CrimeDetection

May 5, 2018

1 Problem Description : Crime Detection Regression Analysis

```
In [1]: from keras.models import Sequential
        from keras.layers import Dense
        from keras.layers import LSTM
        from keras.layers import Dropout
        from keras.layers import Flatten,Input
        from keras import backend as K
        from keras.models import Model, load_model
        from sklearn.preprocessing import StandardScaler
        import matplotlib.pyplot as plt
        from sklearn.model_selection import KFold
        from scipy.spatial import distance
        from sklearn.decomposition import PCA
        from numpy import linalg as LA
        from keras.objectives import categorical_crossentropy
        from sklearn.metrics import roc_curve, auc
        import math
        from scipy.stats import pearsonr
        import copy
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import mean_squared_error, r2_score
        import itertools
        import csv
        from sklearn import metrics
        import tensorflow as tf
        import tensorflow.contrib.layers as tl
        import numpy as np
        import pandas as pd
        from sklearn import linear_model
        from sklearn.ensemble import RandomForestClassifier
        import seaborn as sb
        %matplotlib inline
```

```
/home/ramchalamkr/.local/lib/python2.7/site-packages/h5py/__init__.py:36: FutureWarning: Conve
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

1.1 Steps

1.2 1. Checking Data

```
In [18]: X = pd.read_csv('crime_prep.csv',delimiter=',')
         print X.shape
         X[0:20]
```

```
(1994, 128)
```

```
Out[18]:     target  v_cont_0  v_cat_0  v_cat_1                 v_cat_2  v_cat_3  \
         0     0.20         8      NaN      NaN            Lakewoodcity        1
         1     0.67        53      NaN      NaN             Tukwilacity        1
         2     0.43        24      NaN      NaN            Aberdeentown        1
         3     0.12        34      5.0  81440.0     Willingborotownship        1
         4     0.03        42     95.0   6096.0       Bethlehemtownship        1
         5     0.14         6      NaN      NaN       SouthPasadenacity        1
         6     0.03        44      7.0  41500.0             Lincolntown        1
         7     0.55         6      NaN      NaN                Selmacity        1
         8     0.53        21      NaN      NaN           Hendersoncity        1
         9     0.15        29      NaN      NaN             Claytoncity        1
         10    0.24         6      NaN      NaN            DalyCitycity        1
         11    0.08        36      NaN      NaN  RockvilleCentrevillage        1
         12    0.06        25     21.0  44105.0             Needhamtown        1
         13    0.09        55     87.0  30075.0          GrandChutetown        1
         14    0.21         6      NaN      NaN           DanaPointcity        1
         15    0.30        19    187.0  91370.0           FortDodgecity        1
         16    0.49        36      1.0   1000.0              Albanycity        1
         17    0.07        34     27.0  17650.0        Denvilletownship        1
         18    0.15        18      NaN      NaN          Valparaisocity        1
         19    0.03        42    129.0  66376.0      Rostravertownship        1

             v_cont_5  v_cont_6  v_cont_7  v_cont_8    ...      v_cont_117  \
         0       0.19      0.33      0.02      0.90    ...            0.29
         1       0.00      0.16      0.12      0.74    ...             NaN
         2       0.00      0.42      0.49      0.56    ...             NaN
         3       0.04      0.77      1.00      0.08    ...             NaN
         4       0.01      0.55      0.02      0.95    ...             NaN
         5       0.02      0.28      0.06      0.54    ...             NaN
         6       0.01      0.39      0.00      0.98    ...             NaN
         7       0.01      0.74      0.03      0.46    ...             NaN
         8       0.03      0.34      0.20      0.84    ...             NaN
         9       0.01      0.40      0.06      0.87    ...             NaN
         10      0.13      0.71      0.15      0.07    ...             NaN
         11      0.02      0.46      0.08      0.91    ...             NaN
         12      0.03      0.47      0.01      0.96    ...             NaN
         13      0.01      0.44      0.00      0.98    ...             NaN
         14      0.04      0.36      0.01      0.85    ...             NaN
```

15	0.03	0.34	0.06	0.93	...	NaN
16	0.15	0.31	0.40	0.63	...	0.22
17	0.01	0.53	0.01	0.94	...	NaN
18	0.02	0.47	0.01	0.97	...	NaN
19	0.00	0.41	0.05	0.96	...	NaN

	v_cont_118	v_cont_119	v_cont_120	v_cont_121	v_cont_122	v_cont_123 \
0	0.12	0.26	0.20	0.06	0.04	0.90
1	0.02	0.12	0.45	NaN	NaN	NaN
2	0.01	0.21	0.02	NaN	NaN	NaN
3	0.02	0.39	0.28	NaN	NaN	NaN
4	0.04	0.09	0.02	NaN	NaN	NaN
5	0.01	0.58	0.10	NaN	NaN	NaN
6	0.05	0.08	0.06	NaN	NaN	NaN
7	0.01	0.33	0.00	NaN	NaN	NaN
8	0.04	0.17	0.04	NaN	NaN	NaN
9	0.00	0.47	0.11	NaN	NaN	NaN
10	0.02	1.00	1.00	NaN	NaN	NaN
11	0.01	0.63	1.00	NaN	NaN	NaN
12	0.03	0.18	0.59	NaN	NaN	NaN
13	0.08	0.04	0.00	NaN	NaN	NaN
14	0.02	0.40	0.15	NaN	NaN	NaN
15	0.04	0.15	0.04	NaN	NaN	NaN
16	0.06	0.39	0.84	0.06	0.06	0.91
17	0.03	0.09	0.21	NaN	NaN	NaN
18	0.03	0.20	0.07	NaN	NaN	NaN
19	0.09	0.03	0.05	NaN	NaN	NaN

	v_cont_124	v_cont_125	v_cont_126
0	0.5	0.32	0.14
1	NaN	0.00	NaN
2	NaN	0.00	NaN
3	NaN	0.00	NaN
4	NaN	0.00	NaN
5	NaN	0.00	NaN
6	NaN	0.00	NaN
7	NaN	0.00	NaN
8	NaN	0.00	NaN
9	NaN	0.00	NaN
10	NaN	0.00	NaN
11	NaN	0.00	NaN
12	NaN	0.00	NaN
13	NaN	0.00	NaN
14	NaN	0.00	NaN
15	NaN	0.00	NaN
16	0.5	0.88	0.26
17	NaN	0.00	NaN
18	NaN	0.00	NaN

```
19          NaN         0.00          NaN

[20 rows x 128 columns]
```

1.3 2. Tidying the data

Missing data can either be filled with the means of the features or maybe 0 to ignore them or even other data imputation techniques to predict the missing values

In this case the mean of the features is taken for the missing values.

In [19]:
```
print len(X['v_cat_2'].unique())
X = X.fillna(X.mean())
del X['v_cat_2']
del X['v_cat_3']
Y = X['target']
del X['target']
X.head()
```

```
1828
```

Out[19]:
```
   v_cont_0    v_cat_0        v_cat_1  v_cont_5  v_cont_6  v_cont_7  v_cont_8  \
0         8  58.826829   46188.336597      0.19      0.33      0.02      0.90
1        53  58.826829   46188.336597      0.00      0.16      0.12      0.74
2        24  58.826829   46188.336597      0.00      0.42      0.49      0.56
3        34   5.000000   81440.000000      0.04      0.77      1.00      0.08
4        42  95.000000    6096.000000      0.01      0.55      0.02      0.95

   v_cont_9  v_cont_10  v_cont_11     ...       v_cont_117  v_cont_118  \
0      0.12       0.17       0.34     ...         0.290000        0.12
1      0.45       0.07       0.26     ...         0.305987        0.02
2      0.17       0.04       0.39     ...         0.305987        0.01
3      0.12       0.10       0.51     ...         0.305987        0.02
4      0.09       0.05       0.38     ...         0.305987        0.04

   v_cont_119  v_cont_120  v_cont_121  v_cont_122  v_cont_123  v_cont_124  \
0        0.26        0.20    0.060000    0.040000    0.900000    0.500000
1        0.12        0.45    0.163103    0.076708    0.698589    0.440439
2        0.21        0.02    0.163103    0.076708    0.698589    0.440439
3        0.39        0.28    0.163103    0.076708    0.698589    0.440439
4        0.09        0.02    0.163103    0.076708    0.698589    0.440439

   v_cont_125  v_cont_126
0        0.32    0.140000
1        0.00    0.195078
2        0.00    0.195078
3        0.00    0.195078
```

```
4        0.00    0.195078

[5 rows x 125 columns]
```

```
In [20]: #Standardise certain columns
         X['v_cat_0'] = StandardScaler().fit_transform(X['v_cat_0'].values.reshape(-1, 1))
         X['v_cat_1'] = StandardScaler().fit_transform(X['v_cat_1'].values.reshape(-1, 1))
         X['v_cont_0'] = StandardScaler().fit_transform(X['v_cont_0'].values.reshape(-1, 1))
         X.head()
```

```
/usr/local/lib/python2.7/dist-packages/sklearn/utils/validation.py:444: DataConversionWarning:
  warnings.warn(msg, DataConversionWarning)
```

```
Out[20]:    v_cont_0   v_cat_0   v_cat_1  v_cont_5  v_cont_6  v_cont_7  v_cont_8  \
         0 -1.261697  0.000000  0.000000      0.19      0.33      0.02      0.90
         1  1.483304  0.000000  0.000000      0.00      0.16      0.12      0.74
         2 -0.285697  0.000000  0.000000      0.00      0.42      0.49      0.56
         3  0.324303 -0.664357  2.178117      0.04      0.77      1.00      0.08
         4  0.812304  0.446467 -2.477211      0.01      0.55      0.02      0.95

            v_cont_9  v_cont_10  v_cont_11     ...       v_cont_117  v_cont_118  \
         0      0.12       0.17       0.34     ...         0.290000        0.12
         1      0.45       0.07       0.26     ...         0.305987        0.02
         2      0.17       0.04       0.39     ...         0.305987        0.01
         3      0.12       0.10       0.51     ...         0.305987        0.02
         4      0.09       0.05       0.38     ...         0.305987        0.04

            v_cont_119  v_cont_120  v_cont_121  v_cont_122  v_cont_123  v_cont_124  \
         0        0.26        0.20    0.060000    0.040000    0.900000    0.500000
         1        0.12        0.45    0.163103    0.076708    0.698589    0.440439
         2        0.21        0.02    0.163103    0.076708    0.698589    0.440439
         3        0.39        0.28    0.163103    0.076708    0.698589    0.440439
         4        0.09        0.02    0.163103    0.076708    0.698589    0.440439

            v_cont_125  v_cont_126
         0        0.32    0.140000
         1        0.00    0.195078
         2        0.00    0.195078
         3        0.00    0.195078
         4        0.00    0.195078

         [5 rows x 125 columns]
```

1.4 3. Exploratory Analysis

1.4.1 PCA

```
In [21]: print X.shape
         X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.3,random_state=42)
```

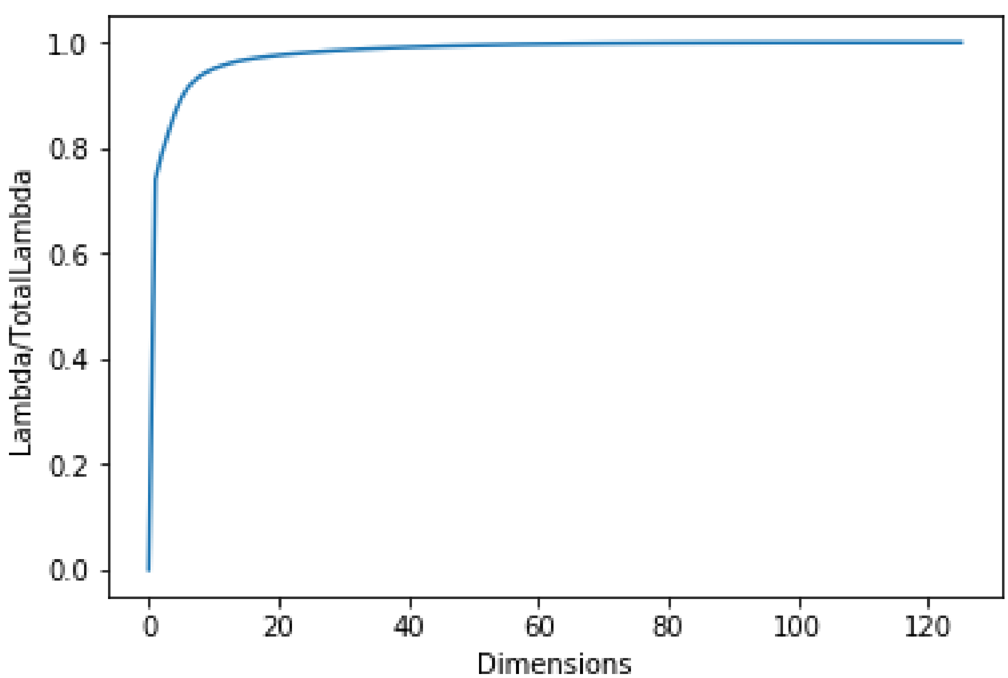
```
Covariance = np.dot(X_train.T,X_train)
print Covariance.shape

Lambda, e = LA.eigh(Covariance)
Lambda = Lambda.reshape(Lambda.shape[0],1)
Lambda = sorted(Lambda,reverse=True)
TotalLambda = np.sum(Lambda)
LambdaProp = []
for i in range(X_train.shape[1]):
    temp = np.sum(Lambda[0:i])*1.0/TotalLambda
    LambdaProp.append(temp)

Dim = np.linspace(0, X_train.shape[1], X_train.shape[1])

plt.plot(Dim,LambdaProp)
plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
           ncol=2, mode="expand", borderaxespad=0.)
plt.xlabel('Dimensions')
plt.ylabel('Lambda/TotalLambda')
plt.show()
```

```
(1994, 125)
(125, 125)
```



In [23]:
```
X1 = X
Y1 = Y

for i in (range(1,100)):
    print i
    NewFeatureSet = np.dot(X1,e[:,125-i:125])
    print NewFeatureSet.shape
    X_train,X_test,Y_train,Y_test = train_test_split(NewFeatureSet,Y1,test_size = 0.2
    #Normal Linear Regression
    regr = linear_model.LinearRegression(normalize = True)
    regr.fit(X_train,Y_train)
    Y_pred = regr.predict(X_test)
    Weights = regr.coef_
    print "RMSE %.2f" % math.sqrt(mean_squared_error(Y_test,Y_pred))
    print('Variance/R2 score: %.2f' % r2_score(Y_test, Y_pred))
```

```
1
(1994, 1)
RMSE 0.22
Variance/R2 score: -0.00
2
(1994, 2)
RMSE 0.21
Variance/R2 score: 0.04
3
(1994, 3)
RMSE 0.17
Variance/R2 score: 0.39
4
(1994, 4)
RMSE 0.16
Variance/R2 score: 0.48
5
(1994, 5)
RMSE 0.15
Variance/R2 score: 0.52
6
(1994, 6)
RMSE 0.15
Variance/R2 score: 0.52
7
(1994, 7)
RMSE 0.15
Variance/R2 score: 0.54
8
(1994, 8)
RMSE 0.15
Variance/R2 score: 0.55
```

```
9
(1994, 9)
RMSE 0.14
Variance/R2 score: 0.59
10
(1994, 10)
RMSE 0.14
Variance/R2 score: 0.61
11
(1994, 11)
RMSE 0.14
Variance/R2 score: 0.61
12
(1994, 12)
RMSE 0.13
Variance/R2 score: 0.62
13
(1994, 13)
RMSE 0.13
Variance/R2 score: 0.63
14
(1994, 14)
RMSE 0.13
Variance/R2 score: 0.63
15
(1994, 15)
RMSE 0.13
Variance/R2 score: 0.63
16
(1994, 16)
RMSE 0.13
Variance/R2 score: 0.63
17
(1994, 17)
RMSE 0.13
Variance/R2 score: 0.63
18
(1994, 18)
RMSE 0.13
Variance/R2 score: 0.64
19
(1994, 19)
RMSE 0.13
Variance/R2 score: 0.64
20
(1994, 20)
RMSE 0.13
Variance/R2 score: 0.64
```

```
21
(1994, 21)
RMSE 0.13
Variance/R2 score: 0.64
22
(1994, 22)
RMSE 0.13
Variance/R2 score: 0.64
23
(1994, 23)
RMSE 0.13
Variance/R2 score: 0.64
24
(1994, 24)
RMSE 0.13
Variance/R2 score: 0.64
25
(1994, 25)
RMSE 0.13
Variance/R2 score: 0.64
26
(1994, 26)
RMSE 0.13
Variance/R2 score: 0.64
27
(1994, 27)
RMSE 0.13
Variance/R2 score: 0.64
28
(1994, 28)
RMSE 0.13
Variance/R2 score: 0.65
29
(1994, 29)
RMSE 0.13
Variance/R2 score: 0.64
30
(1994, 30)
RMSE 0.13
Variance/R2 score: 0.64
31
(1994, 31)
RMSE 0.13
Variance/R2 score: 0.64
32
(1994, 32)
RMSE 0.13
Variance/R2 score: 0.64
```

```
33
(1994, 33)
RMSE 0.13
Variance/R2 score: 0.64
34
(1994, 34)
RMSE 0.13
Variance/R2 score: 0.64
35
(1994, 35)
RMSE 0.13
Variance/R2 score: 0.64
36
(1994, 36)
RMSE 0.13
Variance/R2 score: 0.64
37
(1994, 37)
RMSE 0.13
Variance/R2 score: 0.64
38
(1994, 38)
RMSE 0.13
Variance/R2 score: 0.63
39
(1994, 39)
RMSE 0.13
Variance/R2 score: 0.63
40
(1994, 40)
RMSE 0.13
Variance/R2 score: 0.64
41
(1994, 41)
RMSE 0.13
Variance/R2 score: 0.63
42
(1994, 42)
RMSE 0.13
Variance/R2 score: 0.64
43
(1994, 43)
RMSE 0.13
Variance/R2 score: 0.64
44
(1994, 44)
RMSE 0.13
Variance/R2 score: 0.64
```

```
45
(1994, 45)
RMSE 0.13
Variance/R2 score: 0.64
46
(1994, 46)
RMSE 0.13
Variance/R2 score: 0.64
47
(1994, 47)
RMSE 0.13
Variance/R2 score: 0.64
48
(1994, 48)
RMSE 0.13
Variance/R2 score: 0.63
49
(1994, 49)
RMSE 0.13
Variance/R2 score: 0.63
50
(1994, 50)
RMSE 0.13
Variance/R2 score: 0.63
51
(1994, 51)
RMSE 0.13
Variance/R2 score: 0.63
52
(1994, 52)
RMSE 0.13
Variance/R2 score: 0.63
53
(1994, 53)
RMSE 0.13
Variance/R2 score: 0.63
54
(1994, 54)
RMSE 0.13
Variance/R2 score: 0.62
55
(1994, 55)
RMSE 0.13
Variance/R2 score: 0.62
56
(1994, 56)
RMSE 0.14
Variance/R2 score: 0.62
```

```
57
(1994, 57)
RMSE 0.14
Variance/R2 score: 0.62
58
(1994, 58)
RMSE 0.14
Variance/R2 score: 0.62
59
(1994, 59)
RMSE 0.14
Variance/R2 score: 0.62
60
(1994, 60)
RMSE 0.13
Variance/R2 score: 0.62
61
(1994, 61)
RMSE 0.13
Variance/R2 score: 0.62
62
(1994, 62)
RMSE 0.13
Variance/R2 score: 0.62
63
(1994, 63)
RMSE 0.13
Variance/R2 score: 0.62
64
(1994, 64)
RMSE 0.14
Variance/R2 score: 0.62
65
(1994, 65)
RMSE 0.13
Variance/R2 score: 0.62
66
(1994, 66)
RMSE 0.14
Variance/R2 score: 0.62
67
(1994, 67)
RMSE 0.14
Variance/R2 score: 0.62
68
(1994, 68)
RMSE 0.14
Variance/R2 score: 0.62
```

```
69
(1994, 69)
RMSE 0.14
Variance/R2 score: 0.62
70
(1994, 70)
RMSE 0.14
Variance/R2 score: 0.62
71
(1994, 71)
RMSE 0.14
Variance/R2 score: 0.62
72
(1994, 72)
RMSE 0.14
Variance/R2 score: 0.61
73
(1994, 73)
RMSE 0.14
Variance/R2 score: 0.61
74
(1994, 74)
RMSE 0.14
Variance/R2 score: 0.61
75
(1994, 75)
RMSE 0.14
Variance/R2 score: 0.61
76
(1994, 76)
RMSE 0.14
Variance/R2 score: 0.61
77
(1994, 77)
RMSE 0.14
Variance/R2 score: 0.61
78
(1994, 78)
RMSE 0.14
Variance/R2 score: 0.61
79
(1994, 79)
RMSE 0.14
Variance/R2 score: 0.61
80
(1994, 80)
RMSE 0.14
Variance/R2 score: 0.61
```

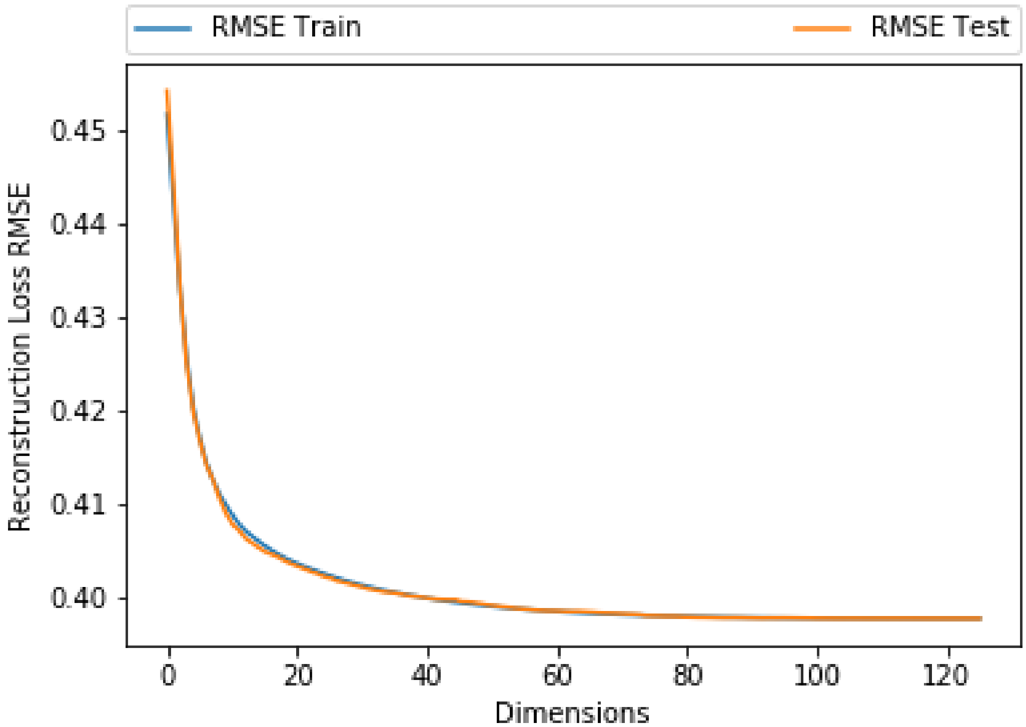
```
81
(1994, 81)
RMSE 0.14
Variance/R2 score: 0.61
82
(1994, 82)
RMSE 0.14
Variance/R2 score: 0.61
83
(1994, 83)
RMSE 0.14
Variance/R2 score: 0.61
84
(1994, 84)
RMSE 0.14
Variance/R2 score: 0.61
85
(1994, 85)
RMSE 0.14
Variance/R2 score: 0.61
86
(1994, 86)
RMSE 0.14
Variance/R2 score: 0.61
87
(1994, 87)
RMSE 0.14
Variance/R2 score: 0.61
88
(1994, 88)
RMSE 0.14
Variance/R2 score: 0.61
89
(1994, 89)
RMSE 0.14
Variance/R2 score: 0.60
90
(1994, 90)
RMSE 0.14
Variance/R2 score: 0.61
91
(1994, 91)
RMSE 0.14
Variance/R2 score: 0.61
92
(1994, 92)
RMSE 0.14
Variance/R2 score: 0.61
```

```
93
(1994, 93)
RMSE 0.14
Variance/R2 score: 0.61
94
(1994, 94)
RMSE 0.14
Variance/R2 score: 0.61
95
(1994, 95)
RMSE 0.14
Variance/R2 score: 0.61
96
(1994, 96)
RMSE 0.14
Variance/R2 score: 0.62
97
(1994, 97)
RMSE 0.14
Variance/R2 score: 0.62
98
(1994, 98)
RMSE 0.14
Variance/R2 score: 0.62
99
(1994, 99)
RMSE 0.14
Variance/R2 score: 0.62
```

```
In [24]: X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.3,random_state=42)
         RMSETrain =[]
         RMSETest =[]
         VaraincеRatio = []
         Dim = np.linspace(0, X_train.shape[1], X_train.shape[1])
         for i in range(1,X_train.shape[1]+1):
             pca = PCA(n_components = i)
             pca.fit(X_train)
             VaraincеRatio.append(pca.explained_variance_ratio_)
             X_trainhat = np.dot(pca.transform(X_train)[:,:i], pca.components_[:i,:])
             RMSETrain.append(math.sqrt(mean_squared_error(X_train,X_trainhat)))
             X_testhat = np.dot(pca.transform(X_test)[:,:i], pca.components_[:i,:])
             RMSETest.append(math.sqrt(mean_squared_error(X_test,X_testhat)))

         plt.plot(Dim,RMSETrain,label="RMSE Train")
         plt.plot(Dim,RMSETest,label="RMSE Test")
         plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
                    ncol=2, mode="expand", borderaxespad=0.)
```

```
plt.xlabel('Dimensions')
plt.ylabel('Reconstruction Loss RMSE')
plt.show()
```




```
In [25]: X1 = X
         Y1 = Y
         for i in range(1,15):
             pca = PCA(n_components = i)
             X_1 = pca.fit_transform(X1)
             #Varaince Ratio.append(pca.explained_variance_ratio_)
             print X_1
             print "no of componenets",i
             print "predict for transformed data"

             X_train,X_test,Y_train,Y_test = train_test_split(X_1,Y1,test_size = 0.2,random_sta
             #Normal Linear Regression
             regr = linear_model.LinearRegression(normalize = True)
             regr.fit(X_train,Y_train)
             Y_pred = regr.predict(X_test)
             Weights = regr.coef_
             print "RMSE %.2f" % math.sqrt(mean_squared_error(Y_test,Y_pred))
             print('Variance/R2 score: %.2f' % r2_score(Y_test, Y_pred))
```

```
[[-0.83893174]
 [ 0.91684945]
 [ 0.21196309]
 ...
 [-0.6326288 ]
 [-0.63799423]
 [-1.45620074]]
no of componenets 1
predict for transformed data
RMSE 0.22
Variance/R2 score: -0.02
[[-0.83893174  0.19106065]
 [ 0.91684946 -0.35978791]
 [ 0.21196309  0.41829515]
 ...
 [-0.6326288   0.70376198]
 [-0.63799423 -0.32252535]
 [-1.45620074  1.73787881]]
no of componenets 2
predict for transformed data
RMSE 0.16
Variance/R2 score: 0.49
[[-0.83893174  0.19106065  0.39788124]
 [ 0.91684946 -0.35978791 -0.32442181]
 [ 0.21196309  0.41829515 -0.13784697]
 ...
 [-0.6326288   0.70376198  1.6834653 ]
 [-0.63799423 -0.32252534  1.46103493]
 [-1.45620074  1.73787882  0.58259365]]
no of componenets 3
predict for transformed data
RMSE 0.16
Variance/R2 score: 0.48
[[-0.83893174  0.19106065  0.39788124  0.28589089]
 [ 0.91684946 -0.35978791 -0.32442181 -0.0468139 ]
 [ 0.21196309  0.41829515 -0.13784697 -0.30614276]
 ...
 [-0.6326288   0.70376198  1.6834653  -1.63788755]
 [-0.63799423 -0.32252534  1.46103493 -0.80879764]
 [-1.45620074  1.73787882  0.58259365  0.30865221]]
no of componenets 4
predict for transformed data
RMSE 0.15
Variance/R2 score: 0.50
[[-0.83893174  0.19106065  0.39788124  0.28589089 -0.9762273 ]
 [ 0.91684946 -0.35978791 -0.32442181 -0.0468139   0.86189957]
 [ 0.21196309  0.41829515 -0.13784697 -0.30614276 -0.70036877]
 ...
```

```
 [-0.6326288   0.70376198  1.6834653  -1.63788755 -0.3569823 ]
 [-0.63799423 -0.32252534  1.46103493 -0.80879764  0.67782456]
 [-1.45620074  1.73787882  0.58259365  0.30865221  1.00820566]]
no of componenets 5
predict for transformed data
RMSE 0.15
Variance/R2 score: 0.51
[[-0.83893174  0.19106065  0.39788124  0.28589089 -0.97622731  0.38611268]
 [ 0.91684946 -0.35978791 -0.32442181 -0.0468139   0.86189956  1.07084359]
 [ 0.21196309  0.41829515 -0.13784697 -0.30614276 -0.70036876 -0.16831285]
 ...
 [-0.6326288   0.70376198  1.6834653  -1.63788755 -0.35698233  0.25897959]
 [-0.63799423 -0.32252534  1.46103493 -0.80879764  0.67782455  0.4963086 ]
 [-1.45620074  1.73787882  0.58259365  0.30865221  1.00820565 -0.71288661]]
no of componenets 6
predict for transformed data
RMSE 0.15
Variance/R2 score: 0.51
[[-8.38931741e-01  1.91060648e-01  3.97881242e-01 ... -9.76227318e-01
   3.86112475e-01  1.63136503e-01]
 [ 9.16849455e-01 -3.59787912e-01 -3.24421808e-01 ...  8.61899550e-01
   1.07084340e+00  2.51164529e-01]
 [ 2.11963089e-01  4.18295154e-01 -1.37846971e-01 ... -7.00368786e-01
  -1.68313217e-01  2.32590950e-04]
 ...
 [-6.32628800e-01  7.03761983e-01  1.68346530e+00 ... -3.56982307e-01
   2.58979854e-01 -8.13425272e-01]
 [-6.37994231e-01 -3.22525345e-01  1.46103493e+00 ...  6.77824565e-01
   4.96308786e-01  6.61473722e-02]
 [-1.45620074e+00  1.73787882e+00  5.82593653e-01 ...  1.00820566e+00
  -7.12886689e-01  1.93132382e-01]]
no of componenets 7
predict for transformed data
RMSE 0.15
Variance/R2 score: 0.55
[[-8.38931741e-01  1.91060648e-01  3.97881242e-01 ...  3.86112724e-01
   1.63136417e-01 -5.09189088e-01]
 [ 9.16849455e-01 -3.59787912e-01 -3.24421808e-01 ...  1.07084352e+00
   2.51164623e-01 -1.49584591e-01]
 [ 2.11963089e-01  4.18295154e-01 -1.37846971e-01 ... -1.68312990e-01
   2.32076835e-04 -4.38400635e-02]
 ...
 [-6.32628800e-01  7.03761983e-01  1.68346530e+00 ...  2.58979684e-01
  -8.13425816e-01 -1.68552412e-01]
 [-6.37994231e-01 -3.22525345e-01  1.46103493e+00 ...  4.96308677e-01
   6.61473485e-02  2.97791972e-01]
 [-1.45620074e+00  1.73787882e+00  5.82593653e-01 ... -7.12886814e-01
   1.93131924e-01 -5.57367450e-01]]
```

```
no of componenets 8
predict for transformed data
RMSE 0.14
Variance/R2 score: 0.59
[[-8.38931741e-01  1.91060648e-01  3.97881242e-01 ...  1.63136521e-01
  -5.09186136e-01 -4.34768594e-01]
 [ 9.16849455e-01 -3.59787912e-01 -3.24421808e-01 ...  2.51164624e-01
  -1.49586134e-01 -8.24804485e-01]
 [ 2.11963089e-01  4.18295154e-01 -1.37846971e-01 ...  2.32085531e-04
  -4.38406691e-02  1.48838088e-01]
 ...
 [-6.32628800e-01  7.03761983e-01  1.68346530e+00 ... -8.13425763e-01
  -1.68551238e-01 -6.53347688e-02]
 [-6.37994231e-01 -3.22525345e-01  1.46103493e+00 ...  6.61473512e-02
   2.97790406e-01 -1.26347619e-01]
 [-1.45620074e+00  1.73787882e+00  5.82593653e-01 ...  1.93132012e-01
  -5.57363522e-01 -3.25756539e-01]]
no of componenets 9
predict for transformed data
RMSE 0.14
Variance/R2 score: 0.61
[[-0.83893174  0.19106065  0.39788124 ... -0.50918594 -0.43477104
   0.04578273]
 [ 0.91684946 -0.35978791 -0.32442181 ... -0.14958673 -0.824802
   0.24931816]
 [ 0.21196309  0.41829515 -0.13784697 ... -0.04383982  0.14883761
   0.44705059]
 ...
 [-0.6326288   0.70376198  1.6834653  ... -0.16855119 -0.06534024
  -0.13388745]
 [-0.63799423 -0.32252534  1.46103493 ...  0.29779086 -0.12634794
  -0.58565302]
 [-1.45620074  1.73787882  0.58259365 ... -0.55736359 -0.32576343
   0.00474748]]
no of componenets 10
predict for transformed data
RMSE 0.13
Variance/R2 score: 0.63
[[-0.83893174  0.19106065  0.39788124 ... -0.43476782  0.0457842
   0.02812753]
 [ 0.91684946 -0.35978791 -0.32442181 ... -0.82480229  0.24931289
  -0.72522575]
 [ 0.21196309  0.41829515 -0.13784697 ...  0.14883595  0.44705413
  -0.50833336]
 ...
 [-0.6326288   0.70376198  1.6834653  ... -0.0653433  -0.13387715
   0.1310201 ]
 [-0.63799423 -0.32252534  1.46103493 ... -0.12635255 -0.58564017
```

```
  -0.0699597 ]
 [-1.45620074  1.73787882  0.58259365 ... -0.3257647   0.00472953
   0.03595945]]
no of componenets 11
predict for transformed data
RMSE 0.13
Variance/R2 score: 0.63
[[-0.83893174  0.19106065  0.39788124 ...  0.04575498  0.02812399
   0.18395406]
 [ 0.91684946 -0.35978791 -0.32442181 ...  0.24932038 -0.72522364
  -0.02691651]
 [ 0.21196309  0.41829515 -0.13784697 ...  0.447057   -0.50832742
  -0.01767973]
 ...
 [-0.6326288   0.70376198  1.6834653  ... -0.13384475  0.13102212
   0.10336862]
 [-0.63799423 -0.32252534  1.46103493 ... -0.58561298 -0.06995331
  -0.0472847 ]
 [-1.45620074  1.73787882  0.58259365 ...  0.00476318  0.03596686
   0.10974159]]
no of componenets 12
predict for transformed data
RMSE 0.13
Variance/R2 score: 0.63
[[-0.83893174  0.19106071  0.39788123 ...  0.02780265  0.18346462
   0.08048317]
 [ 0.91684945 -0.35978793 -0.32442186 ... -0.725408   -0.0270232
  -0.15179718]
 [ 0.2119631   0.41829514 -0.13784691 ... -0.50811985 -0.01690639
   0.38017355]
 ...
 [-0.6326288   0.70376196  1.68346533 ...  0.13085839  0.10375661
   0.44058922]
 [-0.63799422 -0.32252537  1.46103492 ... -0.0696856  -0.04660295
   0.51356321]
 [-1.45620074  1.7378788   0.58259357 ...  0.03529639  0.11024077
  -0.0257058 ]]
no of componenets 13
predict for transformed data
RMSE 0.13
Variance/R2 score: 0.63
[[-0.83893171  0.19106063  0.39788118 ...  0.18225289  0.08196732
  -0.22805535]
 [ 0.91684944 -0.3597879  -0.32442179 ... -0.02744453 -0.14949181
  -0.30509614]
 [ 0.21196309  0.41829516 -0.13784697 ... -0.01795268  0.38026324
  -0.27291182]
 ...
```

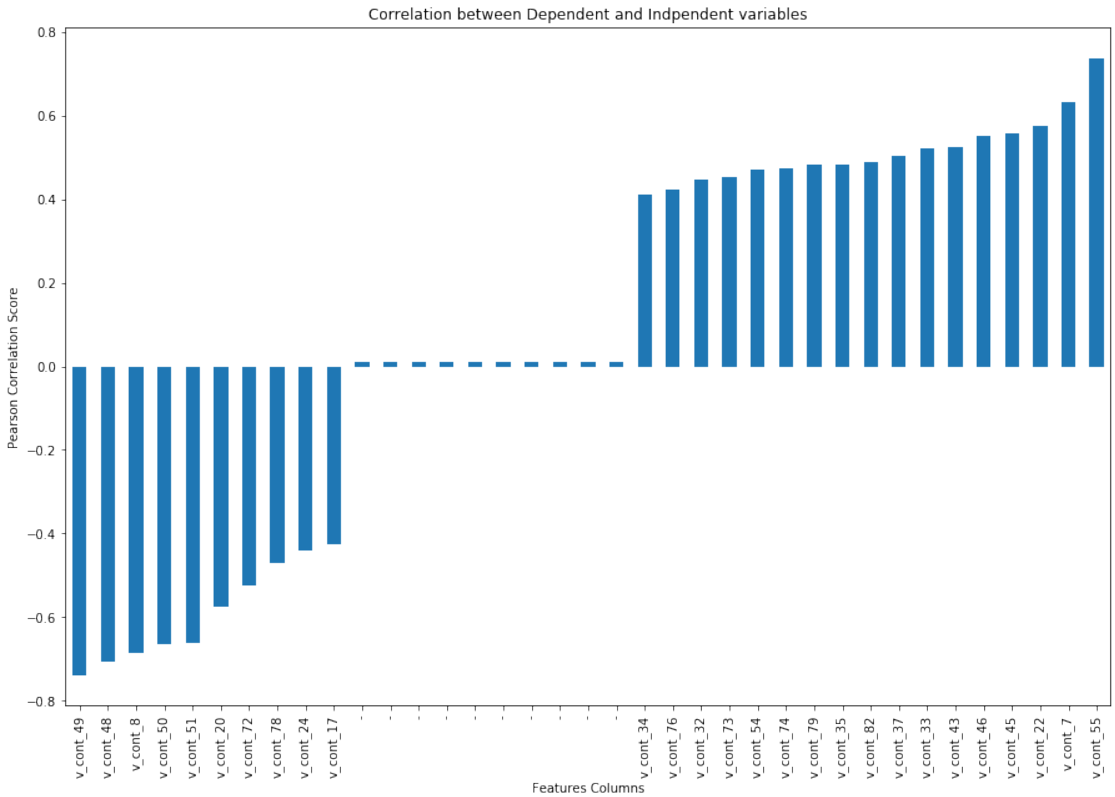
```
 [-0.6326288   0.70376199  1.6834653  ...  0.10205886  0.44238049
   0.05283028]
 [-0.63799422 -0.32252531  1.46103489 ... -0.04973349  0.51472393
   0.08188874]
 [-1.45620075  1.73787882  0.58259369 ...  0.11085379 -0.02437587
   0.17981367]]
no of componenets 14
predict for transformed data
RMSE 0.13
Variance/R2 score: 0.63
```

1.4.2 Feature engineering using Pearson's Coefficient

In [26]:
```
X = (X - X.mean())
Matrix = X.as_matrix()
#print TrainX['ALSQM_Count']
r=[]
p=[]
for i in range(X.shape[1]):
    t1,t2 = pearsonr(Matrix[:,i],Y)
    #print t1,i,
    r.append((t1,i,X.columns[i],t2))
r.sort()
```

In [27]:
```
frequencies = []
labels =[]
count=0
for ind,x in enumerate(r):
    if(x[0]<-0.4 or x[0]>0.4):
        frequencies.append(x[0])
        labels.append(x[2])
    else:
        if count<10:
            labels.append('-')
            frequencies.append(0.01)
            count+=1
print len(frequencies)
freq_series = pd.Series.from_array(frequencies)
x_labels = range(len(freq_series))
plt.figure(figsize=(15, 10))
ax = freq_series.plot(kind='bar')
ax.set_title('Correlation between Dependent and Indpendent variables')
ax.set_xlabel('Features Columns')
ax.set_ylabel('Pearson Correlation Score')
rects = ax.patches
ax.set_xticklabels(labels)
plt.show()
```

37



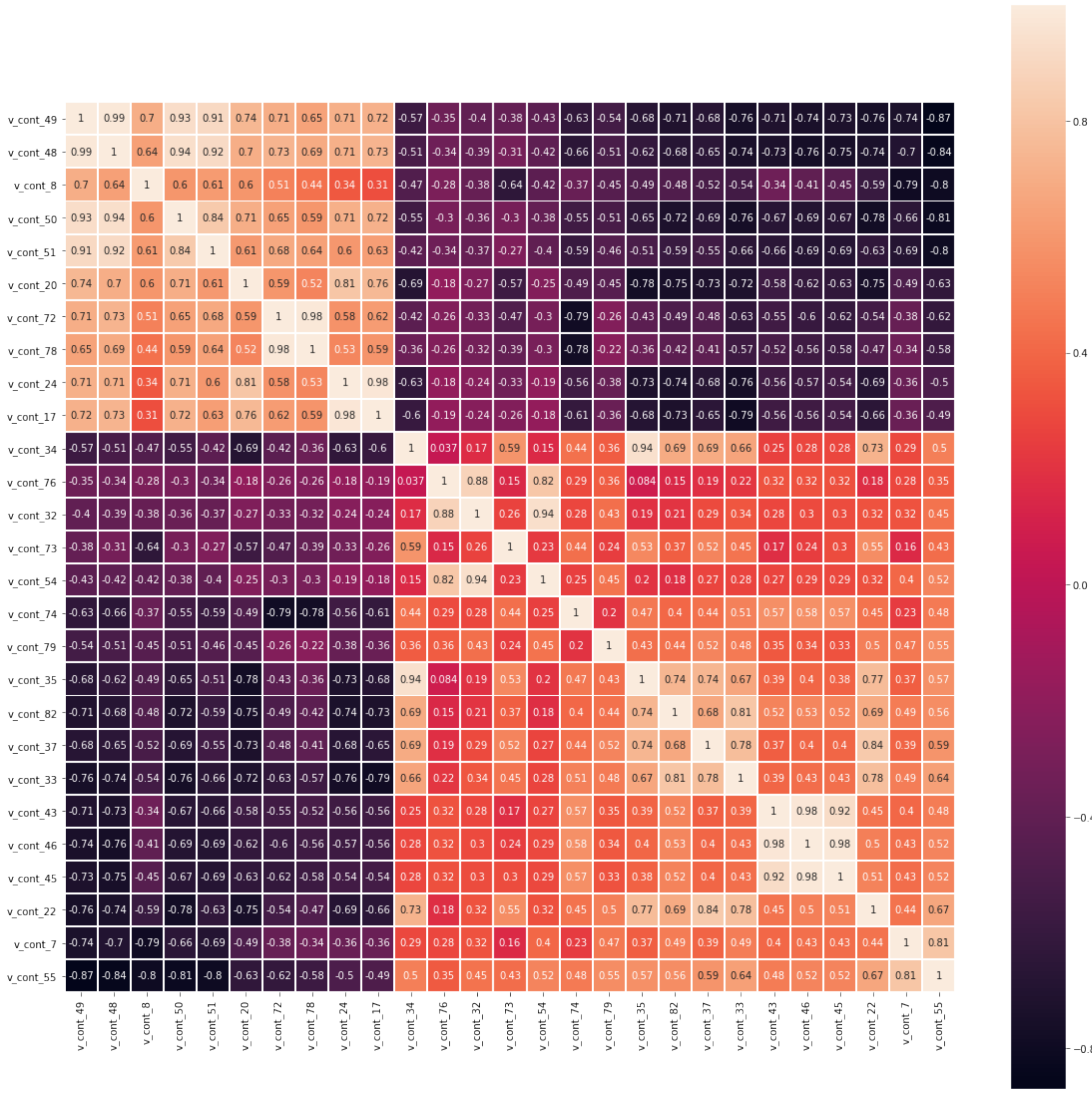
In [28]:
```
IndList =[]
for ind,x in enumerate(r):
    if(x[0]<-0.4 or x[0]>0.4):
        IndList.append(x[1])
CorrList =[]
CorrMatrix = np.zeros((len(IndList),len(IndList)))
Cols =[]
for i in range(len(IndList)):
    Cols.append(X.columns[IndList[i]])
    for j in range(len(IndList)):
        t1,t2 = pearsonr(Matrix[:,IndList[i]],Matrix[:,IndList[j]])
        CorrList.append((t1,IndList[i],X.columns[IndList[i]],IndList[j],X.columns[IndL
        CorrMatrix[i][j] = t1
print CorrMatrix.shape
```

```
(27, 27)
```

In [29]:
```
fig, ax = plt.subplots(figsize=(20,20))
sb.heatmap(CorrMatrix,
```

```
            xticklabels=Cols,
            yticklabels=Cols,linewidths=1,annot=True, ax=ax,square =True)

plt.show()
```



1.5 4. Model Development and Performance

```
In [30]: print IndList
         t=[]
         for i in IndList:
             t.append(X.columns[i])
         print t
         FinalTrainSet = np.zeros((len(IndList),Matrix.shape[0]))
```

```
for ind,i in enumerate(IndList):
    FinalTrainSet[ind] = (Matrix[:,i])
FinalTrainSet = FinalTrainSet.T
print FinalTrainSet.shape
print Y.shape
```

```
[47, 46, 6, 48, 49, 18, 70, 76, 22, 15, 32, 74, 30, 71, 52, 72, 77, 33, 80, 35, 31, 41, 44, 43
['v_cont_49', 'v_cont_48', 'v_cont_8', 'v_cont_50', 'v_cont_51', 'v_cont_20', 'v_cont_72', 'v_
(1994, 27)
(1994,)
```

1.5.1 Linear Regression

In [31]:
```
X_train,X_test,Y_train,Y_test = train_test_split(FinalTrainSet,Y,test_size = 0.3,rando
#Normal Linear Regression
clf = linear_model.LinearRegression(normalize = True)
clf.fit(X_train,Y_train)
Y_pred = clf.predict(X_test)
Weights = clf.coef_
print Weights
print "RMSE %.2f" % math.sqrt(mean_squared_error(Y_test,Y_pred))
print('Variance/R2 score: %.2f' % r2_score(Y_test, Y_pred))

# Linear Regression With Ridge regularisation
clf = linear_model.Ridge(alpha = 0.1,normalize = True)
clf.fit(X_train,Y_train)
Y_pred = clf.predict(X_test)
print "RMSE %.2f" % math.sqrt(mean_squared_error(Y_test,Y_pred))
print('Variance/R2 score: %.2f' % r2_score(Y_test, Y_pred))


# Linear Regression With Lasso regularisation
clf = linear_model.Lasso(alpha = 0.1,normalize = True)
clf.fit(X_train,Y_train)
Y_pred = clf.predict(X_test)
print "RMSE %.2f" % math.sqrt(mean_squared_error(Y_test,Y_pred))
print('Variance/R2 score: %.2f' % r2_score(Y_test, Y_pred))
```

```
[-0.06986744 -0.20319033  0.00282052 -0.0764294  -0.00365693 -0.11640938
 -0.3154949   0.35358964  0.21298279 -0.04158515 -0.16352962  0.26255075
  0.12639153  0.2220687  -0.23364425  0.10495532  0.04729825  0.11573447
  0.02870576 -0.05019055 -0.06960878  0.4324694  -0.61741931  0.26619755
  0.05778619  0.19690957  0.16119416]
RMSE 0.13
Variance/R2 score: 0.64
RMSE 0.13
Variance/R2 score: 0.64
```

```
RMSE 0.22
Variance/R2 score: -0.01
```

1.5.2 Neural Nets

```
In [32]: X_train,X_test,Y_train,Y_test = train_test_split(FinalTrainSet,Y,test_size = 0.3,rando
         InputWidth = X_train.shape[1]
         K.clear_session()
         model = Sequential()
         model.add(Dense(128, input_shape = (InputWidth,), activation='relu'))
         model.add(Dropout(0.2))
         model.add(Dense(64, activation='relu'))
         model.add(Dropout(0.2))
         model.add(Dense(32, activation='relu'))
         model.add(Dropout(0.2))
         model.add(Dense(1, activation='relu'))
         #model.add(Dense(InputWidth, activation='relu'))
         print model.summary()
         # Compile model
         model.compile(loss='mean_squared_error', optimizer='sgd', metrics=['accuracy'])
         op = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), nb_epoch=150, batch
         y_pred = model.predict(X_test)
         print "RMSE %.2f" % math.sqrt(mean_squared_error(Y_test,Y_pred))
         print('Variance score: %.2f' % r2_score(Y_test, Y_pred))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               3584
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout_3 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 33
=================================================================
Total params: 13,953
Trainable params: 13,953
Non-trainable params: 0
_________________________________________________________________
None
```

```
/usr/local/lib/python2.7/dist-packages/keras/models.py:848: UserWarning: The `nb_epoch` argumen
  warnings.warn('The `nb_epoch` argument in `fit` '


Train on 1395 samples, validate on 599 samples
Epoch 1/150
14s - loss: 0.0808 - acc: 0.0036 - val_loss: 0.0559 - val_acc: 0.0083
Epoch 2/150
0s - loss: 0.0626 - acc: 0.0050 - val_loss: 0.0459 - val_acc: 0.0083
Epoch 3/150
0s - loss: 0.0555 - acc: 0.0065 - val_loss: 0.0394 - val_acc: 0.0083
Epoch 4/150
0s - loss: 0.0497 - acc: 0.0079 - val_loss: 0.0347 - val_acc: 0.0100
Epoch 5/150
0s - loss: 0.0455 - acc: 0.0115 - val_loss: 0.0313 - val_acc: 0.0100
Epoch 6/150
0s - loss: 0.0406 - acc: 0.0122 - val_loss: 0.0288 - val_acc: 0.0117
Epoch 7/150
0s - loss: 0.0412 - acc: 0.0136 - val_loss: 0.0268 - val_acc: 0.0117
Epoch 8/150
0s - loss: 0.0388 - acc: 0.0143 - val_loss: 0.0255 - val_acc: 0.0117
Epoch 9/150
0s - loss: 0.0375 - acc: 0.0158 - val_loss: 0.0242 - val_acc: 0.0117
Epoch 10/150
0s - loss: 0.0362 - acc: 0.0151 - val_loss: 0.0232 - val_acc: 0.0150
Epoch 11/150
0s - loss: 0.0370 - acc: 0.0151 - val_loss: 0.0226 - val_acc: 0.0150
Epoch 12/150
0s - loss: 0.0334 - acc: 0.0179 - val_loss: 0.0219 - val_acc: 0.0150
Epoch 13/150
0s - loss: 0.0331 - acc: 0.0194 - val_loss: 0.0215 - val_acc: 0.0150
Epoch 14/150
0s - loss: 0.0318 - acc: 0.0172 - val_loss: 0.0210 - val_acc: 0.0150
Epoch 15/150
0s - loss: 0.0324 - acc: 0.0186 - val_loss: 0.0207 - val_acc: 0.0167
Epoch 16/150
0s - loss: 0.0316 - acc: 0.0186 - val_loss: 0.0205 - val_acc: 0.0167
Epoch 17/150
0s - loss: 0.0322 - acc: 0.0215 - val_loss: 0.0203 - val_acc: 0.0167
Epoch 18/150
0s - loss: 0.0305 - acc: 0.0208 - val_loss: 0.0201 - val_acc: 0.0167
Epoch 19/150
0s - loss: 0.0306 - acc: 0.0208 - val_loss: 0.0200 - val_acc: 0.0167
Epoch 20/150
0s - loss: 0.0297 - acc: 0.0179 - val_loss: 0.0199 - val_acc: 0.0167
Epoch 21/150
```

```
0s - loss: 0.0318 - acc: 0.0194 - val_loss: 0.0198 - val_acc: 0.0167
Epoch 22/150
0s - loss: 0.0318 - acc: 0.0186 - val_loss: 0.0198 - val_acc: 0.0167
Epoch 23/150
0s - loss: 0.0308 - acc: 0.0201 - val_loss: 0.0196 - val_acc: 0.0184
Epoch 24/150
0s - loss: 0.0311 - acc: 0.0194 - val_loss: 0.0196 - val_acc: 0.0184
Epoch 25/150
0s - loss: 0.0311 - acc: 0.0208 - val_loss: 0.0196 - val_acc: 0.0184
Epoch 26/150
0s - loss: 0.0296 - acc: 0.0222 - val_loss: 0.0195 - val_acc: 0.0184
Epoch 27/150
0s - loss: 0.0284 - acc: 0.0237 - val_loss: 0.0194 - val_acc: 0.0184
Epoch 28/150
0s - loss: 0.0291 - acc: 0.0186 - val_loss: 0.0194 - val_acc: 0.0184
Epoch 29/150
0s - loss: 0.0295 - acc: 0.0237 - val_loss: 0.0193 - val_acc: 0.0184
Epoch 30/150
0s - loss: 0.0288 - acc: 0.0201 - val_loss: 0.0194 - val_acc: 0.0184
Epoch 31/150
0s - loss: 0.0290 - acc: 0.0201 - val_loss: 0.0193 - val_acc: 0.0184
Epoch 32/150
0s - loss: 0.0283 - acc: 0.0229 - val_loss: 0.0193 - val_acc: 0.0184
Epoch 33/150
0s - loss: 0.0290 - acc: 0.0229 - val_loss: 0.0192 - val_acc: 0.0184
Epoch 34/150
0s - loss: 0.0297 - acc: 0.0222 - val_loss: 0.0193 - val_acc: 0.0184
Epoch 35/150
0s - loss: 0.0286 - acc: 0.0229 - val_loss: 0.0192 - val_acc: 0.0184
Epoch 36/150
0s - loss: 0.0278 - acc: 0.0237 - val_loss: 0.0192 - val_acc: 0.0184
Epoch 37/150
0s - loss: 0.0280 - acc: 0.0222 - val_loss: 0.0193 - val_acc: 0.0184
Epoch 38/150
0s - loss: 0.0310 - acc: 0.0194 - val_loss: 0.0192 - val_acc: 0.0184
Epoch 39/150
0s - loss: 0.0278 - acc: 0.0186 - val_loss: 0.0191 - val_acc: 0.0184
Epoch 40/150
0s - loss: 0.0286 - acc: 0.0201 - val_loss: 0.0191 - val_acc: 0.0184
Epoch 41/150
0s - loss: 0.0277 - acc: 0.0215 - val_loss: 0.0191 - val_acc: 0.0184
Epoch 42/150
0s - loss: 0.0275 - acc: 0.0222 - val_loss: 0.0190 - val_acc: 0.0184
Epoch 43/150
0s - loss: 0.0276 - acc: 0.0201 - val_loss: 0.0191 - val_acc: 0.0184
Epoch 44/150
0s - loss: 0.0273 - acc: 0.0222 - val_loss: 0.0190 - val_acc: 0.0184
Epoch 45/150
```

```
0s - loss: 0.0285 - acc: 0.0229 - val_loss: 0.0190 - val_acc: 0.0184
Epoch 46/150
0s - loss: 0.0276 - acc: 0.0208 - val_loss: 0.0191 - val_acc: 0.0184
Epoch 47/150
0s - loss: 0.0285 - acc: 0.0194 - val_loss: 0.0190 - val_acc: 0.0184
Epoch 48/150
0s - loss: 0.0265 - acc: 0.0237 - val_loss: 0.0190 - val_acc: 0.0184
Epoch 49/150
0s - loss: 0.0282 - acc: 0.0201 - val_loss: 0.0191 - val_acc: 0.0184
Epoch 50/150
0s - loss: 0.0282 - acc: 0.0201 - val_loss: 0.0191 - val_acc: 0.0184
Epoch 51/150
0s - loss: 0.0280 - acc: 0.0215 - val_loss: 0.0189 - val_acc: 0.0184
Epoch 52/150
0s - loss: 0.0267 - acc: 0.0229 - val_loss: 0.0190 - val_acc: 0.0184
Epoch 53/150
0s - loss: 0.0286 - acc: 0.0215 - val_loss: 0.0190 - val_acc: 0.0184
Epoch 54/150
0s - loss: 0.0270 - acc: 0.0208 - val_loss: 0.0189 - val_acc: 0.0184
Epoch 55/150
0s - loss: 0.0271 - acc: 0.0208 - val_loss: 0.0190 - val_acc: 0.0184
Epoch 56/150
0s - loss: 0.0274 - acc: 0.0229 - val_loss: 0.0189 - val_acc: 0.0184
Epoch 57/150
0s - loss: 0.0267 - acc: 0.0222 - val_loss: 0.0189 - val_acc: 0.0184
Epoch 58/150
0s - loss: 0.0278 - acc: 0.0222 - val_loss: 0.0188 - val_acc: 0.0184
Epoch 59/150
0s - loss: 0.0256 - acc: 0.0201 - val_loss: 0.0188 - val_acc: 0.0184
Epoch 60/150
0s - loss: 0.0260 - acc: 0.0237 - val_loss: 0.0188 - val_acc: 0.0184
Epoch 61/150
0s - loss: 0.0276 - acc: 0.0201 - val_loss: 0.0187 - val_acc: 0.0184
Epoch 62/150
0s - loss: 0.0284 - acc: 0.0215 - val_loss: 0.0187 - val_acc: 0.0184
Epoch 63/150
0s - loss: 0.0268 - acc: 0.0229 - val_loss: 0.0187 - val_acc: 0.0184
Epoch 64/150
0s - loss: 0.0274 - acc: 0.0215 - val_loss: 0.0187 - val_acc: 0.0184
Epoch 65/150
0s - loss: 0.0264 - acc: 0.0215 - val_loss: 0.0187 - val_acc: 0.0184
Epoch 66/150
0s - loss: 0.0266 - acc: 0.0229 - val_loss: 0.0187 - val_acc: 0.0184
Epoch 67/150
0s - loss: 0.0261 - acc: 0.0229 - val_loss: 0.0187 - val_acc: 0.0184
Epoch 68/150
0s - loss: 0.0278 - acc: 0.0208 - val_loss: 0.0186 - val_acc: 0.0184
Epoch 69/150
```

```
0s - loss: 0.0251 - acc: 0.0244 - val_loss: 0.0186 - val_acc: 0.0184
Epoch 70/150
0s - loss: 0.0275 - acc: 0.0222 - val_loss: 0.0186 - val_acc: 0.0184
Epoch 71/150
0s - loss: 0.0270 - acc: 0.0237 - val_loss: 0.0186 - val_acc: 0.0184
Epoch 72/150
0s - loss: 0.0274 - acc: 0.0244 - val_loss: 0.0186 - val_acc: 0.0184
Epoch 73/150
0s - loss: 0.0264 - acc: 0.0208 - val_loss: 0.0186 - val_acc: 0.0184
Epoch 74/150
0s - loss: 0.0261 - acc: 0.0244 - val_loss: 0.0186 - val_acc: 0.0184
Epoch 75/150
0s - loss: 0.0262 - acc: 0.0229 - val_loss: 0.0185 - val_acc: 0.0184
Epoch 76/150
0s - loss: 0.0260 - acc: 0.0222 - val_loss: 0.0185 - val_acc: 0.0184
Epoch 77/150
0s - loss: 0.0252 - acc: 0.0237 - val_loss: 0.0185 - val_acc: 0.0184
Epoch 78/150
0s - loss: 0.0259 - acc: 0.0201 - val_loss: 0.0185 - val_acc: 0.0184
Epoch 79/150
0s - loss: 0.0271 - acc: 0.0208 - val_loss: 0.0185 - val_acc: 0.0184
Epoch 80/150
0s - loss: 0.0260 - acc: 0.0222 - val_loss: 0.0185 - val_acc: 0.0184
Epoch 81/150
0s - loss: 0.0248 - acc: 0.0229 - val_loss: 0.0185 - val_acc: 0.0184
Epoch 82/150
0s - loss: 0.0278 - acc: 0.0222 - val_loss: 0.0185 - val_acc: 0.0184
Epoch 83/150
0s - loss: 0.0258 - acc: 0.0215 - val_loss: 0.0184 - val_acc: 0.0184
Epoch 84/150
0s - loss: 0.0260 - acc: 0.0229 - val_loss: 0.0184 - val_acc: 0.0184
Epoch 85/150
0s - loss: 0.0256 - acc: 0.0237 - val_loss: 0.0184 - val_acc: 0.0184
Epoch 86/150
0s - loss: 0.0244 - acc: 0.0237 - val_loss: 0.0184 - val_acc: 0.0184
Epoch 87/150
0s - loss: 0.0264 - acc: 0.0222 - val_loss: 0.0184 - val_acc: 0.0184
Epoch 88/150
0s - loss: 0.0256 - acc: 0.0244 - val_loss: 0.0184 - val_acc: 0.0184
Epoch 89/150
0s - loss: 0.0248 - acc: 0.0229 - val_loss: 0.0184 - val_acc: 0.0184
Epoch 90/150
0s - loss: 0.0274 - acc: 0.0229 - val_loss: 0.0183 - val_acc: 0.0184
Epoch 91/150
0s - loss: 0.0260 - acc: 0.0229 - val_loss: 0.0184 - val_acc: 0.0184
Epoch 92/150
0s - loss: 0.0252 - acc: 0.0229 - val_loss: 0.0183 - val_acc: 0.0184
Epoch 93/150
```

```
0s - loss: 0.0247 - acc: 0.0194 - val_loss: 0.0183 - val_acc: 0.0184
Epoch 94/150
0s - loss: 0.0265 - acc: 0.0208 - val_loss: 0.0183 - val_acc: 0.0184
Epoch 95/150
0s - loss: 0.0250 - acc: 0.0222 - val_loss: 0.0183 - val_acc: 0.0184
Epoch 96/150
0s - loss: 0.0255 - acc: 0.0194 - val_loss: 0.0183 - val_acc: 0.0184
Epoch 97/150
0s - loss: 0.0258 - acc: 0.0237 - val_loss: 0.0183 - val_acc: 0.0184
Epoch 98/150
0s - loss: 0.0259 - acc: 0.0194 - val_loss: 0.0183 - val_acc: 0.0184
Epoch 99/150
0s - loss: 0.0254 - acc: 0.0222 - val_loss: 0.0183 - val_acc: 0.0184
Epoch 100/150
0s - loss: 0.0240 - acc: 0.0237 - val_loss: 0.0183 - val_acc: 0.0184
Epoch 101/150
0s - loss: 0.0251 - acc: 0.0208 - val_loss: 0.0182 - val_acc: 0.0184
Epoch 102/150
0s - loss: 0.0256 - acc: 0.0237 - val_loss: 0.0182 - val_acc: 0.0184
Epoch 103/150
0s - loss: 0.0264 - acc: 0.0215 - val_loss: 0.0182 - val_acc: 0.0184
Epoch 104/150
0s - loss: 0.0248 - acc: 0.0244 - val_loss: 0.0182 - val_acc: 0.0184
Epoch 105/150
0s - loss: 0.0258 - acc: 0.0237 - val_loss: 0.0182 - val_acc: 0.0184
Epoch 106/150
0s - loss: 0.0259 - acc: 0.0244 - val_loss: 0.0182 - val_acc: 0.0184
Epoch 107/150
0s - loss: 0.0254 - acc: 0.0251 - val_loss: 0.0182 - val_acc: 0.0184
Epoch 108/150
0s - loss: 0.0261 - acc: 0.0229 - val_loss: 0.0182 - val_acc: 0.0184
Epoch 109/150
0s - loss: 0.0239 - acc: 0.0222 - val_loss: 0.0182 - val_acc: 0.0184
Epoch 110/150
0s - loss: 0.0248 - acc: 0.0251 - val_loss: 0.0182 - val_acc: 0.0184
Epoch 111/150
0s - loss: 0.0243 - acc: 0.0237 - val_loss: 0.0182 - val_acc: 0.0184
Epoch 112/150
0s - loss: 0.0257 - acc: 0.0229 - val_loss: 0.0182 - val_acc: 0.0184
Epoch 113/150
0s - loss: 0.0246 - acc: 0.0215 - val_loss: 0.0181 - val_acc: 0.0184
Epoch 114/150
0s - loss: 0.0246 - acc: 0.0251 - val_loss: 0.0181 - val_acc: 0.0184
Epoch 115/150
0s - loss: 0.0250 - acc: 0.0237 - val_loss: 0.0181 - val_acc: 0.0184
Epoch 116/150
0s - loss: 0.0264 - acc: 0.0229 - val_loss: 0.0182 - val_acc: 0.0184
Epoch 117/150
```

```
0s - loss: 0.0258 - acc: 0.0237 - val_loss: 0.0181 - val_acc: 0.0184
Epoch 118/150
0s - loss: 0.0261 - acc: 0.0201 - val_loss: 0.0181 - val_acc: 0.0184
Epoch 119/150
0s - loss: 0.0254 - acc: 0.0237 - val_loss: 0.0181 - val_acc: 0.0184
Epoch 120/150
0s - loss: 0.0253 - acc: 0.0244 - val_loss: 0.0181 - val_acc: 0.0184
Epoch 121/150
0s - loss: 0.0254 - acc: 0.0215 - val_loss: 0.0181 - val_acc: 0.0184
Epoch 122/150
0s - loss: 0.0253 - acc: 0.0244 - val_loss: 0.0181 - val_acc: 0.0184
Epoch 123/150
0s - loss: 0.0234 - acc: 0.0251 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 124/150
0s - loss: 0.0239 - acc: 0.0237 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 125/150
0s - loss: 0.0242 - acc: 0.0215 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 126/150
0s - loss: 0.0243 - acc: 0.0222 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 127/150
0s - loss: 0.0251 - acc: 0.0208 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 128/150
0s - loss: 0.0264 - acc: 0.0222 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 129/150
0s - loss: 0.0235 - acc: 0.0222 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 130/150
0s - loss: 0.0255 - acc: 0.0215 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 131/150
0s - loss: 0.0262 - acc: 0.0229 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 132/150
0s - loss: 0.0249 - acc: 0.0237 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 133/150
0s - loss: 0.0247 - acc: 0.0222 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 134/150
0s - loss: 0.0266 - acc: 0.0208 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 135/150
0s - loss: 0.0249 - acc: 0.0237 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 136/150
0s - loss: 0.0265 - acc: 0.0215 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 137/150
0s - loss: 0.0240 - acc: 0.0229 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 138/150
0s - loss: 0.0241 - acc: 0.0251 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 139/150
0s - loss: 0.0236 - acc: 0.0237 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 140/150
0s - loss: 0.0261 - acc: 0.0222 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 141/150
```

```
0s - loss: 0.0255 - acc: 0.0229 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 142/150
0s - loss: 0.0235 - acc: 0.0222 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 143/150
0s - loss: 0.0248 - acc: 0.0229 - val_loss: 0.0179 - val_acc: 0.0184
Epoch 144/150
0s - loss: 0.0244 - acc: 0.0244 - val_loss: 0.0180 - val_acc: 0.0184
Epoch 145/150
0s - loss: 0.0249 - acc: 0.0229 - val_loss: 0.0179 - val_acc: 0.0184
Epoch 146/150
0s - loss: 0.0234 - acc: 0.0244 - val_loss: 0.0179 - val_acc: 0.0184
Epoch 147/150
0s - loss: 0.0231 - acc: 0.0251 - val_loss: 0.0179 - val_acc: 0.0184
Epoch 148/150
0s - loss: 0.0235 - acc: 0.0244 - val_loss: 0.0179 - val_acc: 0.0184
Epoch 149/150
0s - loss: 0.0232 - acc: 0.0229 - val_loss: 0.0179 - val_acc: 0.0184
Epoch 150/150
0s - loss: 0.0243 - acc: 0.0222 - val_loss: 0.0179 - val_acc: 0.0184
RMSE 0.22
Variance score: -0.01
```

FNN doesnt perform as well possibly due to the limited number of training samples.

In [33]:
```
HighCorrTuples =[]
LowCorrTuples=[]
Col =[]
ht ={}
CorrCopy = copy.deepcopy(CorrList)
for var in CorrCopy:
    if((var[0]>=0.8 or var[0]<=-0.8) and var[1]!=var[3]):
        if(var[1] not in ht or var[3] not in ht):
            print var
            ht[var[3]] =0
    else:
        LowCorrTuples.append(var)
print len(ht)
FinalFeatureSet =[]
print len(CorrCopy)
for index,var in enumerate(CorrCopy):
    if(not(var[1] in ht or var[3] in ht)):
        #print var
        FinalFeatureSet.append(var)
print len(CorrList)
print len(CorrCopy)
print len(FinalFeatureSet)
'''
```

```
        assert(len(HighCorrTuples)+len(LowCorrTuples)==len(Corr))
        UncorrelatedFeatureIndices=[]

        for i in HighCorrTuples:
            UncorrelatedFeatureIndices.remove(i[1])
        print len(UncorrelatedFeatureIndices)

        for i in LowCorrTuples:
            if(i[1]!=i[3]):
                if(i[1] not in UncorrelatedFeatureIndices):
                    UncorrelatedFeatureIndices.append(i[1])
                if(i[3] not in UncorrelatedFeatureIndices):
                    UncorrelatedFeatureIndices.append(i[3])
        print len(UncorrelatedFeatureIndices)
        print UncorrelatedFeatureIndices

        '''
```

```
(0.985358028188035, 47, 'v_cont_49', 46, 'v_cont_48', 0.0)
(0.9311545163340417, 47, 'v_cont_49', 48, 'v_cont_50', 0.0)
(0.9076518065031174, 47, 'v_cont_49', 49, 'v_cont_51', 0.0)
(-0.8701983204454441, 47, 'v_cont_49', 53, 'v_cont_55', 0.0)
(0.985358028188035, 46, 'v_cont_48', 47, 'v_cont_49', 0.0)
(-0.8034519072989161, 6, 'v_cont_8', 53, 'v_cont_55', 0.0)
(0.8132433874202087, 18, 'v_cont_20', 22, 'v_cont_24', 0.0)
(0.981898517014815, 70, 'v_cont_72', 76, 'v_cont_78', 0.0)
(0.981898517014815, 76, 'v_cont_78', 70, 'v_cont_72', 0.0)
(0.8132433874202087, 22, 'v_cont_24', 18, 'v_cont_20', 0.0)
(0.9776158180285813, 22, 'v_cont_24', 15, 'v_cont_17', 0.0)
(0.9438503379649609, 32, 'v_cont_34', 33, 'v_cont_35', 0.0)
(0.8814456236410259, 74, 'v_cont_76', 30, 'v_cont_32', 0.0)
(0.8226685567453994, 74, 'v_cont_76', 52, 'v_cont_54', 0.0)
(0.8814456236410259, 30, 'v_cont_32', 74, 'v_cont_76', 0.0)
(0.9438503379649609, 33, 'v_cont_35', 32, 'v_cont_34', 0.0)
(0.8137998648445638, 80, 'v_cont_82', 31, 'v_cont_33', 0.0)
(0.8361952961320445, 35, 'v_cont_37', 20, 'v_cont_22', 0.0)
(0.8137998648445638, 31, 'v_cont_33', 80, 'v_cont_82', 0.0)
(0.9757255165940261, 41, 'v_cont_43', 44, 'v_cont_46', 0.0)
(0.9223946342691782, 41, 'v_cont_43', 43, 'v_cont_45', 0.0)
(0.9757255165940261, 44, 'v_cont_46', 41, 'v_cont_43', 0.0)
(0.8361952961320445, 20, 'v_cont_22', 35, 'v_cont_37', 0.0)
(0.8107003756835877, 5, 'v_cont_7', 53, 'v_cont_55', 0.0)
(-0.8034519072989161, 53, 'v_cont_55', 6, 'v_cont_8', 0.0)
(0.8107003756835877, 53, 'v_cont_55', 5, 'v_cont_7', 0.0)
24
729
729
729
```

```
9
```

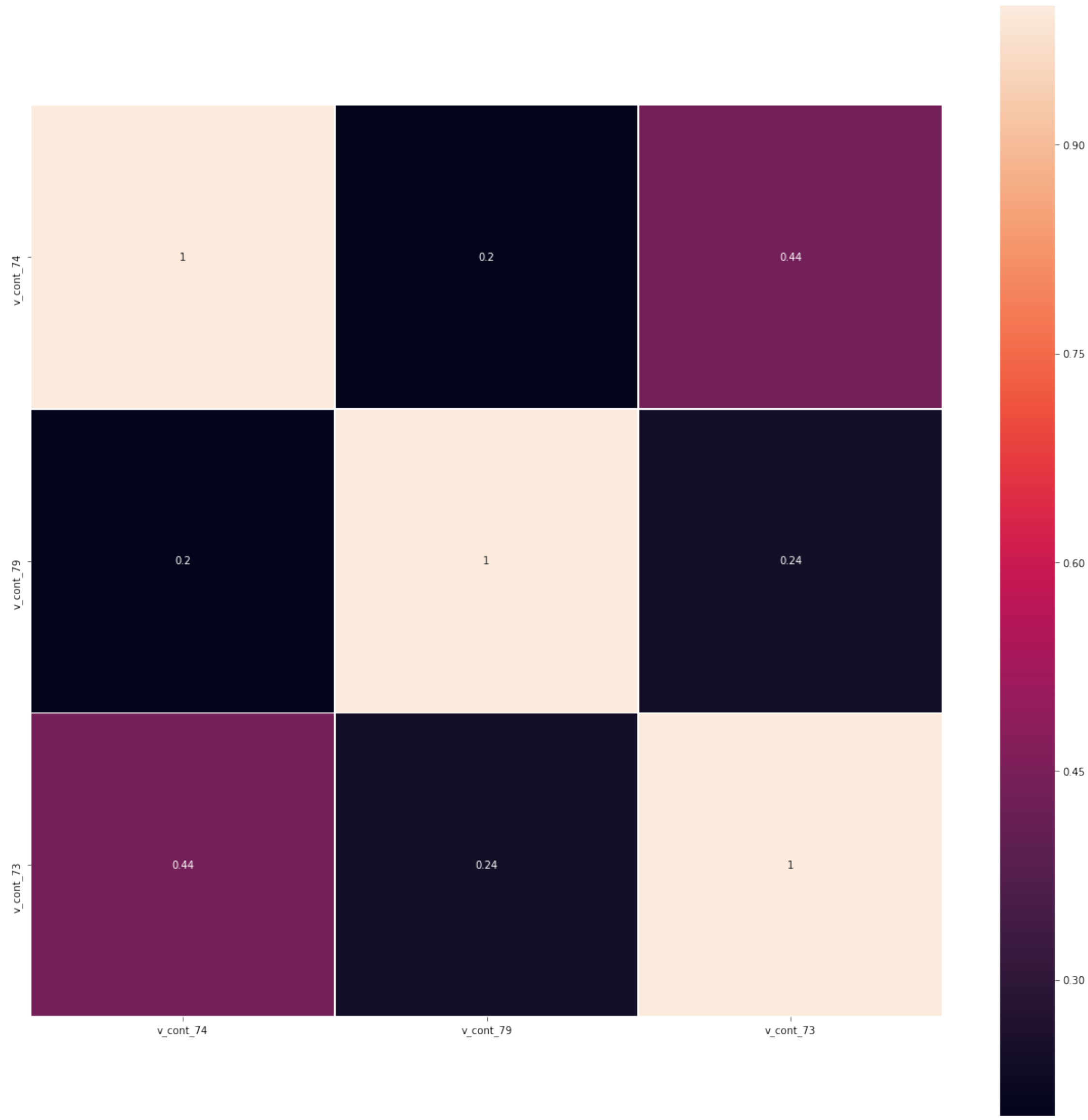
```
Out[33]: '\nassert(len(HighCorrTuples)+len(LowCorrTuples)==len(Corr))\nUncorrelatedFeatureIndi
```

```
In [34]: Indfor ={}
         for ind,x in enumerate(FinalFeatureSet):
             if(x[1] not in Indfor):
                 Indfor[x[1]] = x[1]
         print len(Indfor)
         IndforCor =[]
         for key,val in Indfor.items():
             IndforCor.append(val)
         Cols=[]
         FinalcorMatrix = np.zeros((len(Indfor),len(Indfor)))
         for i in range(FinalcorMatrix.shape[0]):
             Cols.append(X.columns[IndforCor[i]])
             for j in range(FinalcorMatrix.shape[1]):
                 t1,t2 = pearsonr(Matrix[:,IndforCor[i]],Matrix[:,IndforCor[j]])
                 FinalcorMatrix[i][j] = t1

         fig, ax = plt.subplots(figsize=(20,20))
         sb.heatmap(FinalcorMatrix,
                 xticklabels=Cols,
                 yticklabels=Cols,linewidths=1,annot=True, ax=ax,square =True)

         plt.show()
```

```
3
```




```
In [35]: print IndforCor
```

```
[72, 77, 71]
```

1.5.3 Linear regression on the remaining features

```
In [36]: FinalTrainSetForTrain = np.zeros((3,Matrix.shape[0]))
         for ind,k in enumerate(IndforCor):
             FinalTrainSetForTrain[ind] = (Matrix[:,k])
         FinalTrainSetForTrain = FinalTrainSetForTrain.T
         X_train,X_test,Y_train,Y_test = train_test_split(FinalTrainSetForTrain,Y,test_size = (
```

```python
#Normal Linear Regression

clf = linear_model.LinearRegression(normalize = True)
clf.fit(X_train,Y_train)
Y_pred = clf.predict(X_test)
Weights = clf.coef_
print "RMSE %.2f" % math.sqrt(mean_squared_error(Y_test,Y_pred))
print('Variance score: %.2f' % r2_score(Y_test, Y_pred))

# Linear Regression With Ridge regularisation
clf = linear_model.Ridge(alpha = 0.1,normalize = True)
clf.fit(X_train,Y_train)
Y_pred = clf.predict(X_test)
print "RMSE %.2f" % math.sqrt(mean_squared_error(Y_test,Y_pred))
print('Variance score: %.2f' % r2_score(Y_test, Y_pred))


# Linear Regression With Lasso regularisation
clf = linear_model.Lasso(alpha = 0.1,normalize = True)
clf.fit(X_train,Y_train)
Y_pred = clf.predict(X_test)
print "RMSE %.2f" % math.sqrt(mean_squared_error(Y_test,Y_pred))
print('Variance score: %.2f' % r2_score(Y_test, Y_pred))
```

```
RMSE 0.17
Variance score: 0.42
RMSE 0.17
Variance score: 0.42
RMSE 0.22
Variance score: -0.01
```