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 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 *
```

```
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 [6]: def transform(df, j):
         df.columns      = ['var', 'binary']
         try:
          df['decile'] = pd.qcut(df['var'], q=np.arange(1, 51)/50, labels=False)
          df           = pd.merge(df.astype(float), df.groupby('decile', as_index=Fals
        e)['binary'].mean(), \
                                   on='decile', how='inner')
         except:
          df           = pd.merge(df.astype(float), df.groupby('var'    , as_index=Fals
        e)['binary'].mean(), \
                                   on='var'    , how='inner')
         pi            = df.iloc[:, -1].fillna(0.5).replace(0, 0.00001)
         return pd.DataFrame({'X'+str(j): np.log(pi/(1-pi)).replace(np.inf, 15)})
        X_scaled        = transform(dataframe.iloc[:, [0,24]], 1)
        for i in range(1, dataframe.shape[1]-1):
         X_scaled['X'+str(i+1)] = transform(dataframe.iloc[:, [i,24]], i+1)
        X_scaled        = X_scaled.fillna(0)
        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 variance of each transformed input variable as an indication of which variables contain a lot of information:

```
In [7]: print(X_train.var())
```

```
X1       0.762004
X2       3.421997
X3       0.295841
X4      14.081804
X5       0.226596
X6       0.085566
X7       0.043210
X8       0.003601
X9       0.110774
X10      1.683886
X11      0.053565
X12      0.013477
X13      0.000043
X14      0.006430
X15      0.056956
X16      0.041302
X17      0.065301
X18      0.000003
X19      0.014833
X20      0.036471
X21      0.079953
X22      0.000149
X23      0.002296
X24      0.000869
dtype: float64
```

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

```
In [8]: print("Raw input variable X2: ", st.kendalltau(Y, X[:, 1]))
        print('Transformed input variable X2:', st.kendalltau(Y, X_scaled.iloc[:, 1]))
```

```
Raw input variable X2:  KendalltauResult(correlation=0.17609245525504136, pva
lue=7.975280722434196e-11)
Transformed input variable X2: KendalltauResult(correlation=0.013532833811191
117, pvalue=0.6174826267368242)
```

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

```
In [9]: 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 [10]: betas_start                = np.append(np.log(Y_train.mean()/(1-Y_train.mean
         ())), np.zeros(8))
         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_[0:4, 7:9, 16, 20]], axis=1)
         [:, 1:], Y_train)
         X_IRLS                     = np.append(np.ones([len(X_scaled.index), 2]), \
                                      X_scaled.iloc[:, np.r_[0:4, 7:9, 16, 20]], 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 8 highest correlated variables:

```
In [11]: 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[:, np.r_[0, 3, 6,  10:12]])
```

```
Performance metrics: [AUC all 53.59%]
Coefficients regression: [ 0.63104141 -0.00249486 -0.02409827  0.21552432 -0.
00565019  0.70273931
  0.23757129  0.47568141  0.12372487]
Jeffrey test
                    rating_PD        PD         Y        H0     p_val
                        count      mean      mean
rating_PD
0.21890161217657667       2.0  0.198208  0.500000  0.198208  0.140557
0.31496223179993454     529.0  0.273935  0.270321  0.273935  0.571111
0.4426995169025332      469.0  0.333599  0.332623  0.333599  0.515712
Portfolio              1000.0  0.805742  1.102944  0.301766  0.546622
```

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

```
In [20]: betas_start                 = np.append(np.log(Y_tr.mean()/(1-Y_tr.mean())), np.z
eros(8))
betas_IRLS, y_train_IRLS = logistic_regression().IRLS(betas_start, np.append(n
p.ones([len(X_tr), 2]), \
                         X_tr[:, np.r_[0:4, 7:9, 16, 20]], axis=1)[:, 1:], Y
_tr)
X_IRLS                   = np.append(np.ones([len(X), 2]), \
                         X[:, np.r_[0:4, 7:9, 16, 20]], 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[:, np.r_[0, 3, 6,  10:12]])
```

```
IRLS converged after 5iterations.
Performance metrics: [AUC all 77.88%]
Coefficients regression: [ 0.77847929 -0.56161866  0.0246094  -0.36420994  0.
00514031 -0.0544452
  0.20944244 -1.17560474 -0.51219642]
Jeffrey test
                     rating_PD        PD         Y        H0     p_val
                         count      mean      mean
rating_PD
0.029760560578478736       4.0  0.027608  0.000000  0.027608  0.374643
0.04491339548733918       26.0  0.039108  0.000000  0.039108  0.852174
0.06751703372078455       63.0  0.056960  0.031746  0.056960  0.800781
0.10090929621852611       96.0  0.083958  0.083333  0.083958  0.488103
0.14953859341903025      115.0  0.125885  0.130435  0.125885  0.428017
0.21890161217657667      135.0  0.184365  0.177778  0.184365  0.569288
0.31496223179993454      162.0  0.263878  0.253086  0.263878  0.617424
0.4426995169025332       163.0  0.377768  0.404908  0.377768  0.236454
0.603682514062032l       128.0  0.526771  0.531250  0.526771  0.460227
0.7933816324488658        90.0  0.688752  0.677778  0.688752  0.594176
1.0                       18.0  0.850305  0.833333  0.850305  0.609116
Portfolio               1000.0  3.225356  3.123647  0.301186  0.530763
```

Elastic net logistic regression with raw numeric input variables

In [55]:
```
solution                 = optimize.minimize(fun=logistic_regression().el_logicreg
, x0=betas_IRLS, args=(Y_tr, \
                           X_tr[:, np.r_[0:4, 7:9, 16, 20]], 0.3, 0.5), method='Ne
lder-Mead', options={"maxiter":5000})
X_LL                     = np.append(np.ones([len(X), 2]), X[:, np.r_[0:4, 7:9, 16
, 20]], 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 8 highest correlated variables:

In [57]:
```
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,  10:12]])
```

```
Results logistic regression ML: Elastic net (Lambda=0.3, L1 ratio=0.5)
Performance metrics: [AUC all 72.35%]
Coefficients regression: [ 1.34782851e+00 -3.40238037e-01  5.28438161e-02 -4.
65549331e-01
 -1.58887734e-02 -1.50196892e-01 -1.39382220e-04  9.86211126e-10
 -1.51418317e-01]
Jeffrey test
                    rating_PD        PD         Y        H0     p_val
                        count      mean      mean
rating_PD
0.04491339548733918       3.0  0.036647  0.333333  0.036647  0.023039
0.06751703372078455       5.0  0.063221  0.400000  0.063221  0.009732
0.10090929621852611      28.0  0.091162  0.035714  0.091162  0.849184
0.14953859341903025      87.0  0.123939  0.080460  0.123939  0.895987
0.21890161217657667     162.0  0.185623  0.154321  0.185623  0.847680
0.31496223179993454     184.0  0.263588  0.228261  0.263588  0.862264
0.4426995169025332      204.0  0.376796  0.225490  0.376796  0.999998
0.603682514062032l      195.0  0.514633  0.482051  0.514633  0.818697
0.7933816324488658      119.0  0.685217  0.605042  0.685217  0.968352
1.0                      13.0  0.841622  0.769231  0.841622  0.776087
Portfolio              1000.0  3.182449  3.313904  0.362034  0.999982
```

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

In [58]:
```
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 [59]: 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[:, np.r_[0, 3, 6,  10:12]])
```

```
Performance metrics: Mean Accuracy CV 74.90% (STD 2.88%) [AUC test 78.31%] [A
UC all 92.09%]
Jeffrey test
                       rating_PD        PD         Y        H0     p_val
                           count      mean      mean
rating_PD
0.00029999999999999976      27.0  0.000120  0.000000  0.000120  0.064528
0.0004562208354469178        5.0  0.000394  0.000000  0.000394  0.051319
0.0006937632793232519       14.0  0.000580  0.000000  0.000580  0.102292
0.0010549227058496641       15.0  0.000922  0.000000  0.000922  0.133248
0.0016039437457623122       14.0  0.001259  0.000000  0.001259  0.150303
0.00243834722966525l        24.0  0.001964  0.041667  0.001964  0.007378
0.0037060190379897217       20.0  0.003030  0.000000  0.003030  0.274099
0.005630882685189334        19.0  0.004764  0.052632  0.004764  0.019101
0.00855121588022675         28.0  0.007344  0.035714  0.007344  0.061640
0.012976261070719905        23.0  0.010860  0.043478  0.010860  0.080403
0.01966854325104041         30.0  0.016925  0.033333  0.016925  0.202517
0.029760560578478736        37.0  0.025303  0.000000  0.025303  0.832971
0.04491339548733918         44.0  0.036913  0.045455  0.036913  0.338925
0.06751703372078455         67.0  0.055981  0.059701  0.055981  0.416403
0.10090929621852611         63.0  0.084232  0.031746  0.084232  0.947688
0.14953859341903025         60.0  0.124943  0.133333  0.124943  0.403778
0.21890161217657667         79.0  0.180652  0.088608  0.180652  0.988616
0.31496223179993454         79.0  0.266653  0.215190  0.266653  0.850006
0.4426995169025332          84.0  0.373171  0.416667  0.373171  0.204085
0.603682514062032l          82.0  0.516930  0.609756  0.516930  0.045744
0.7933816324488658          87.0  0.695557  0.885057  0.695557  0.000015
1.0                         99.0  0.885664  0.939394  0.885664  0.037534
Portfolio                 1000.0  3.294161  3.631731  0.277586  0.057593
```

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

```
In [63]: LR                   = LogisticRegression(C=0.3, penalty='elasticnet', solver=
'saga', l1_ratio=0.5, \
                                          max_iter=1000, tol=0.001)
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 [64]:
```
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[:, np.r_[0, 3, 6,  10:12]])
```

```
Results logistic regression: Elastic net (Lambda=0.3, L1 ratio=0.5)
Performance metrics: Mean Accuracy CV 75.10% (STD 2.84%) [AUC test 80.51%] [A
UC all 81.15%]
Coefficients regression: [[-0.65714083  0.30592054 -0.39515111  0.15154385 -
0.23556821 -0.15974222
  -0.11155633  0.01905668  0.16390315 -0.25583818 -0.22045609  0.14207899
   0.          -0.07077531 -0.21050127  0.28600726 -0.2452701   0.20259282
   0.10702668  0.01964218 -0.19007338 -0.10013797 -0.04893568  0.          ]]
Jeffrey test
```

	rating_PD	PD	Y	H0	p_val
	count	mean	mean		
rating_PD					
0.00855121588022675	1.0	0.006156	0.000000	0.006156	0.099792
0.012976261070719905	2.0	0.012371	0.000000	0.012371	0.187656
0.01966854325104041	6.0	0.016613	0.000000	0.016613	0.352909
0.029760560578478736	21.0	0.024908	0.000000	0.024908	0.699527
0.04491339548733918	40.0	0.036768	0.000000	0.036768	0.917541
0.06751703372078455	61.0	0.056392	0.049180	0.056392	0.564286
0.10090929621852611	91.0	0.084324	0.065934	0.084324	0.725217
0.14953859341903025	105.0	0.122635	0.142857	0.122635	0.256485
0.21890161217657667	145.0	0.182612	0.151724	0.182612	0.831874
0.31496223179993454	130.0	0.261591	0.253846	0.261591	0.573564
0.4426995169025332	128.0	0.373937	0.359375	0.373937	0.630607
0.6036825140620321	134.0	0.518178	0.567164	0.518178	0.128114
0.7933816324488658	106.0	0.687456	0.735849	0.687456	0.140659
1.0	30.0	0.843433	0.700000	0.843433	0.978151
Portfolio	1000.0	3.227374	3.025930	0.302073	0.555003

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.