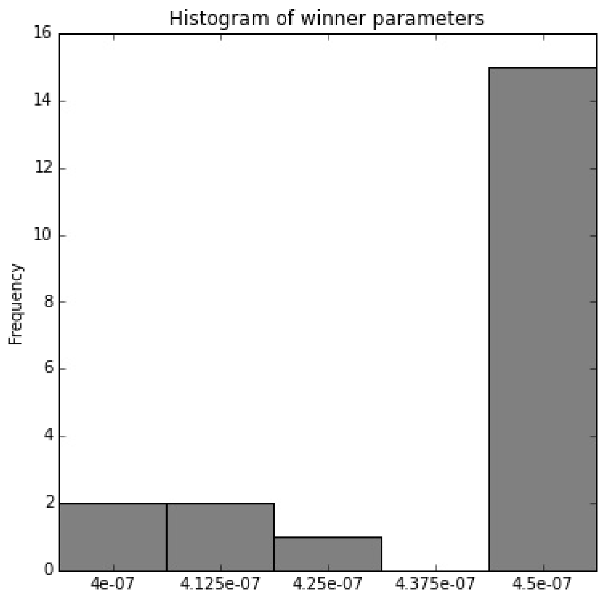


```
Training error: 292.641332227
Testing error: 216.286212008
N° iterations: 66
Beta: [ 0.00777387  0.26225985  0.0228718  -0.01769797 -0.01559905  0.0187834
  0.13911965 -0.05018793  0.13196834 -0.03041284  0.01187493  0.00535709
  0.00780838]
##############################################################


##############################################################
Dataset: 14
Best alpha: 4.5e-07
```



```
Training error: 236.224475078
Testing error: 425.608168754
N° iterations: 54
Beta: [ 0.0056634   0.25890305  0.02099604 -0.00953796 -0.00318058  0.00795135
  0.10658788 -0.03180157  0.10336748  0.02722773  0.01067373  0.00410388
  0.00649339]
##############################################################
```

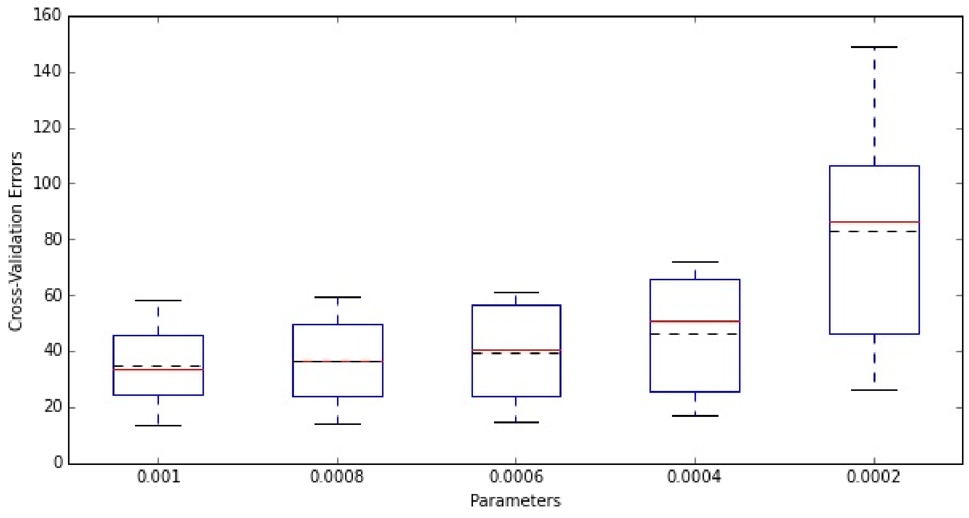


Rescaled data

```
In [47]:
```

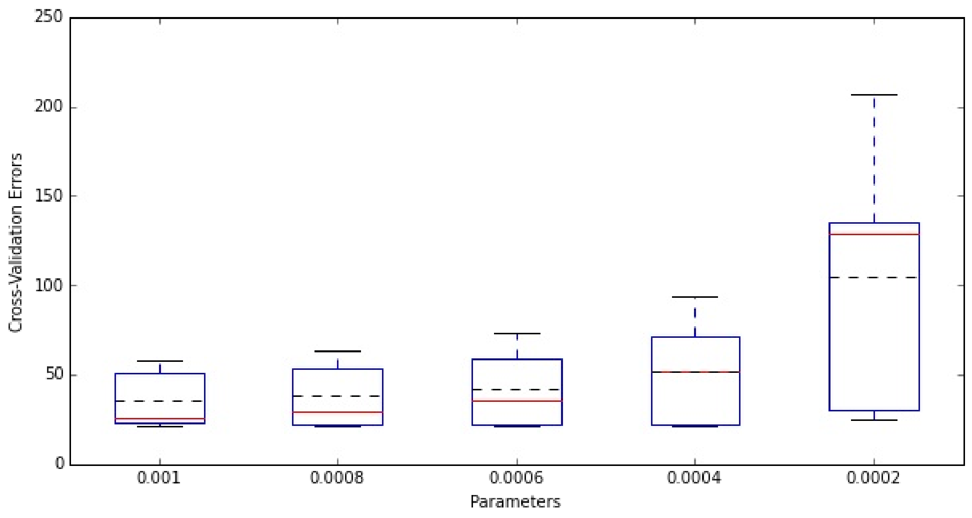
```
solve_regression(gd_online, 'linear', params=alphas2, data_func=rescale, show=[0,14])
```

```
##############################################################
Dataset: 0
Best alpha: 0.001
```

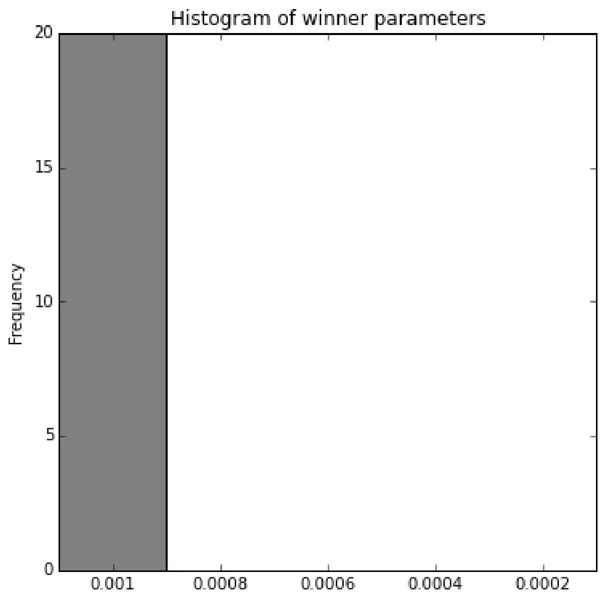


```
Training error: 20.6132205513
Testing error: 42.5978431746
N° iterations: 531
Beta: [ 39.25933591 -13.60674956  17.89442223 -16.95968836 -13.34024487
  21.02057345  14.28707965 -17.853764    10.30068558 -11.61868158
   3.58670305   4.99221433  18.68272861]
##############################################################


##############################################################
Dataset: 14
Best alpha: 0.001
```



```
Training error: 21.9786834767
Testing error: 90.8722318343
N° iterations: 548
Beta: [ 31.87057958 -11.74525815  18.46781379 -15.57800553 -10.41308722
  16.7220343   16.37303632 -14.73147343   7.43757436  -7.23999133
   4.19564227   3.56494144  18.04271745]
##############################################################
```

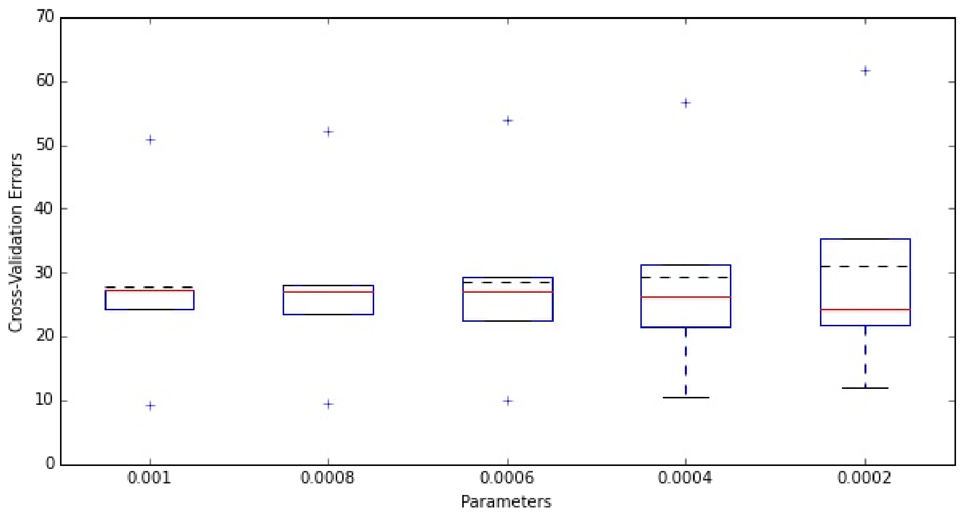


Normalized data

In [48]:

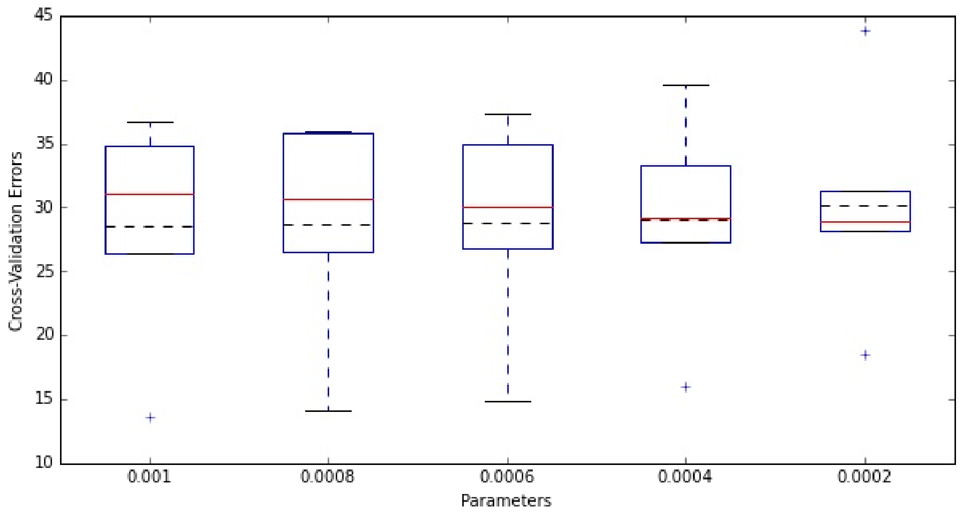
```
solve_regression(gd_online, 'linear', params=alphas2, data_func=normalize, show=[0,14])
```

```
##############################################################
Dataset: 0
Best alpha: 0.001
```

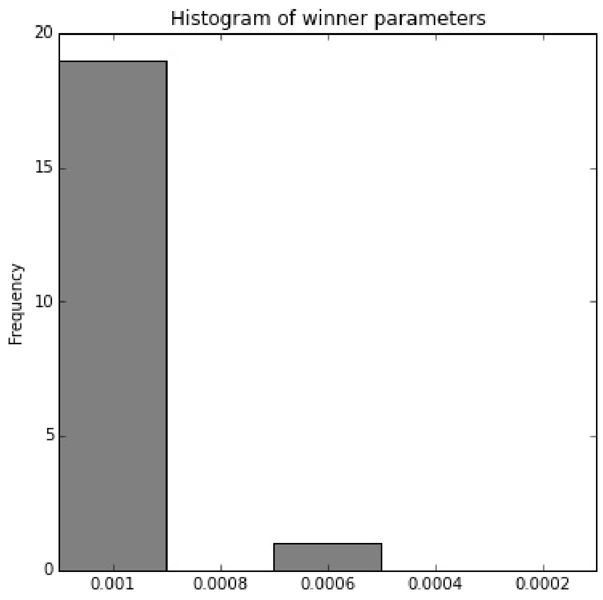


```
Training error: 13.4160931261
Testing error: 36.4127700422
N° iterations: 137
Beta: [ 42.22489559  -4.01812148   4.22806184  -3.50756328  -3.86158705
   6.74149223   3.19810492  -4.11134324  -0.61343112  -3.0115829
   1.97701226   1.44230261   3.57298429]
##############################################################


##############################################################
Dataset: 14
Best alpha: 0.001
```



```
Training error: 16.3153752729
Testing error: 65.4658388653
N° iterations: 95
Beta: [ 40.88418912  -3.86109769   4.6767013   -3.52334001  -3.23350274
   5.05247161   3.00863291  -3.07361825   0.48458418  -1.72351009
   1.90061769   1.19267068   3.59741601]
##############################################################
```



1c) Newton Raphson

Raw data

In [33]:

```
solve_regression(nr_linear, 'linear', show=[0,14])
```

```
##############################################################
Dataset: 0
Training error: 12.690235978
Testing error: 29.8692128464
N° iterations: 2
Beta: [ 27.77312374  -0.19196441   3.64271996  -2.61978866  -0.04940511
   3.27558724   0.8950743   -0.94639251  -0.03448292  -0.13677948
   2.89862662  12.4397042   13.71043458]
##############################################################


##############################################################
Dataset: 14
Training error: 15.5168045543
Testing error: 22.256882111
N° iterations: 2
Beta: [  2.92044009e+01  -2.45567403e-01   3.72846884e+00  -2.50624227e+00
  -3.72800837e-02   2.92327199e+00   8.33168001e-01  -6.98788024e-01
  -1.71242668e-02  -1.04831696e-01   2.76379484e+00   1.24136911e+01
   1.31720844e+01]
##############################################################
```

Rescaled data

```
In [35]:
```

```
solve_regression(nr_linear, 'linear', data_func=rescale, show=[0,14])
```

```
##############################################################
Dataset: 0
Training error: 12.690235978
Testing error: 70.8230012379
N° iterations: 2
Beta: [ 37.68776506 -21.30804932  18.21359978 -15.71873198 -15.80963497
  45.85822131  16.11133741 -17.03506525 -10.86211949 -13.67794787
   5.79725325  12.4397042   24.27432442]
##############################################################


##############################################################
Dataset: 14
Training error: 15.5168045543
Testing error: 153.683962628
N° iterations: 2
Beta: [ 32.95912586 -27.25798171  18.64234419 -15.03745362 -11.92962679
  29.23271995  18.32969603 -12.57818444  -5.37701976 -10.48316963
   5.52758969  12.41369113  23.32117544]
##############################################################
```

Normalized data

```
In [36]:
```

```
solve_regression(nr_linear, 'linear', data_func=normalize, show=[0,14])
```

```
##############################################################
Dataset: 0
Training error: 12.690235978
Testing error: 39.1167896725
N° iterations: 2
Beta: [ 42.22724     -3.99216233   4.26196505  -3.44486437  -3.93375017
   8.28671753   3.23536861  -4.20056151  -2.37742955  -3.2700182
   2.44555845   1.89445678   3.72993749]
##############################################################


##############################################################
Dataset: 14
Training error: 15.5168045543
Testing error: 61.1820073443
N° iterations: 2
Beta: [ 40.89617333  -4.54027498   4.63264649  -3.14293395  -2.94434592
   6.47945038   3.19280105  -3.05719739  -1.09358413  -2.01526648
   2.3439993    1.82778689   3.80201253]
##############################################################
```

2a) Gradiente Descendente Batch

DataSet	Training MSE raw data	Training MSE rescaled data	Training MSE normalized data	Testing MSE raw data	Testing MSE rescaled data	Testing MSE normalized data
0	47.7018651247	13.0062050605	12.7266740348	98.5365507026	65.2571301369	38.4079585793
1	64.7550597054	16.3602257442	16.0715344155	48.2315757796	92.0230671423	65.1941175822
2	60.7481463585	18.1457842322	17.7788520385	79.3615414274	115.610985266	18.2024811924
3	58.9987667645	14.1017329842	13.9377761345	97.6149558458	171.105766665	48.6525469488
4	60.3072912204	15.5749130727	15.3894271276	88.023192366	55.4477742699	24.1581873664
5	50.8161360455	14.2610314693	13.3703648326	103.273046989	53.7595016258	27.1554482493
6	65.6541052631	17.0409159049	15.7806050056	57.5892393027	138.507544424	50.1360687667
7	55.4439862945	16.7780811429	16.5841191369	75.1149772436	47.9171960704	28.077556808
8	54.5426469971	17.6594834655	17.0797580788	82.2107819498	27.7334819973	18.6923467845
9	62.4168368555	16.3081104286	15.8926108555	59.7595593067	337.991435488	32.4322681958
10	54.2623055884	13.4248859313	13.2809364772	102.249628704	76.0132818686	51.964392846
11	62.0340545439	15.8168112322	15.1229661239	96.3581158257	29.6423010367	58.2778323768
12	60.1712404807	14.7897334494	14.2704574864	47.6982336899	79.6061371602	33.4736955049
13	63.8754224266	13.5218597059	13.3686890914	54.6855768246	246.084167693	49.4021401471
14	55.3889121746	16.0666898014	15.5571989603	87.9559175097	117.196223122	62.1502546032
15	46.1065815649	7.96105164535	7.7831599526	120.74160843	102.779537282	103.539701305
16	58.7921426086	10.6974314356	10.4686233286	82.0169113703	509.388015464	79.8182740031
17	65.6326333879	14.3023622226	14.0575024157	50.8059305089	72.9172185356	39.4462557317
18	56.5555399322	16.4039458304	16.1738495233	89.5246242367	101.716074157	19.219045776
19	53.8965499152	13.1956721947	12.7453067041	101.423157308	79.6597927613	41.4905102572

Cuadro 1: MSE obtenidos en cada dataset con **Gradiente Descendente Batch**

2b) Gradiente Descendente Online

DataSet	Training MSE raw data	Training MSE rescaled data	Training MSE normalized data	Testing MSE raw data	Testing MSE rescaled data	Testing MSE normalized data
0	47.893692584	13.0074188914	12.7265440344	92.0966603914	65.2775158068	38.3704391845
1	64.6988609429	16.3629180412	16.0711580478	47.9123694862	93.6913520605	65.2908059333
2	60.9392772213	18.1476046506	17.7880949034	78.1358655876	115.411791641	18.0041725612
3	58.9787768796	14.1002166575	13.9375577433	99.2667062665	171.566596167	48.6369000543
4	60.1774212144	15.5749021184	15.3892917875	88.333392035	55.4869693485	24.1745417821
5	51.0422341216	14.260531204	13.370563385	103.788709289	53.7282011004	27.1142854438
6	65.5165575721	17.0406053124	15.7815019554	57.4194579169	138.43489681	50.2731709417
7	55.3588268682	16.780307963	16.5841646531	75.6410018834	47.5944048007	28.0681605601
8	54.6439352519	17.6584519859	17.0797739827	80.7421944912	27.7337192325	18.6695277152
9	62.1799637644	16.1913895977	15.892886139	58.8581361381	342.077775481	32.407908827
10	54.2693780177	13.4256912568	13.2804573566	102.231247534	75.9971396011	52.0015285506
11	62.2595665841	15.8160994494	15.1230550605	86.3623966795	29.6439787773	58.2777057111
12	60.2843606249	14.7899074557	14.2703041781	48.357913035	79.7909631268	33.5181606162
13	63.2651920301	13.5241617939	13.3663998961	53.681043548	245.88695962	49.3798234136
14	55.8103948422	16.0677328294	15.5578083558	90.6429254352	116.829116646	62.0802332363
15	45.8872186824	7.96097566712	7.7830831329	120.024573066	102.830744793	103.525379024
16	59.139425774	10.700077422	10.4703925557	84.7003440117	507.801818338	80.0407382824
17	65.8135992766	14.3060025597	14.0814594646	51.1345452402	73.3125213211	39.0982468252
18	56.780939942	16.4046917857	16.1771940462	89.4773158001	100.697687488	19.2541349743
19	56.4175529102	13.1946751264	12.74837702	95.239291694	79.7129789852	41.4471018701

Cuadro 2: MSE obtenidos en cada dataset con **Gradiente Descendente Online**

2c) Newton Raphson

DataSet	Training MSE raw data	Training MSE rescaled data	Training MSE normalized data	Testing MSE raw data	Testing MSE rescaled data	Testing MSE normalized data
0	12.690235978	12.690235978	12.690235978	29.8692128464	70.8230012379	39.1167896725
1	15.9947888908	15.9947888908	15.9947888908	19.1213963039	105.300112485	67.378076989
2	17.7637059001	17.7637059001	17.7637059001	13.1702299683	139.177707921	18.9190672901
3	13.9043524941	13.9043524941	13.9043524941	28.1945882457	191.26333343	49.6104454505
4	15.3443534674	15.3443534674	15.3443534674	18.5845078343	60.566792667	24.2427186032
5	13.3367816832	13.3367816832	13.3367816832	29.3915243362	58.7330478202	26.930481888
6	15.7714352077	15.7714352077	15.7714352077	19.0828661051	183.563542547	50.2499676832
7	16.5377588238	16.5377588238	16.5377588238	16.3703204049	50.4205344035	28.4870497309
8	17.0315091906	17.0315091906	17.0315091906	16.1490495786	30.3017645531	18.8447579043
9	15.8241747153	15.8241747153	15.8241747153	21.9579912403	363.7143535	34.6682484071
10	13.2459812442	13.2459812442	13.2459812442	31.6607303987	77.7453577289	51.7167752009
11	15.0861602358	15.0861602358	15.0861602358	30.0689055712	30.5014191423	57.8116748334
12	14.2247910054	14.2247910054	14.2247910054	29.1616123114	85.4768097109	33.1587602379
13	13.3574116939	13.3574116939	13.3574116939	29.9555390274	245.72965064	49.7847446069
14	15.5168045543	15.5168045543	15.5168045543	22.256882111	153.683962628	61.1820073443
15	7.74295071369	7.74295071369	7.74295071369	86.5926285733	119.511903868	108.660106191
16	10.4584851241	10.4584851241	10.4584851241	40.2690867836	609.941477727	81.8616628828
17	14.0324636777	14.0324636777	14.0324636777	27.1971535112	76.3009933269	40.5837491905
18	16.1663277544	16.1663277544	16.1663277544	18.400069049	120.393356486	19.3481384185
19	12.7383399098	12.7383399098	12.7383399098	30.4741332593	90.2870113388	41.4018142274

Cuadro 3: MSE obtenidos en cada dataset con **Newton Raphson**

3) Locally weighted linear regression

In [49]:

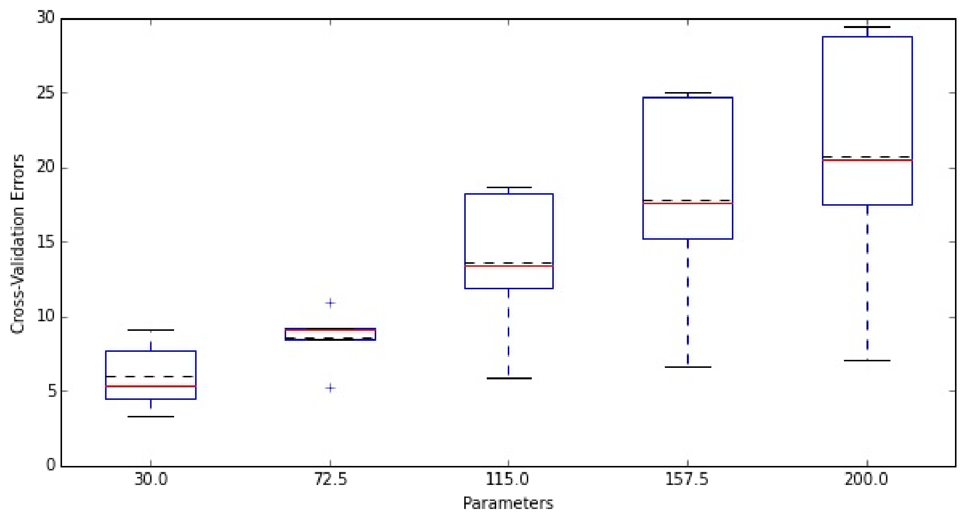
```
taus1 = np.linspace(30.,200.,5, endpoint=True)
taus2 = np.linspace(1.,10.,5, endpoint=True)
```

Raw data

In [57]:

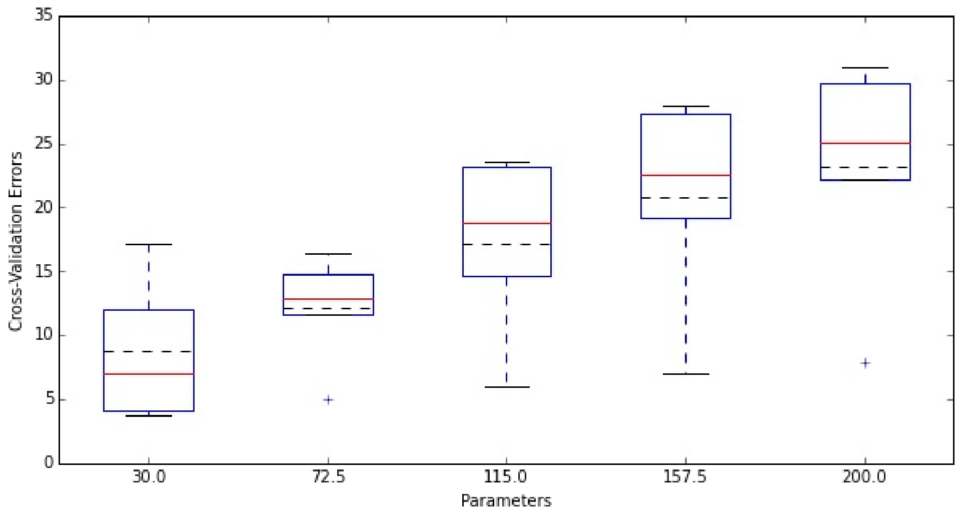
```
solve_weighted(taus1, show=[0,14])
```

```
##############################################################
Dataset: 0
Best tau: 30.0
```

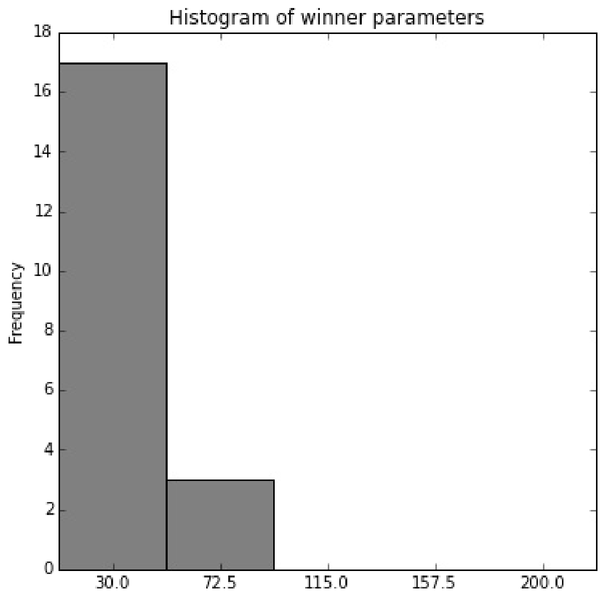


```
Training error (weighted): 0.185160073025
Testing error (weighted): 9.27828692439
Training error: 240.503072277
Testing error: 332.258498635
##############################################################


##############################################################
Dataset: 14
Best tau: 30.0
```



```
Training error (weighted): 0.26539563072
Testing error (weighted): 2.51726674875
Training error: 99.4681361427
Testing error: 136.928680467
##############################################################
```

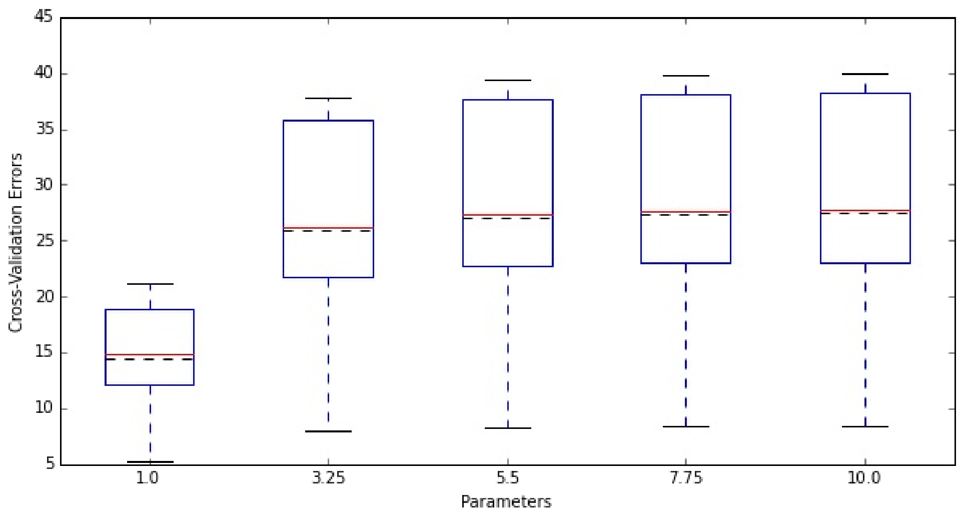


Rescaled data

In [58]:

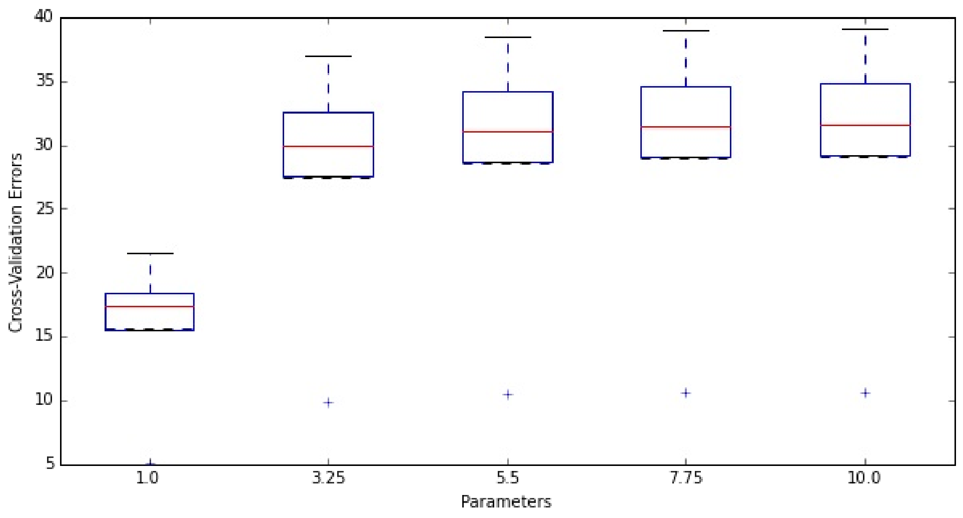
```
solve_weighted(taus2, data_func=rescale, show=[0,14])
```

```
##############################################################
Dataset: 0
Best tau: 1.0
```

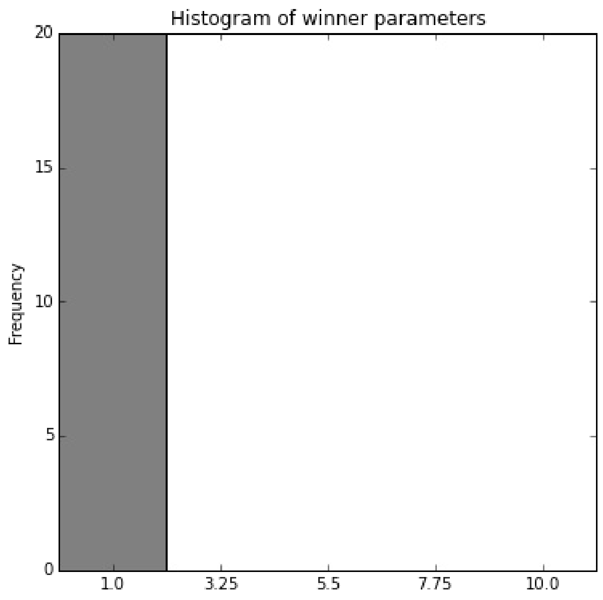


```
Training error (weighted): 5.8374635318
Testing error (weighted): 34.5095818121
Training error: 12.9492812413
Testing error: 70.2835184179
##############################################################


##############################################################
Dataset: 14
Best tau: 1.0
```



```
Training error (weighted): 6.62249985344
Testing error (weighted): 66.2273010366
Training error: 16.0705146463
Testing error: 159.675444498
##############################################################
```

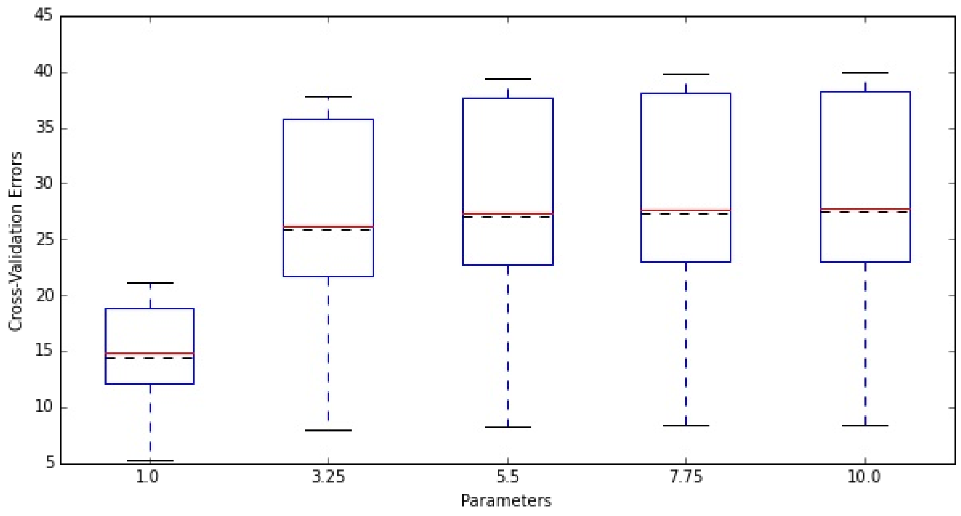


Normalized data

```
In [59]:
```

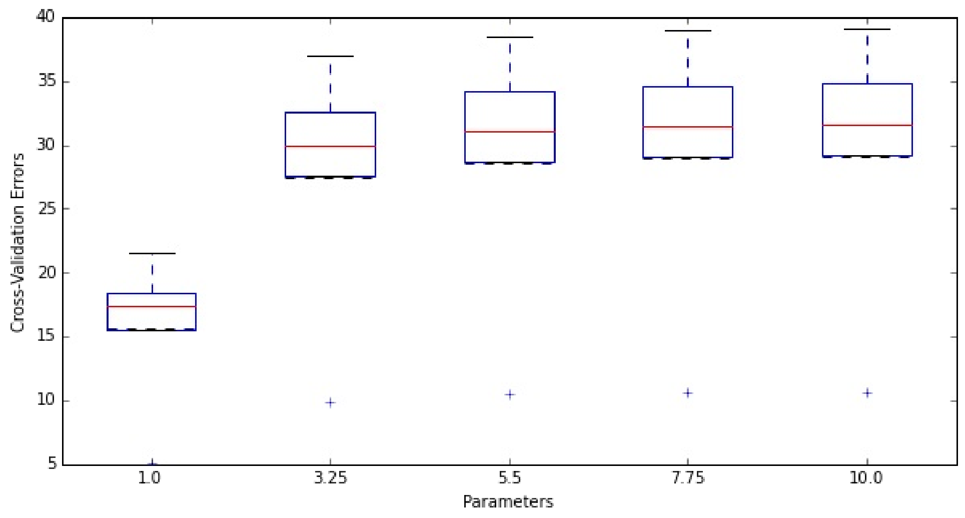
```
solve_weighted(taus2, data_func=rescale, show=[0,14])
```

```
##############################################################
Dataset: 0
Best tau: 1.0
```

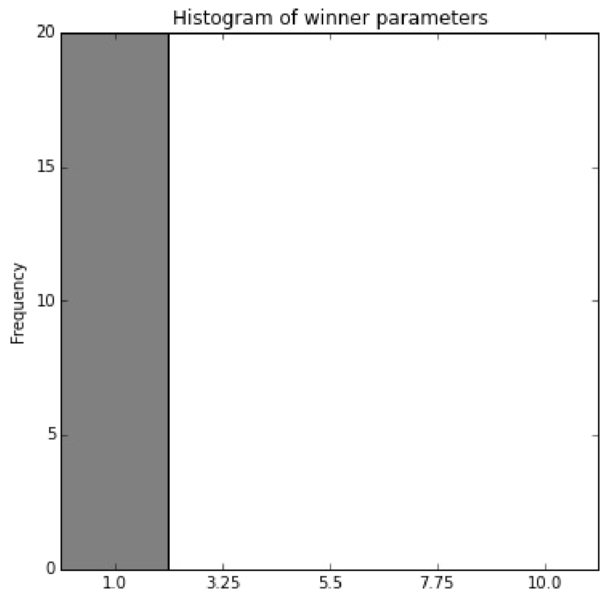


```
Training error (weighted): 5.8374635318
Testing error (weighted): 34.5095818121
Training error: 12.9492812413
Testing error: 70.2835184179
##############################################################


##############################################################
Dataset: 14
Best tau: 1.0
```



```
Training error (weighted): 6.62249985344
Testing error (weighted): 66.2273010366
Training error: 16.0705146463
Testing error: 159.675444498
##############################################################
```



4)

DataSet	Mean wMSE (tr) raw data	Mean wMSE (tr) rescaled data	Mean wMSE (tr) normalized data	Mean wMSE (ts) raw data	Mean wMSE (ts) rescaled data	Mean wMSE (ts) normalized data
0	0.185160073025	5.8374635318	0.0116268353211	9.27828692439	34.5095818121	2.7189137064
1	0.551491481187	7.42357667352	0.0140799530963	3.05010877493	45.8455651983	3.3947079304
2	4.22787452513	8.48216750479	0.0226595699254	4.11617089859	61.9495888587	1.53273149511
3	0.191190178173	6.41636593947	0.0164662870107	7.20283101934	80.2978274427	3.23027756977
4	0.286859127666	6.69590718878	0.00958848273139	5.87301786989	29.8022620265	2.69509073311
5	0.226068459737	5.99605881163	0.00721053354812	4.49815058632	28.8133003779	3.24257416177
6	0.228083553709	7.67975166864	0.0168670776746	10.6024610579	84.2729908046	3.50642360936
7	0.34233665971	7.12120761645	0.0114984678544	2.44973336275	22.0677520929	1.68708756991
8	0.30614464963	7.60106315371	0.0101669385035	8.8393949496	16.8947768907	1.98017305453
9	0.356655968295	6.90782752004	0.0116325674784	2.64749173719	168.900834481	1.8423408332
10	0.186997085423	6.06839289636	0.021925671126	8.51090362379	33.0346360177	4.03371732978
11	0.276747062181	7.54961200333	0.0181817424015	4.58154236743	15.0643647309	2.04514230794
12	0.384425821692	6.6105341413	0.0113366594152	3.9540924914	35.5408277928	1.23149062131
13	0.384997960811	5.04906381141	0.0169705085118	5.71929159871	115.979760351	3.13274681413
14	0.26539563072	6.62249985344	0.00717152063922	2.51726674875	66.2273010366	2.71192505401
15	1.4738023321	3.53142047	0.0097002934449	30.8945551067	51.3328189029	5.85392884421
16	0.257585475846	4.32100222221	0.0047222240917	7.64154951211	236.272101869	4.71691292037
17	0.199371376596	6.2636251599	0.00905094522373	6.73644640536	29.8133605895	2.98543515937
18	3.57537468158	7.01610995628	0.0185609581019	6.17709649767	50.9331564681	1.58109126714
19	0.299194740568	5.79724233979	0.023799033615	5.02264510199	37.2626320223	2.43789003229

Cuadro 4: Mean wMSE (weighted Mean Squared Error) en cada dataset con **Locally Weighted Linear Regression**

5)

Parte 2 - Regresión Logística

1a) Gradiente ascendente online

In [61]:

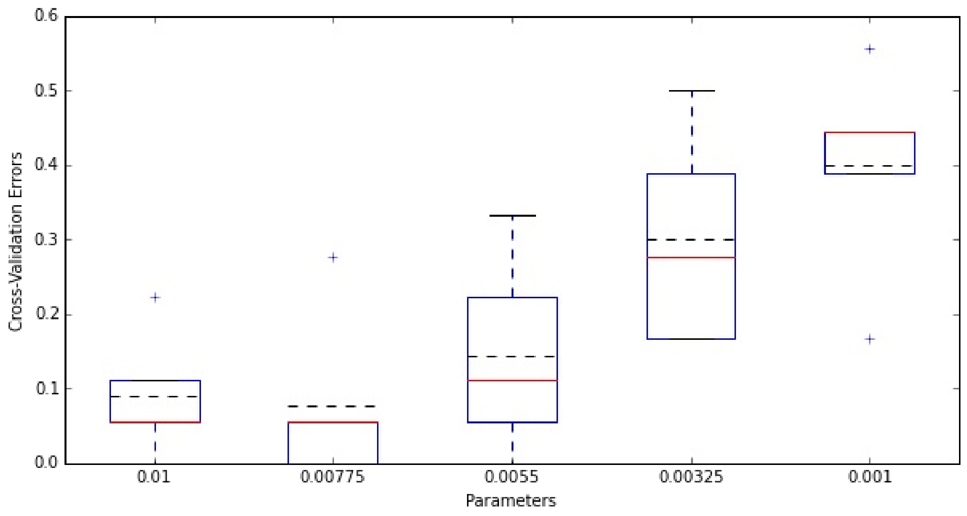
```
#alphas to try on ascent gradient stochastic
alphas3 = np.linspace(1e-2, 1e-3, 5, endpoint=True)
```

Raw data

In [73]:

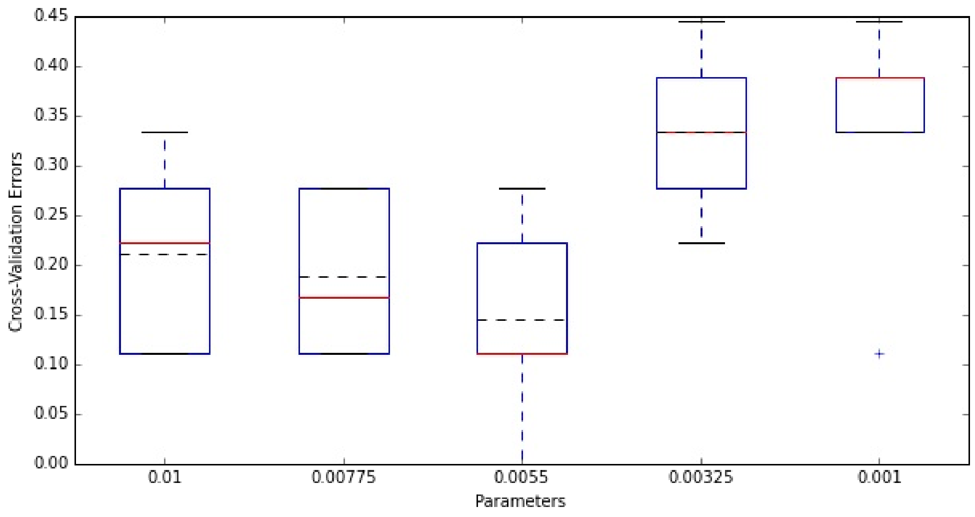
```
solve_regression(gd_stochastic, 'logistic', params=alphas3, show=[0,14])
```

```
################################################################
Dataset: 0
Best alpha: 0.00775
```

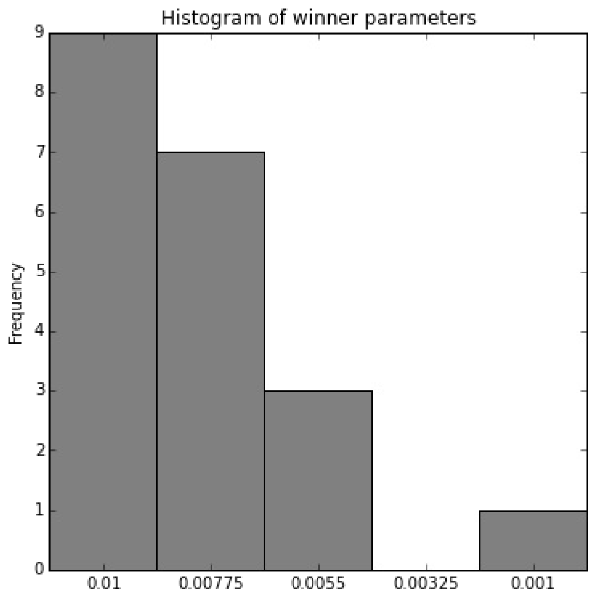


```
Training error: 0.0444444444444
Testing error: 0.0666666666667
N° iterations: 13625
Beta: [ -6.07258811e+00   7.03329870e+00  -8.26724873e+00  -2.25430210e-01
   1.24680827e+02  -2.80636719e+02   3.07024050e+00]
##############################################################


##############################################################
Dataset: 14
Best alpha: 0.0055
```



```
Training error: 0.155555555556
Testing error: 0.166666666667
N° iterations: 100000
Beta: [  85.582449     5.51768624   -6.92887744    0.90383805  137.34901635
 -394.48396529    0.9807746 ]
##############################################################
```

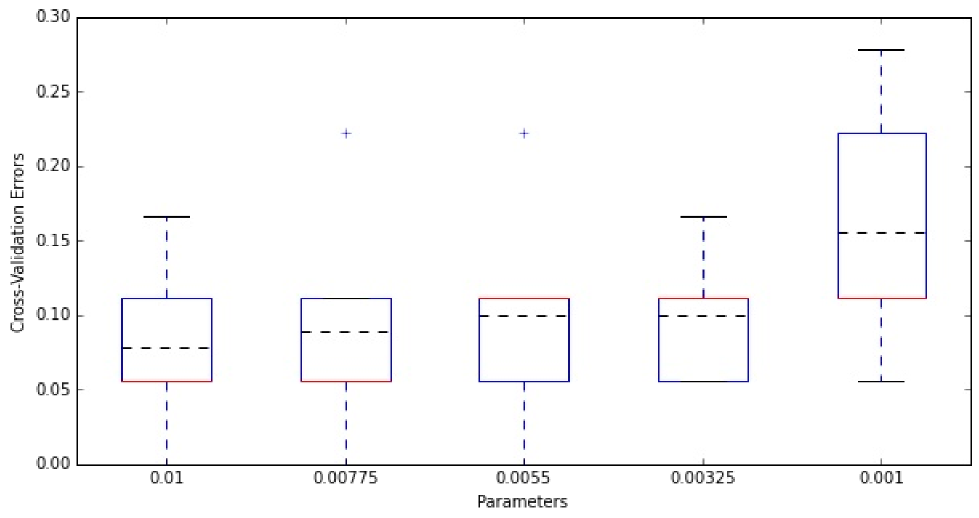


Rescaled data

In [71]:

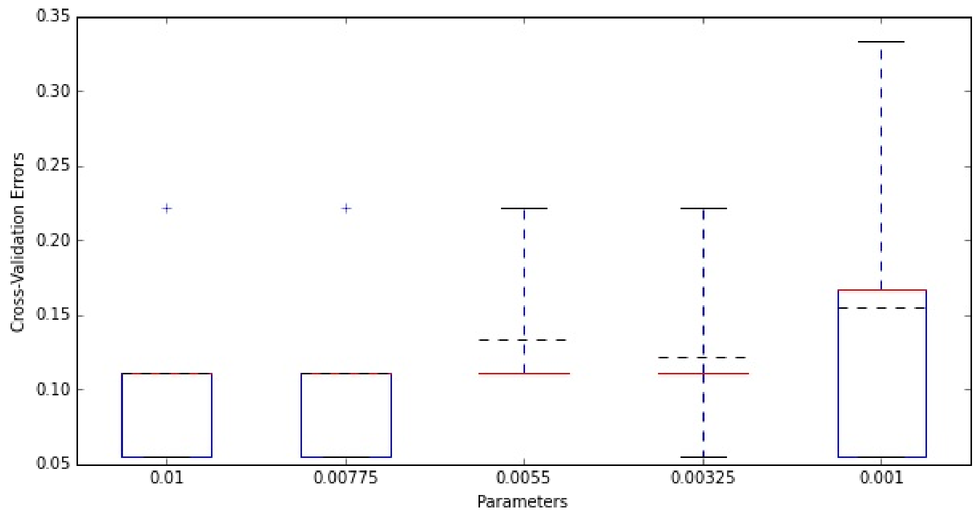
```
solve_regression(gd_stochastic, 'logistic', params=alphas3, data_func=rescale, show=[0,14])
```

```
##############################################################
Dataset: 0
Best alpha: 0.01
```

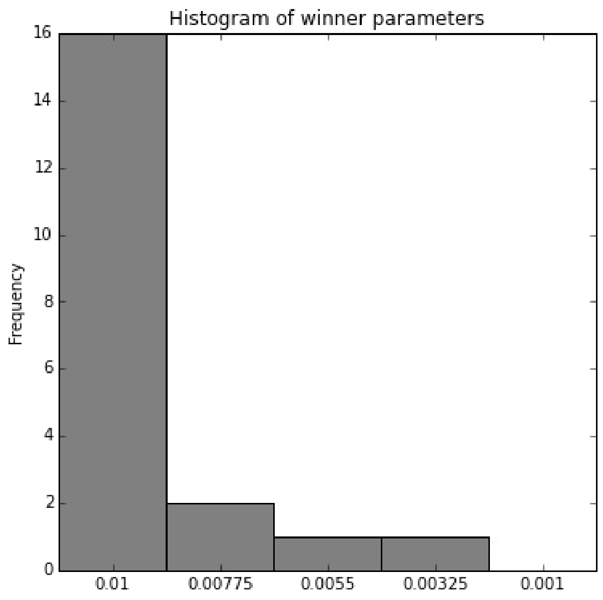


```
Training error: 0.0666666666667
Testing error: 0.1
N° iterations: 364
Beta: [-0.89067992  1.89240682 -3.06533656 -0.49413702  8.38805164 -3.67234941
  1.39561496]
##############################################################


##############################################################
Dataset: 14
Best alpha: 0.01
```



```
Training error: 0.0888888888889
Testing error: 0.1
N° iterations: 341
Beta: [-0.86739457  2.20861945 -2.52849556 -0.58885793  8.37423558 -3.46780645
  0.54455779]
##############################################################
```

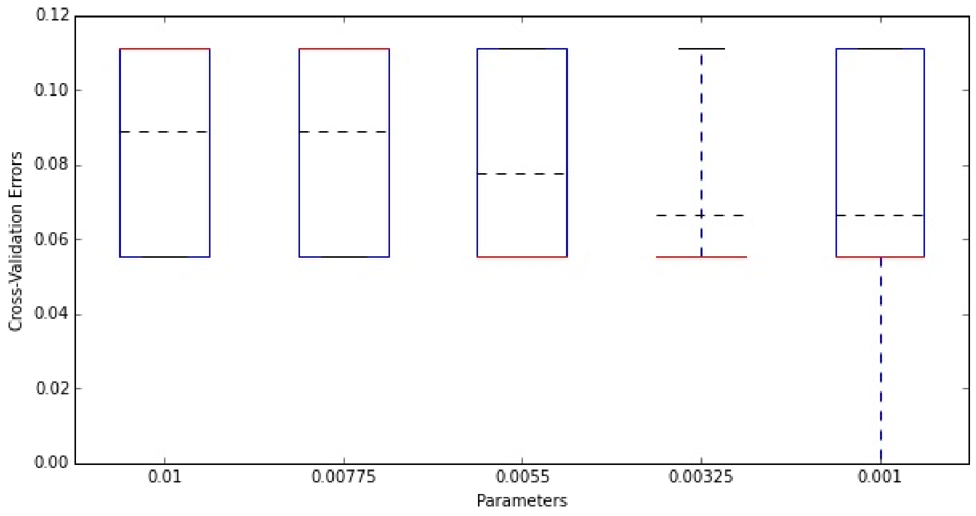


Normalized data

In [72]:

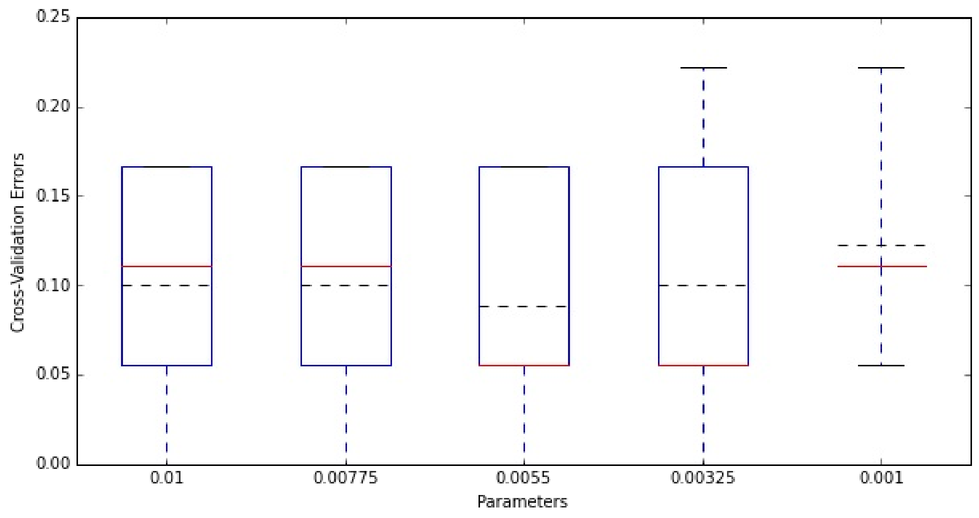
```
solve_regression(gd_stochastic, 'logistic', params=alphas3, data_func=normalize, show=[0,14])
```

```
##############################################################
Dataset: 0
Best alpha: 0.00325
```

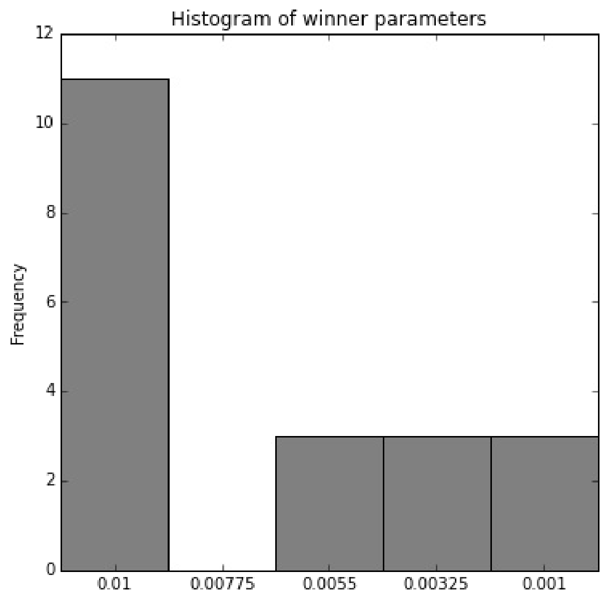


```
Training error: 0.0222222222222
Testing error: 0.0333333333333
N° iterations: 345
Beta: [-0.2385682   0.81850858 -1.27981156 -0.11349321  3.88872458 -2.76731914
  0.40282272]
##############################################################


##############################################################
Dataset: 14
Best alpha: 0.0055
```



```
Training error: 0.0444444444444
Testing error: 0.0666666666667
N° iterations: 290
Beta: [ 0.20829953  1.02332025 -1.47941872 -0.31923802  4.65487619 -3.11664099
  0.09574176]
################################################################
```



1b) Newton Raphson

Raw data

In [67]:

```
solve_regression(nr_logistic, 'logistic', data_func=normalize, show=[0,14])
```

```
##############################################################
Dataset: 0
Training error: 0.0
Testing error: 0.0333333333333
N° iterations: 17
Beta: [ -10.17632429   51.90495222 -223.06462833  -72.5822921   446.92953071
 -290.39247559    5.29862312]
##############################################################


##############################################################
Dataset: 14
Training error: 0.0222222222222
Testing error: 0.0333333333333
N° iterations: 14
Beta: [   6.21575176   18.14870684 -124.70691036  -40.65161461  235.09789749
 -145.40751338   -3.76500934]
##############################################################
```

Rescaled data

In [69]:

```
solve_regression(nr_logistic, 'logistic', data_func=rescale, show=[0,14])
```

```
##############################################################
Dataset: 0
Training error: 0.0
Testing error: 0.1
N° iterations: 17
Beta: [   73.98701993   180.88595375  -814.87368456  -183.34781483  1865.9231994
  -602.60602669    22.16612173]
##############################################################


##############################################################
Dataset: 14
Training error: 0.0222222222222
Testing error: 0.133333333333
N° iterations: 14
Beta: [  66.60527006   62.6697612  -440.52106091  -98.52834405  948.8637574
 -296.81184378  -17.07077338]
##############################################################
```

Normalized data

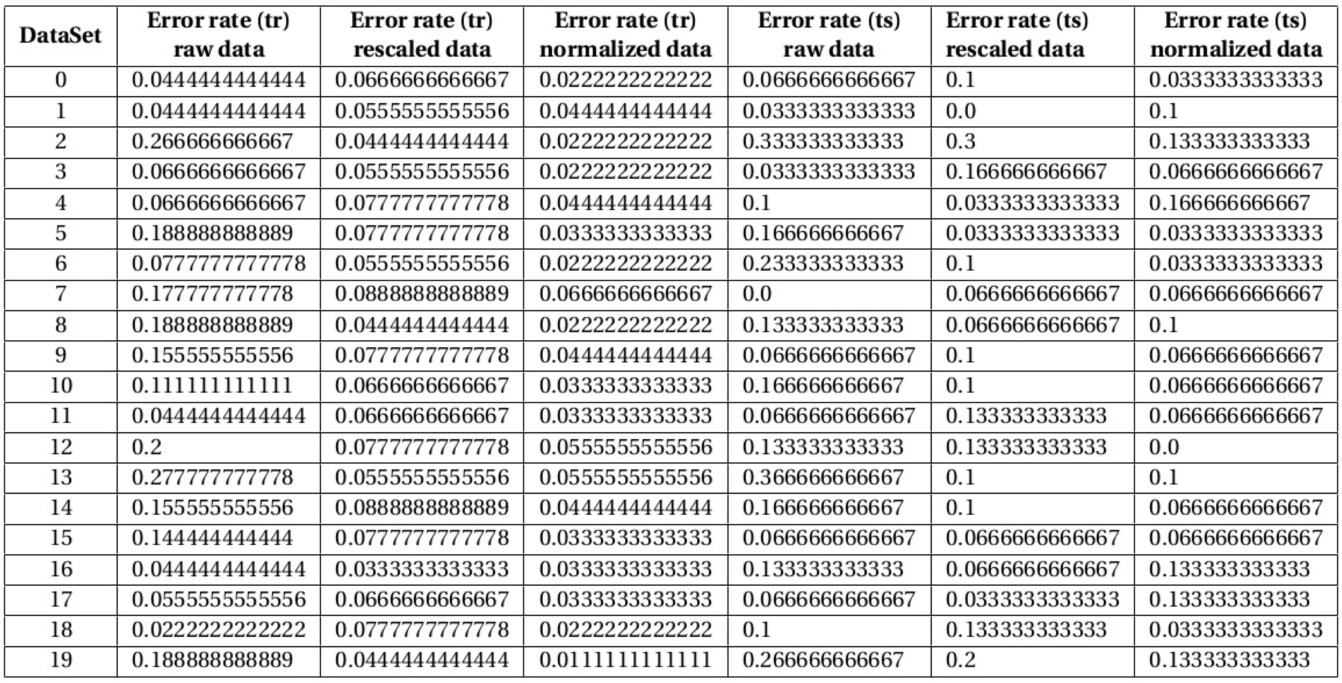
```
In [70]:
```

```
solve_regression(nr_logistic, 'logistic', data_func=normalize, show=[0,14])
```

```
############################################################
Dataset: 0
Training error: 0.0
Testing error: 0.0333333333333
N° iterations: 17
Beta: [ -10.17632429   51.90495222 -223.06462833  -72.5822921   446.92953071
 -290.39247559    5.29862312]
############################################################


############################################################
Dataset: 14
Training error: 0.0222222222222
Testing error: 0.0333333333333
N° iterations: 14
Beta: [   6.21575176   18.14870684 -124.70691036  -40.65161461  235.09789749
 -145.40751338   -3.76500934]
############################################################
```

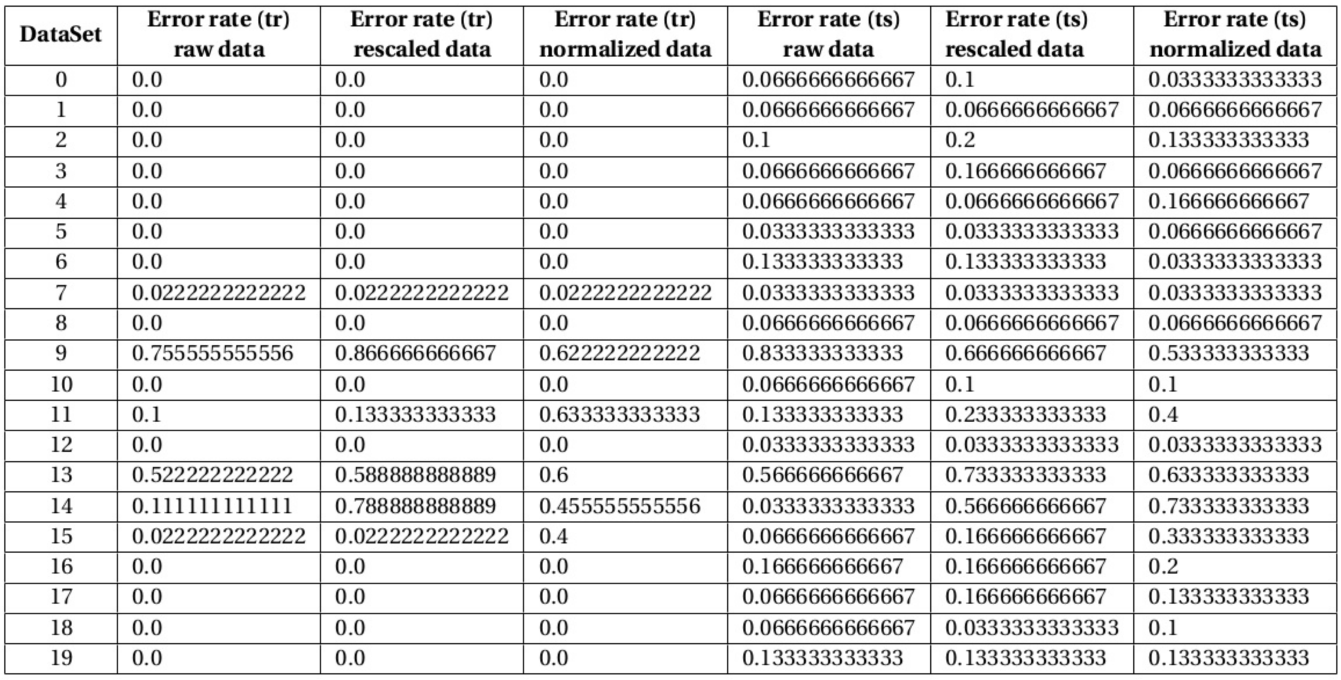
2a) Gradiente Ascendente Online



DataSet	Error rate (tr) raw data	Error rate (tr) rescaled data	Error rate (tr) normalized data	Error rate (ts) raw data	Error rate (ts) rescaled data	Error rate (ts) normalized data
0	0.0444444444444	0.0666666666667	0.0222222222222	0.0666666666667	0.1	0.0333333333333
1	0.0444444444444	0.0555555555556	0.0444444444444	0.0333333333333	0.0	0.1
2	0.266666666667	0.0444444444444	0.0222222222222	0.333333333333	0.3	0.133333333333
3	0.0666666666667	0.0555555555556	0.0222222222222	0.0333333333333	0.166666666667	0.0666666666667
4	0.0666666666667	0.0777777777778	0.0444444444444	0.1	0.0333333333333	0.166666666667
5	0.188888888889	0.0777777777778	0.0333333333333	0.166666666667	0.0333333333333	0.0333333333333
6	0.0777777777778	0.0555555555556	0.0222222222222	0.233333333333	0.1	0.0333333333333
7	0.177777777778	0.0888888888889	0.0666666666667	0.0	0.0666666666667	0.0666666666667
8	0.188888888889	0.0444444444444	0.0222222222222	0.133333333333	0.0666666666667	0.1
9	0.155555555556	0.0777777777778	0.0444444444444	0.0666666666667	0.1	0.0666666666667
10	0.111111111111	0.0666666666667	0.0333333333333	0.166666666667	0.1	0.0666666666667
11	0.0444444444444	0.0666666666667	0.0333333333333	0.0666666666667	0.133333333333	0.0666666666667
12	0.2	0.0777777777778	0.0555555555556	0.133333333333	0.133333333333	0.0
13	0.277777777778	0.0555555555556	0.0555555555556	0.366666666667	0.1	0.1
14	0.155555555556	0.0888888888889	0.0444444444444	0.166666666667	0.1	0.0666666666667
15	0.144444444444	0.0777777777778	0.0333333333333	0.0666666666667	0.0666666666667	0.0666666666667
16	0.0444444444444	0.0333333333333	0.0333333333333	0.133333333333	0.0666666666667	0.133333333333
17	0.0555555555556	0.0666666666667	0.0333333333333	0.0666666666667	0.0333333333333	0.133333333333
18	0.0222222222222	0.0777777777778	0.0222222222222	0.1	0.133333333333	0.0333333333333
19	0.188888888889	0.0444444444444	0.0111111111111	0.266666666667	0.2	0.133333333333

Cuadro 5: Error rate en cada dataset obtenido con **Gradiente Ascendente Online** para regresión logística

2b) Newton Raphson



DataSet	Error rate (tr) raw data	Error rate (tr) rescaled data	Error rate (tr) normalized data	Error rate (ts) raw data	Error rate (ts) rescaled data	Error rate (ts) normalized data
0	0.0	0.0	0.0	0.0666666666667	0.1	0.0333333333333
1	0.0	0.0	0.0	0.0666666666667	0.0666666666667	0.0666666666667
2	0.0	0.0	0.0	0.1	0.2	0.133333333333
3	0.0	0.0	0.0	0.0666666666667	0.166666666667	0.0666666666667
4	0.0	0.0	0.0	0.0666666666667	0.0666666666667	0.166666666667
5	0.0	0.0	0.0	0.0333333333333	0.0333333333333	0.0666666666667
6	0.0	0.0	0.0	0.133333333333	0.133333333333	0.0333333333333
7	0.0222222222222	0.0222222222222	0.0222222222222	0.0333333333333	0.0333333333333	0.0333333333333
8	0.0	0.0	0.0	0.0666666666667	0.0666666666667	0.0666666666667
9	0.755555555556	0.866666666667	0.622222222222	0.833333333333	0.666666666667	0.533333333333
10	0.0	0.0	0.0	0.0666666666667	0.1	0.1
11	0.1	0.133333333333	0.633333333333	0.133333333333	0.233333333333	0.4
12	0.0	0.0	0.0	0.0333333333333	0.0333333333333	0.0333333333333
13	0.522222222222	0.588888888889	0.6	0.566666666667	0.733333333333	0.633333333333
14	0.111111111111	0.788888888889	0.455555555556	0.0333333333333	0.566666666667	0.733333333333
15	0.0222222222222	0.0222222222222	0.4	0.0666666666667	0.166666666667	0.333333333333
16	0.0	0.0	0.0	0.166666666667	0.166666666667	0.2
17	0.0	0.0	0.0	0.0666666666667	0.166666666667	0.133333333333
18	0.0	0.0	0.0	0.0666666666667	0.0333333333333	0.1
19	0.0	0.0	0.0	0.133333333333	0.133333333333	0.133333333333

Cuadro 6: Error rate en cada dataset obtenido con **Newton Raphson** para regresión logística

3)

Conclusiones

Anexos

En la siguiente sección se encuentra todo el código necesario para reproducir cada uno de los resultados mostrados anteriormente. Para poder ejecutar el código en el informe, se debe en primer lugar ejecutar las celdas de código presentes en este anexo.

Configuración del notebook

In [17]:

```
#notebook settings
%matplotlib inline

#import some useful libraries and utilities
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cross_validation import KFold
from sklearn.cross_validation import KFold

#avoid numpy warning (they are handled correctly)
np.seterr('ignore')

#setting some paths
path1 = './cereales/'
#data directory
path2 = './credit/'
```

Métricas de error para regresión lineal

In [18]:

```
#overall cost function for linear regresion
def J(X, y, beta):
    f = np.dot(X,beta)
    diff = f-y
    return 0.5*np.dot(diff,diff)

#mean squared error for linear regression
def mse(X, y, beta):
    M,_ = X.shape
    f = np.dot(X,beta)
    diff = f-y
    return (1./(M-1))*np.dot(diff,diff)
```

Implementación de algoritmos de regresión lineal

In [32]:

```
#batch gradient descent for linear regression
def gd_batch(X, y, alpha, eps=1e-3, max_iter=100000):
    M,N = X.shape
    beta = np.zeros(N)
    J1 = J(X,y,beta)
    for i in xrange(max_iter):
        J0 = J1
        f = np.dot(X,beta)
        dJ = np.dot(X.T,f-y)
        beta -= alpha*dJ
        J1 = J(X,y,beta)
        if np.abs(J1-J0)/J0 < eps:
            break
    return (beta,i+1)

#online gradient descent for linear regression
def gd_online(X, y, alpha, eps=1e-3, max_iter=100000):
    M,N = X.shape
    beta = np.zeros(N)
    J1 = J(X,y,beta)
    for i in xrange(max_iter):
        J0 = J1
        for m in xrange(M):
            beta -= alpha*(np.dot(X[m],beta)-y[m])*X[m]
        J1 = J(X,y,beta)
        if np.abs(J1-J0)/J0 < eps: break
    return (beta,i+1)

#Newton-Raphson method for linear regression
def nr_linear(X, y, eps=1e-5, max_iter=100000):
    M,N = X.shape
    beta = np.zeros(N)
    J1 = J(X,y,beta)
    Hess = np.dot(X.T,X)
    for i in xrange(max_iter):
        J0 = J1
        f = np.dot(X,beta)
        dJ = np.dot(X.T,f-y)
        beta -= np.linalg.solve(Hess, dJ)
        J1 = J(X,y,beta)
        if np.abs(J1-J0)/J0 < eps: break
    return (beta,i+1)
```

Comentarios de implementación:

- Para todos los algoritmos existen básicamente dos criterios de salida. El primero es cuando el error relativo es menor a *eps*, vale decir, cuando la función de error esta cambiando muy poco de iteración en iteración. El segundo es el número máximo de iteraciones, mas que nada para detener algoritmos que no pueden cumplir con el criterio del error (learning rates muy altos por ejemplo).
- Todos los *starting guest* son el vector zeros. Esto para reproducir y comparar resultados de manera adecuada.
- En vez de invertir la matriz Hessiana en Newton-Raphson, se opta por resolver el sistema lineal asociado, por razones de estabilidad numérica.

Implementación de locally weighted linear regression

In [53]:

```
#compute weights for all samples in X matrix, respect to x0
def weight(X, x0, tau):
    Diff = X - x0
    Diff *= Diff
    return np.exp(-1*np.sum(Diff,axis=1)/(2.*tau**2))

#weighted cost function
def wJ(X, y, beta, w):
    f = np.dot(X,beta)
    diff = f-y
    diff **=2
    return 0.5*np.dot(w,diff)

#weighted mean squared error
def wmse(X, y, beta, w):
    M,_ = X.shape
    f = np.dot(X,beta)
    diff = f-y
    diff **=2
    return (1./(M-1))*np.dot(w,diff)

#find best beta for locally weighted linear regression
def min_weighted(X, y, w):
    W = np.diag(w)
    M = np.dot(X.T, np.dot(W, X))
    b = np.dot(X.T, np.dot(W, y))
    return np.linalg.solve(M,b)
```

Métricas de error para regresión logística

In [20]:

```
#log likelihood function for logistic regression
"""
Computing l this way, make it more stable numerically (no overflows en exp)
"""
def l(X, y, beta):
    y1_mask = y.astype(bool)
    y0_mask = np.logical_not(y1_mask)
    f = sigmoid(np.dot(X,beta))
    return (np.log(f[y1_mask])).sum() + (np.log(1-f[y0_mask])).sum()


#error rate for logistic regression
def error_rate(X, y, beta):
    h = np.round(sigmoid(np.dot(X,beta)))
    h = h.astype(int)
    y = y.astype(int)
    m, = h.shape
    return np.logical_xor(h,y).sum()/np.float(m)
```

Implementación de algoritmo de regresión logística

In [66]:

```
#sigmoid function
def sigmoid(z):
    return 1./(1.+np.exp(-z))

#stochastic gradient ascent for logistic regression
def gd_stochastic(X, y, alpha, eps=1e-3, max_iter=100000):
    M,N = X.shape
    beta = np.zeros(N)
    l1 = l(X, y, beta)+1.
    for i in xrange(max_iter):
        l0 = l1
        for m in xrange(M):
            beta += alpha*(y[m]-sigmoid(np.dot(X[m],beta)))*X[m]
        l1 = l(X,y,beta)+1.
        if np.abs(l1-l0)/np.abs(l0) < eps: break
    return (beta,i+1)

#Newton-Raphson method for logistic regression
def nr_logistic(X, y, eps=1e-3, max_iter=100000):
    M,N = X.shape
    beta = np.zeros(N)
    l1 = l(X, y, beta)+1.
    for i in xrange(max_iter):
        l0 = l1
        f = sigmoid(np.dot(X,beta))
        W = np.diag(f*(1-f))
        Hess = -1*np.dot(X.T, np.dot(W, X))
        Dl = np.dot(X.T, y-f)
        #when it converges, Hess became singular
        try:
            beta -= np.linalg.solve(Hess, Dl)
        except np.linalg.LinAlgError:
            break
        l1 = l(X, y, beta)+1.
        if np.abs(l1-l0)/np.abs(l0) < eps: break
    return (beta,i+1)
```

Comentarios de implementación:

- Para ambos algoritmos hay dos criterios de salida. El primero es cuando la función *log verosimilitud* cambia relativamente menor a *eps* en cada iteración (pues es la función que se quiere maximizar). Se tiene en cuenta además que en el óptimo esta función debe ser 0 (en el óptimo la función de *verosimilitud* es 1, pues maximiza la probabilidad para cada dato), por lo que se le suma un 1 para evitar problemas al computar el criterio de salida. El segundo criterio el número máximo de iteraciones.
- Existe un tercer criterio de salida en el método de *Newton-Raphson*. A medida que converge, el vector f con las probabilidades de pertenecer a la clase 1 de todos los datos, tiene sólo valores cercanos a 0 y 1. Luego al computar la matriz W, esta empezará a tener filas completas de 0 o valores muy cercanos a 0, y por lo tanto la matriz Hessiana también, y al converger esta matriz se vuelve singular. Para eso se ocupa el manejo de la excepción en caso de existir singularidad.

Funciones para manejo de la data

In [22]:

```
#Rescale features of M to [a,b] range
def rescale(M, a=0., b=1.):
    #max and min vectors
    maxv = np.max(M, axis=0)
    minv = np.min(M, axis=0)
    return (b-a)*M/(maxv-minv) + (a*maxv-b*minv)/(maxv-minv)

#Normalize features of M
def normalize(M):
    #mean and standard deviation vectors
    meanv = np.mean(M, axis=0)
    stdv = np.std(M, axis=0)
    return (M-meanv)/stdv
```

Funciones para Cross-Validation

In [23]:

```
""" find the best learning parameter for algorithm, between
parameters in params using 5-fold cross validation """
def cross_alpha(X, y, algorithm, error_func, params):
    #creating kfold
    m,n = X.shape
    kf = KFold(m, n_folds=5)
    cv_err = np.empty((5,5))
    i = 0 #index of fold

    for tr_index,ts_index in kf:
        j = 0 #index of parameter
        X_tr, X_ts = X[tr_index], X[ts_index]
        y_tr, y_ts = y[tr_index], y[ts_index]
        for param in params:
            beta,_ = algorithm(X_tr, y_tr, alpha=param)
            cv_err[i,j] = error_func(X_ts, y_ts, beta)
            j += 1
        i += 1

    #arrays with mean cv-error for each alpha
    cv_mean = np.mean(cv_err, axis=0)
    return params[np.argmin(cv_mean)], cv_err

""" find the best band width parameter for locally
weighted linear regression, between parameters in params
using 5-fold cross validation """
def cross_tau(X, y, params):
    #creating kfolds
    m,n = X.shape
    kf = KFold(m, n_folds=5)
    cv_err = np.zeros((5,5))
    i = 0 #index of fold

    for tr_index,ts_index in kf:
        X_tr, X_ts = X[tr_index], X[ts_index]
        y_tr, y_ts = y[tr_index], y[ts_index]
        j = 0 #index of parameter
        for tau in params:
            for x0 in X_ts:
                w1 = weight(X_tr, x0, tau)
                w2 = weight(X_ts, x0, tau)
                beta = min_weighted(X_tr, y_tr, w1)
                cv_err[i,j] += wmse(X_ts, y_ts, beta, w2)
            cv_err[i,j] /= X_ts.shape[0]
            j +=1
        i +=1

    #arrays with mean costs for each alpha
    cv_mean = np.mean(cv_err, axis=0)
    return params[np.argmin(cv_mean)], cv_err
```

Funciones complementarias (Helpers) para obtener resultados

In [56]:

```
"""
Function to generate histogram of winners
"""
def make_hist(winners,params):
    winners = np.array(winners)
    freqs = np.zeros(5)
    for i in xrange(5):
        freqs[i] = np.sum(params[i]==winners)

    labels = map(str,params)
    pos = np.arange(len(labels))
    width = 1.0
    fig = plt.figure()
    fig.set_figheight(6)
    fig.set_figwidth(6)
    ax = plt.axes()

    ax.set_xticks(pos + (width / 2))
    ax.set_xticklabels(labels)
    plt.ylabel('Frequency')
    plt.title('Histogram of winner parameters')
    plt.bar(pos, freqs, width, color='0.5')
    plt.show()

"""
Generate solutions for regression problems
(linear and logistic)
"""
def solve_regression(algorithm, kind, params=None, data_func=None, show=None):
    if params is not None:
        winners = list()

    if kind=='linear':
        path = path1+'cereales'
        error_func = mse
    elif kind=='logistic':
        path = path2+'credit'
        error_func = error_rate
    else:
        print "Unknown kind!"
        return -1

    for i in xrange(20):
        #Loading dataset
        tr_file = path+'-tr-{0}.npy'.format(i)
        ts_file = path+'-ts-{0}.npy'.format(i)
        tr_data = np.load(tr_file)
        ts_data = np.load(ts_file)

        if data_func is not None:
            X_tr = data_func(tr_data[:,:-1])
        else:
            X_tr = tr_data[:,:-1]
        y_tr = np.ascontiguousarray(tr_data[:,-1])
        #Adding column of 1's
        m,n = X_tr.shape
        X_tr = np.concatenate((np.ones((m,1)),X_tr),axis=1)

        if data_func is not None:
            X_ts = data_func(ts_data[:,:-1])
        else:
            X_ts = ts_data[:,:-1]
        y_ts = np.ascontiguousarray(ts_data[:,-1])
        #Adding column of 1's
        m,n = X_ts.shape
        X_ts = np.concatenate((np.ones((m,1)),X_ts),axis=1)

        if params is not None:
            alpha,cv_err = cross_alpha(X_tr, y_tr, algorithm, error_func, params)
            winners.append(alpha)
            beta,it = algorithm(X_tr, y_tr, alpha)
        else:
            beta,it = algorithm(X_tr, y_tr)

        if (show is not None) and (i not in show): continue
        print "################################################################"
        print "Dataset: {0}".format(i)
        if params is not None:
            print 'Best alpha: {0}'.format(alpha)
            fig = plt.figure()
            fig.set_figheight(5)
            fig.set_figwidth(10)
            plt.xlabel('Parameters')
            plt.ylabel('Cross-Validation Errors')
            plt.boxplot(cv_err, showmeans=True, meanline=True)
            plt.xticks([1, 2, 3, 4, 5], map(str,params))
            plt.show()
        print 'Training error: {0}'.format(error_func(X_tr,y_tr,beta))
        print 'Testing error: {0}'.format(error_func(X_ts,y_ts,beta))
        print 'N° iterations: {0}'.format(it)
        print 'Beta: {0}'.format(beta)
        print "################################################################"
        print '\n'


    if params is not None:
        make_hist(winners,params)

"""
Generate solutions for locally weighted linear regression problems
"""
def solve_weighted(params, data_func=None, show=None):
    #list with winners-alphas
    winners = list()

    for i in xrange(20):
        #Loading dataset
        tr_file = path1+'cereales-tr-{0}.npy'.format(i)
        ts_file = path1+'cereales-ts-{0}.npy'.format(i)
        tr_data = np.load(tr_file)
        ts_data = np.load(ts_file)

        if data_func is not None:
            X_tr = data_func(tr_data[:,:-1])
        else:
            X_tr = tr_data[:,:-1]
        y_tr = np.ascontiguousarray(tr_data[:,-1])
        #Adding column of 1's
        m,n = X_tr.shape
        X_tr = np.concatenate((np.ones((m,1)),X_tr),axis=1)

        if data_func is not None:
            X_ts = data_func(ts_data[:,:-1])
        else:
            X_ts = ts_data[:,:-1]
        y_ts = np.ascontiguousarray(ts_data[:,-1])
        #Adding column of 1's
        m,n = X_ts.shape
        X_ts = np.concatenate((np.ones((m,1)),X_ts),axis=1)


        tau,cv_err = cross_tau(X_tr, y_tr, params)
        winners.append(tau)
        wtr_err = 0
        wts_err = 0
        tr_err = 0
        ts_err = 0
        for x0 in X_ts:
            w1 = weight(X_tr, x0, tau)
            w2 = weight(X_ts, x0, tau)
            beta = min_weighted(X_tr, y_tr, w1)
            wtr_err += wmse(X_tr, y_tr, beta, w1)
            wts_err += wmse(X_ts, y_ts, beta, w2)
            tr_err += mse(X_tr, y_tr, beta)
            ts_err += mse(X_ts, y_ts, beta)
        M = X_ts.shape[0]
        wtr_err /= M
        wts_err /= M
        tr_err /= M
        ts_err /= M

        if (show is not None) and (i not in show): continue
        print "##############################################################"
        print "Dataset: {0}".format(i)
        print 'Best tau: {0}'.format(tau)
        fig = plt.figure()
        fig.set_figheight(5)
        fig.set_figwidth(10)
        plt.xlabel('Parameters')
        plt.ylabel('Cross-Validation Errors')
        plt.boxplot(cv_err, showmeans=True, meanline=True)
        plt.xticks([1, 2, 3, 4, 5], map(str,params))
        plt.show()
        print 'Training error (weighted): {0}'.format(wtr_err)
        print 'Testing error (weighted): {0}'.format(wts_err)
        print 'Training error: {0}'.format(tr_err)
        print 'Testing error: {0}'.format(ts_err)
        print "##############################################################"
        print '\n'
    make_hist(winners,params)
```