Develop PD model, by Brent Oeyen

The goal of this code is to compare the following Probability of Default models

- An IRLS logistic regression with a classic transformation of variable;
- An elastic net logistic regression with raw numeric input variables, calibrated with scipy minize;
- A Keras DNN binary classification algorithm with raw numeric input variables transformed with a standard score; and
- A Keras elastic net with raw numeric input variables transformed with a standard score.

The following packages are used in the implementation:

In [1]:
```
import os
import lime
import lime.lime_tabular
import pandas                           as pd
import numpy                            as np
import scipy.optimize                   as optimize
import scipy.stats                      as st
from keras.models                       import Sequential
from keras.layers                       import Dense
from keras.wrappers.scikit_learn        import KerasClassifier
from keras.layers                       import Dropout
from keras.constraints                  import maxnorm
from keras.optimizers                   import SGD
from sklearn.model_selection            import cross_val_score
from sklearn.preprocessing              import LabelEncoder
from sklearn.model_selection            import StratifiedKFold
from sklearn.preprocessing              import StandardScaler
from sklearn.pipeline                   import Pipeline
from sklearn.calibration                import CalibratedClassifierCV
from sklearn.linear_model               import LogisticRegression
from sklearn.model_selection            import train_test_split
from Codes.PD.PD_tests                  import *
from Codes.PD.Logistic_regression import *
from Codes.PD.Feature_engineering import *
```

```
Using TensorFlow backend.
```

Load dataset, source: "[http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/ (http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/)](http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/)".

In [2]:
```
location               = os.getcwd() + '/Data/german data numeric'
dataframe              = pd.read_csv(location, header=None, delimiter=r"\s+")
```

Split data set into input (X) input variables and binary (Y) output variable:

In [3]:
```
X                      = dataframe.values[:,0:24].astype(float)
dataframe.iloc[:, 24] = dataframe.iloc[:, 24]==2
Y                      = dataframe.values[:,24].astype(float)
```

Transform input variables with a logit transformation per quantiles:

In [4]:
```
X_scaled       = pd.DataFrame({'X0': FE().transform_(dataframe.iloc[:, [0,24
]])})
for i in range(1, dataframe.shape[1]-1):
 X_scaled['X'+str(i)] = FE().transform_(dataframe.iloc[:, [i,24]])
X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, Y, test_size=0.2
, random_state=42)
X_tr    , X_tes , Y_tr    , Y_tes   = train_test_split(X       , Y, test_size=0.2
, random_state=42)
```

Output rank of the variance of the transformed and not transformed variables as an indication of which variables contain a lot of information (high value indicates a lot of ranking power):

In [17]:
```
print(X_train.var().rank())
print(pd.DataFrame(X_tr).var().rank())
```

```
X0     24.0
X1     23.0
X2     21.0
X3     20.0
X4     22.0
X5     18.0
X6     16.0
X7     13.0
X8     17.0
X9     19.0
X10     6.5
X11    15.0
X12     6.5
X13     6.5
X14     6.5
X15     6.5
X16     6.5
X17     6.5
X18     6.5
X19     6.5
X20     6.5
X21     6.5
X22    14.0
X23     6.5
dtype: float64
0     20.0
1     23.0
2     17.0
3     24.0
4     21.0
5     19.0
6     14.0
7     18.0
8     16.0
9     22.0
10    15.0
11    13.0
12     6.0
13    12.0
14     2.0
15     9.0
16     5.0
17     4.0
18     3.0
19     7.0
20    10.0
21     1.0
22     8.0
23    11.0
dtype: float64
```

Verify whether they the transformation of variable was succesful by comparing the rank correlation:

In [19]:
```
print("Raw input variable X0: ", st.kendalltau(Y, X[:, 0]))
print('Transformed input variable X0:', st.kendalltau(Y, X_scaled.iloc[:, 0]))
```

```
Raw input variable X0:  KendalltauResult(correlation=-0.32341244264035907, pv
alue=3.9303648079470383e-28)
Transformed input variable X0: KendalltauResult(correlation=0.054551530003972
955, pvalue=0.07699078300834665)
```

Create a Master Scale of 22 ratings mapped to equidistant PDs in logit space:

In [20]:
```
ratings              = 22                                        #Number of rat
ing grades
PD_min               = 0.0003                                    #Minimum PD va
lue (regulatory threshold)
slope                = (np.log((2-PD_min)/PD_min)-0)/(ratings-1)  #Slope between
min PD value and default in logit space
MS                   = 2/(1+np.exp(slope*pd.Series(list(range(ratings)))))
idx                  = pd.IntervalIndex.from_arrays(MS[1:].append(pd.Series(0)),
MS, closed='left')
```

IRLS logistic regression with a classic transformation of variable

Calibrate model:

In [23]:
```
betas_start                 = np.append(np.log(Y_train.mean()/(1-Y_train.mean
())), np.zeros(12))
betas_IRLS, y_train_IRLS = logistic_regression().IRLS(betas_start, np.append(n
p.ones([len(X_train.index), 2]), \
                              X_train.iloc[:, np.r_[0:10, 11, 22]], axis=1)[:, 1
:], Y_train)
X_IRLS                      = np.append(np.ones([len(X_scaled.index), 2]), \
                              X_scaled.iloc[:, np.r_[0:10, 11, 22]], axis=1)[:, 1
:]
IRLS_all                    = pd.DataFrame({'PD': 1/(1+np.exp(X_IRLS.dot(-betas_I
RLS)))})
IRLS_all['Y']               = Y
IRLS_all['rating_PD']       = MS[idx.get_indexer(IRLS_all.PD)].values
```

```
IRLS converged after 4iterations.
```

Output results for a regression with the 12 highest correlated variables:

In [25]:
```
print("Performance metrics: [AUC all %.2f%%]" % (PD_tests().AUC(IRLS_all.Y, IR
LS_all.rating_PD,0)[0]*100))
print("Coefficients regression:", betas_IRLS)
dummy                 = PD_tests().Jeffrey(IRLS_all, 'rating_PD', 'PD', 'Y')
print('Jeffrey test')
print(dummy.iloc[:, [0, 3, 6,  11]])
```

```
Performance metrics: [AUC all 55.52%]
Coefficients regression: [ 5.11903847e-01  1.22865455e-01  7.61954753e-02 -4.
84788648e-04
  3.97941005e-01 -2.36673779e-01  4.01736488e-01  4.21172675e-01
 -2.69846974e+00 -2.30495957e-01 -2.84813406e-01 -1.13340218e+00
  1.52628712e+00]
Jeffrey test
                    rating_PD        PD         Y     p_val
                        count      mean      mean
rating_PD
0.14953859341903025       1.0  0.121110  0.000000  0.433984
0.21890161217657667      90.0  0.199081  0.255556  0.092699
0.31496223179993454     497.0  0.272987  0.261569  0.714356
0.4426995169025332      402.0  0.357164  0.360697  0.439347
0.603682514062032 1      10.0  0.462189  0.200000  0.955955
Portfolio              1000.0  0.301915  0.300000  0.550696
```

Given the poor performance of the scaled input variables, the raw input variables are used instead:

In [26]:
```
betas_start              = np.append(np.log(Y_tr.mean()/(1-Y_tr.mean())), np.z
eros(12))
betas_IRLS, y_train_IRLS = logistic_regression().IRLS(betas_start, np.append(n
p.ones([len(X_tr), 2]), \
                           X_tr[:, np.r_[0:10, 11, 22]], axis=1)[:, 1:], Y_tr)
X_IRLS                   = np.append(np.ones([len(X), 2]), \
                           X[:, np.r_[0:10, 11, 22]], axis=1)[:, 1:]
IRLS_all                 = pd.DataFrame({'PD': 1/(1+np.exp(X_IRLS.dot(-betas_I
RLS)))})
IRLS_all['Y']            = Y
IRLS_all['rating_PD']    = MS[idx.get_indexer(IRLS_all.PD)].values
print("Performance metrics: [AUC all %.2f%%]" % (PD_tests().AUC(IRLS_all.Y, IR
LS_all.rating_PD,0)[0]*100))
print("Coefficients regression:", betas_IRLS)
dummy                = PD_tests().Jeffrey(IRLS_all, 'rating_PD', 'PD', 'Y')
print('Jeffrey test')
print(dummy.iloc[:, [0, 3, 6, 11]])
```

```
IRLS converged after 5iterations.
Performance metrics: [AUC all 78.45%]
Coefficients regression: [ 1.65181869 -0.52786127  0.02578897 -0.4140539   0.
00296884 -0.14828576
 -0.13942045 -0.21361935  0.059363    0.25878264 -0.02437638  0.35144775
 -0.03601741]
Jeffrey test
                     rating_PD        PD         Y     p_val
                         count      mean      mean
rating_PD
0.01966854325104041        3.0  0.016135  0.000000  0.255315
0.029760560578478736      14.0  0.025735  0.000000  0.611359
0.04491339548733918       26.0  0.037233  0.038462  0.417974
0.06751703372078455       53.0  0.056503  0.056604  0.462350
0.10090929621852611       91.0  0.086010  0.109890  0.202839
0.14953859341903025      126.0  0.124423  0.103175  0.759961
0.21890161217657667      150.0  0.183381  0.166667  0.695860
0.31496223179993454      149.0  0.262696  0.228188  0.830484
0.4426995169025332       134.0  0.379745  0.417910  0.181017
0.6036825140620321       133.0  0.523686  0.571429  0.135040
0.7933816324488658       103.0  0.679866  0.669903  0.590438
1.0                       18.0  0.842580  0.722222  0.912605
Portfolio               1000.0  0.300253  0.300000  0.505136
```

Elastic net logistic regression with raw numeric input variables

In [29]:
```
solution               = optimize.minimize(fun=logistic_regression().el_logicreg
, x0=betas_IRLS, args=(Y_tr, \
                         X_tr[:, np.r_[0:10, 11, 22]], 0.3, 0.5), method='Nelder
-Mead', options={"maxiter":5000})
X_LL                   = np.append(np.ones([len(X), 2]), X[:, np.r_[0:10, 11, 22
]], axis=1)[:, 1:]
LL_all                 = pd.DataFrame({'PD': 1/(1+np.exp(X_LL.dot(-solution.x
)))})
LL_all['Y']            = Y
LL_all['rating_PD']    = MS[idx.get_indexer(LL_all.PD)].values
```

Output results for a regression with the 12 highest correlated variables:

In [31]:
```
print("Results logistic regression ML: Elastic net (Lambda=0.3, L1 ratio=0.5)"
)
print("Performance metrics: [AUC all %.2f%%]" % (PD_tests().AUC(LL_all.Y, LL_a
ll.rating_PD,0)[0]*100))
print("Coefficients regression:", solution.x)
dummy                   = PD_tests().Jeffrey(LL_all, 'rating_PD', 'PD', 'Y')
print('Jeffrey test')
print(dummy.iloc[:, np.r_[0, 3, 6, 11]])
```

```
Results logistic regression ML: Elastic net (Lambda=0.3, L1 ratio=0.5)
Performance metrics: [AUC all 66.13%]
Coefficients regression: [-5.59696913e-01 -9.26444379e-02  3.18177176e-02  8.
19057293e-08
  9.69142257e-03 -7.16970386e-07  3.80225097e-03  1.56004475e-07
  2.97328529e-02  9.33491415e-04 -3.60486743e-02 -2.27130167e-03
 -7.78605630e-02]
Jeffrey test
                    rating_PD         PD         Y     p_val
                        count       mean      mean
rating_PD
0.04491339548733918       1.0   0.040970  0.000000  0.255947
0.06751703372078455      10.0   0.061351  0.100000  0.255085
0.10090929621852611      39.0   0.085639  0.102564  0.327921
0.14953859341903025     105.0   0.128623  0.152381  0.228300
0.21890161217657667     224.0   0.187454  0.205357  0.243235
0.31496223179993454     299.0   0.265383  0.304348  0.065064
0.4426995169025332      183.0   0.373943  0.360656  0.642732
0.6036825140620321       97.0   0.510901  0.515464  0.464503
0.7933816324488658       37.0   0.666192  0.594595  0.822911
1.0                       5.0   0.827102  0.800000  0.615435
Portfolio              1000.0   0.285613  0.300000  0.156919
```

Keras DNN with raw numeric input variables transformed with a standard score

In [32]:
```
def create_binary():
        # create model
        model = Sequential()
        model.add(Dense(24, input_dim=24, activation='relu'))
        model.add(Dropout(0.2))
        model.add(Dense(24, activation='relu'))
        model.add(Dropout(0.2))
        model.add(Dense(1, activation='sigmoid'))
        # Compile model
        model.compile(loss='binary_crossentropy', optimizer='adam', metrics=[
'accuracy'])
        return model
estimators            = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasClassifier(build_fn=create_binary, epochs=80, b
atch_size=16, verbose=0)))
pipeline              = Pipeline(estimators)
pipeline.fit(X_tr, Y_tr)
kfold                 = StratifiedKFold(n_splits=10, shuffle=True)
results               = cross_val_score(pipeline, X, Y, cv=kfold)
b_pred                = pd.DataFrame(pipeline.predict_proba(X_tes))
b_pred.columns        = ['ID', 'PD']
b_pred['Y']           = Y_tes
b_pred['rating_PD'] = MS[idx.get_indexer(b_pred.PD)].values
b_all                 = pd.DataFrame(pipeline.predict_proba(X))
b_all.columns         = ['ID', 'PD']
b_all['Y']            = Y
b_all['rating_PD']  = MS[idx.get_indexer(b_all.PD)].values
```

Output results with all input variables being used:

In [33]:
```
print("Performance metrics: Mean Accuracy CV %.2f%% (STD %.2f%%) [AUC test %.2
f%%] [AUC all %.2f%%]" \
% (results.mean()*100, results.std()*100, PD_tests().AUC(b_pred.Y, b_pred.rati
ng_PD,0)[0]*100, \
   PD_tests().AUC(b_all.Y, b_all.rating_PD,0)[0]*100))
dummy                    = PD_tests().Jeffrey(b_all, 'rating_PD', 'PD', 'Y')
print('Jeffrey test')
print(dummy.iloc[:, [0, 3, 6, 11]])
```

```
Performance metrics: Mean Accuracy CV 75.60% (STD 3.69%) [AUC test 78.12%] [A
UC all 91.05%]
Jeffrey test
                      rating_PD        PD         Y     p_val
                          count      mean      mean
rating_PD
0.00029999999999999976       36.0  0.000081  0.000000  0.061143
0.0004562208354469178         8.0  0.000375  0.000000  0.062677
0.0006937632793232519        14.0  0.000574  0.000000  0.101781
0.0010549227058496641        10.0  0.000865  0.000000  0.105975
0.0016039437457623122        19.0  0.001308  0.000000  0.177620
0.002438347229665251         15.0  0.002026  0.000000  0.196403
0.0037060190379897217        14.0  0.002993  0.142857  0.000092
0.005630882685189334         15.0  0.004583  0.000000  0.291842
0.00855121588022675          27.0  0.006826  0.000000  0.458799
0.012976261070719905         23.0  0.011057  0.043478  0.082398
0.01966854325104041          30.0  0.015950  0.000000  0.676020
0.029760560578478736         36.0  0.024195  0.055556  0.114437
0.04491339548733918          38.0  0.037048  0.000000  0.910767
0.06751703372078455          63.0  0.055747  0.047619  0.579663
0.10090929621852611          62.0  0.085026  0.032258  0.946266
0.14953859341903025          72.0  0.124531  0.097222  0.751179
0.21890161217657667          64.0  0.182193  0.187500  0.442752
0.31496223179993454          79.0  0.262082  0.215190  0.827911
0.4426995169025332           95.0  0.375557  0.368421  0.553632
0.603682514062032l          106.0  0.526726  0.641509  0.008564
0.7933816324488658          112.0  0.698948  0.821429  0.001575
1.0                          62.0  0.873724  0.951613  0.021739
Portfolio                  1000.0  0.277463  0.300000  0.056588
```

Keras elastic net with raw numeric input variables transformed with a standard score

In [36]:
```
LR                       = LogisticRegression(C=0.3, penalty='elasticnet', solver=
'saga', \
                                              l1_ratio=0.5, max_iter=1000, tol=0.00
1)
scaler                   = StandardScaler()
scaler.fit(X_tr)
LR.fit(scaler.transform(X_tr), Y_tr)
LR_pred                  = pd.DataFrame(LR.predict_proba(scaler.transform(X_tes)))
LR_pred.columns          = ['ID', 'PD']
LR_pred['Y']             = Y_tes
LR_pred['rating_PD']= MS[idx.get_indexer(LR_pred.PD)].values
LR_all                   = pd.DataFrame(LR.predict_proba(scaler.transform(X)))
LR_all.columns           = ['ID', 'PD']
LR_all['Y']              = Y
LR_all['rating_PD'] = MS[idx.get_indexer(LR_all.PD)].values
results                  = cross_val_score(LR, X, Y, cv=kfold)
```

Output results with all input variables being used:

In [37]:
```
print("Results logistic regression: Elastic net (Lambda=0.3, L1 ratio=0.5)")
print("Performance metrics: Mean Accuracy CV %.2f%% (STD %.2f%%) [AUC test %.2
f%%] [AUC all %.2f%%]" \
% (results.mean()*100, results.std()*100, PD_tests().AUC(LR_pred.Y, LR_pred.ra
ting_PD,0)[0]*100, \
   PD_tests().AUC(LR_all.Y, LR_all.rating_PD,0)[0]*100))
print("Coefficients regression:", LR.coef_)
dummy                = PD_tests().Jeffrey(LR_all, 'rating_PD', 'PD', 'Y')
print('Jeffrey test')
print(dummy.iloc[:, [0, 3, 6, 11]])
```

```
Results logistic regression: Elastic net (Lambda=0.3, L1 ratio=0.5)
Performance metrics: Mean Accuracy CV 74.90% (STD 3.59%) [AUC test 80.51%] [A
UC all 81.14%]
Coefficients regression: [[-6.57062014e-01  3.06144488e-01 -3.95271636e-01
1.51811461e-01
  -2.35458802e-01 -1.59515437e-01 -1.11558270e-01  1.88798066e-02
   1.64084969e-01 -2.55788551e-01 -2.20602169e-01  1.41707365e-01
   1.77686243e-04 -7.07621914e-02 -2.10568558e-01  2.85916055e-01
  -2.45430742e-01  2.02501524e-01  1.07027461e-01  1.97936468e-02
  -1.89891235e-01 -1.00384192e-01 -4.87961524e-02  0.00000000e+00]]
Jeffrey test
                     rating_PD        PD         Y     p_val
                         count      mean      mean          
rating_PD                                                   
0.00855121588022675        1.0  0.006152  0.000000  0.099766
0.012976261070719905       2.0  0.012371  0.000000  0.187656
0.01966854325104041        6.0  0.016602  0.000000  0.352800
0.029760560578478736      21.0  0.024902  0.000000  0.699465
0.04491339548733918       40.0  0.036756  0.000000  0.917488
0.06751703372078455       60.0  0.056197  0.050000  0.549392
0.10090929621852611       92.0  0.084127  0.065217  0.733111
0.14953859341903025      105.0  0.122607  0.142857  0.256202
0.21890161217657667      145.0  0.182605  0.151724  0.831822
0.31496223179993454      130.0  0.261580  0.253846  0.573450
0.4426995169025332       128.0  0.373936  0.359375  0.630603
0.6036825140620321       134.0  0.518180  0.567164  0.128119
0.7933816324488658       106.0  0.687518  0.735849  0.140962
1.0                       30.0  0.843481  0.700000  0.978192
Portfolio               1000.0  0.302073  0.300000  0.555007
```

Conclusion

Based on the results of the credit data set, transforming the input data wasn't successful on a small sample population (1K). The imbalance of the observed defaults and performing credit did not create calibration issues.

The Keras library is straight forward to use and requires little development from the user. Albeit fine tuning the configuration parameters to run the models will require some time and experience with both the tool and modelling a given dependent variable. In addition, the Keras library outperforms a cinch implementation of the logistic regression's IRLS and ML elastic net method. The DNN model outperforms the logistic regression based on the AUC of the total population (training and test data) but doesn't on the training set. In addition, the Cross Validation (CV) score of the training set is comparable while the logistic regression produces more stable results.

Considering the comparable performance between DNN and elastic net and the apprehensible results of the elastic net in contrast of the complexity of understanding the probability calculation of the DNN algorithm (not covered in this example), logistic regression with an elastic net is a preferred option for PD models.

Appendix

Understanding the outliers between DNN and EN

The regression coefficients explain the impact of the input variable's EN's PD. While for DNN we will use the LIME method.