Classification

October 10, 2021

SL Lab with Python 4: Classification

Statistical Learning (Sejong University)

Date: 2021.10.10 (By: S. M. Riazul Islam)

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

import sklearn.linear_model as skl_lm
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.metrics import confusion_matrix, classification_report, precision_score
from sklearn import preprocessing
from sklearn import neighbors

import statsmodels.api as sm
import statsmodels.formula.api as smf

%matplotlib inline
plt.style.use('seaborn-white')
```

In [8]:
```
# Load Credit Card Dataset
# This dataset is in Excell format
card = pd.read_excel('C:/DataSL/Default.xlsx')
card.head(3)
```

Out[8]:

	Unnamed: 0	default	student	balance	income
0	1	No	No	729.526495	44361.625074
1	2	No	Yes	817.180407	12106.134700
2	3	No	No	1073.549164	31767.138947

In [17]:
```
# factorize() returns two objects: "labels" and "uniques" array.
 # labels - new values for each of the classes
 # uniques - Mapping back to original labels
```

```
        # [0] keeps only the "labels", not the "uniques".

        card['default2'] = card.default.factorize()[0]
        card['student2'] = card.student.factorize()[0]
        card.head(3)
```

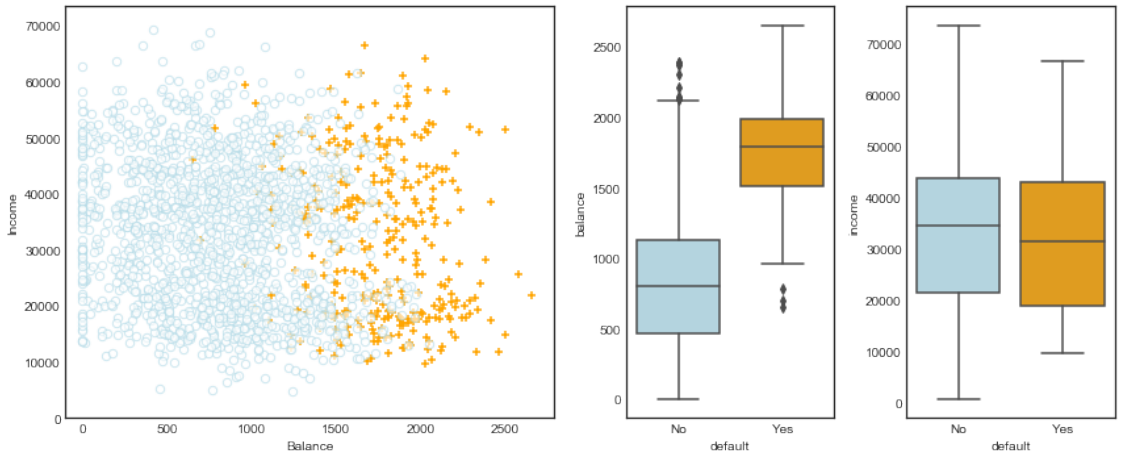
Out[17]:

	Unnamed: 0	default	student	balance	income	default2	student2
0	1	No	No	729.526495	44361.625074	0	0
1	2	No	Yes	817.180407	12106.134700	0	1
2	3	No	No	1073.549164	31767.138947	0	0

Figure 4.1 - Credit Crad Default data set

```
In [19]: fig = plt.figure(figsize=(12,5))
         gs = mpl.gridspec.GridSpec(1, 4)
         ax1 = plt.subplot(gs[0,:-2])
         ax2 = plt.subplot(gs[0,-2])
         ax3 = plt.subplot(gs[0,-1])

         # Take a fraction of the samples where target value (default) is 'no'
         card_no = card[card.default2 == 0].sample(frac=0.15)
         # Take all samples  where target value is 'yes'
         card_yes = card[card.default2 == 1]
         card_ = card_no.append(card_yes)

         ax1.scatter(card_[card_.default == 'Yes'].balance, card_[card_.default == 'Yes'].inco
                     linewidths=1)
         ax1.scatter(card_[card_.default == 'No'].balance, card_[card_.default == 'No'].income
                     edgecolors='lightblue', facecolors='white', alpha=.6)

         ax1.set_ylim(ymin=0)
         ax1.set_ylabel('Income')
         ax1.set_xlim(xmin=-100)
         ax1.set_xlabel('Balance')

         c_palette = {'No':'lightblue', 'Yes':'orange'}
         sns.boxplot('default', 'balance', data=card, orient='v', ax=ax2, palette=c_palette)
         sns.boxplot('default', 'income', data=card, orient='v', ax=ax3, palette=c_palette)
         gs.tight_layout(plt.gcf())

C:\Users\Preload\Anaconda3\lib\site-packages\matplotlib\axes\_base.py:3604: MatplotlibDeprecati
The `ymin` argument was deprecated in Matplotlib 3.0 and will be removed in 3.2. Use `bottom` i
  alternative='`bottom`', obj_type='argument')
C:\Users\Preload\Anaconda3\lib\site-packages\matplotlib\axes\_base.py:3215: MatplotlibDeprecati
The `xmin` argument was deprecated in Matplotlib 3.0 and will be removed in 3.2. Use `left` ins
  alternative='`left`', obj_type='argument')
C:\Users\Preload\Anaconda3\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the
  FutureWarning
```

```
C:\Users\Preload\Anaconda3\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the
  FutureWarning
```



Logistic Regression

Figure 4.2

```
In [38]: #convert "balance" data to the shape of (10000, 1)
         X_train = card.balance.values.reshape(-1,1)
         y = card.default2

         # Creating some test data
         X_test = np.arange(card.balance.min(), card.balance.max())
         X_test=X_test.reshape(-1,1)

In [57]: # Model, Train, and Test
         clf = skl_lm.LogisticRegression(solver='newton-cg')
         # Newton's conjugate gradient

         clf.fit(X_train,y)
         prob = clf.predict_proba(X_test)
             # predict_proba(X): Probability estimates.
             # predict(X): Predicts class labels for samples in X.


         fig, (ax1, ax2) = plt.subplots(1,2, figsize=(12,5))

         # Left plot
         sns.regplot(card.balance, card.default2, order=1, ci=None,
                     scatter_kws={'color':'orange'},
```

```
                    line_kws={'color':'blue', 'lw':2}, ax=ax1)

        # Right plot
        ax2.scatter(X_train, y, color='orange')
        ax2.plot(X_test, prob[:,1], color='red')

        for ax in fig.axes:
            ax.set_ylabel('Probability of default')
            ax.set_xlabel('Balance')
```

```
C:\Users\Preload\Anaconda3\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the
  FutureWarning
```

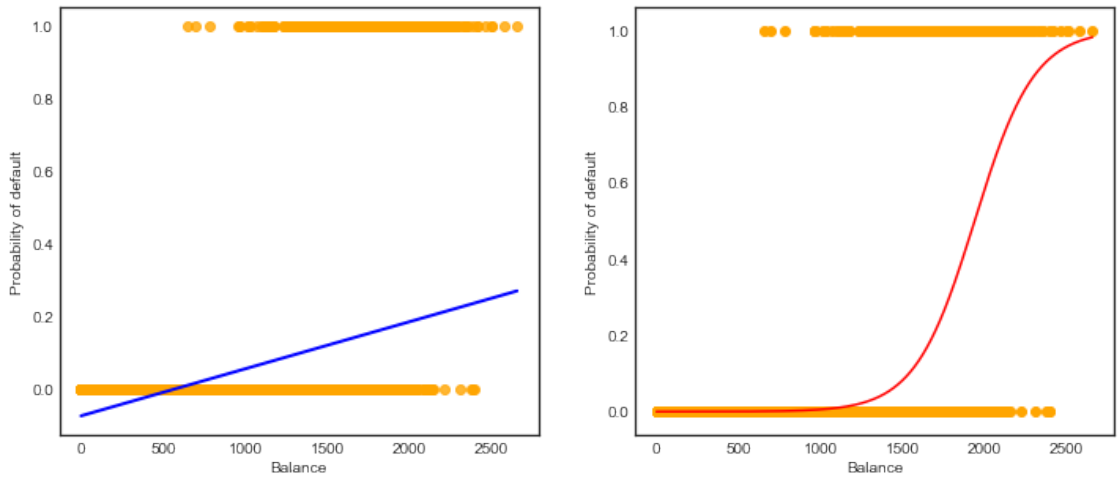


Table 4.1 using scikit-learn

```
In [58]: clf = skl_lm.LogisticRegression(solver='newton-cg')
         X_train = card.balance.values.reshape(-1,1)
         clf.fit(X_train,y)
         print(clf)
         print('classes: ',clf.classes_)
         print('coefficients: ',clf.coef_)
         print('intercept :', clf.intercept_)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='newton-cg',
          tol=0.0001, verbose=0, warm_start=False)
classes:  [0 1]
coefficients:  [[0.00549892]]
intercept : [-10.65132824]
```

Table 4.1 using statsmodels

```
In [62]: X_train = sm.add_constant(card.balance)
         est = smf.Logit(y, X_train).fit()
         est.summary2().tables[1]

Optimization terminated successfully.
         Current function value: 0.079823
         Iterations 10
```

Out[62]:

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
const	-10.651331	0.361169	-29.491287	3.723665e-191	-11.359208	-9.943453
balance	0.005499	0.000220	24.952404	2.010855e-137	0.005067	0.005931

Multiple Logistic Regression and Table 4.3

```
In [63]: X_train = sm.add_constant(card[['balance', 'income', 'student2']])
         est = smf.Logit(y, X_train).fit()
         est.summary2().tables[1]

Optimization terminated successfully.
         Current function value: 0.078577
         Iterations 10
```

Out[63]:

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
const	-10.869045	0.492273	-22.079320	4.995499e-108	-11.833882	-9.904209
balance	0.005737	0.000232	24.736506	4.331521e-135	0.005282	0.006191
income	0.000003	0.000008	0.369808	7.115254e-01	-0.000013	0.000019
student2	-0.646776	0.236257	-2.737595	6.189022e-03	-1.109831	-0.183721

Linear Discriminant Analysis and Table 4.4 (Confusion Matrix)

```
In [64]: X = card[['balance', 'income', 'student2']].as_matrix()
         y = card.default2.as_matrix()

         lda = LinearDiscriminantAnalysis(solver='svd')
         ldafit= lda.fit(X, y)
         y_pred=ldafit.predict(X)

         ldaTable = pd.DataFrame({'True default status': y, 'Predicted default status': y_pred
         ldaTable.replace(to_replace={0:'No', 1:'Yes'}, inplace=True)
         ldaTable.groupby(['Predicted default status','True default status']).size().unstack('

C:\Users\Preload\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: FutureWarning: Method .as
  """Entry point for launching an IPython kernel.
C:\Users\Preload\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning: Method .as
```

```
Out[64]: True default status         No  Yes
         Predicted default status
         No                        9645  254
         Yes                         22   79
```

```
In [90]: print(classification_report(y, y_pred, target_names=['No', 'Yes']))

         #F1 = 2 * (precision * recall) / (precision + recall)

                      precision    recall  f1-score   support

                  No       0.97      1.00      0.99      9667
                 Yes       0.78      0.24      0.36       333

           micro avg       0.97      0.97      0.97     10000
           macro avg       0.88      0.62      0.67     10000
        weighted avg       0.97      0.97      0.97     10000
```

More on Linear Discriminant Analysis (Section 4.6.3)

```
In [69]: # working with the Stock Market Data
         sdata = pd.read_csv('C:/DataSL/Smarket.csv', usecols=range(1,10), index_col=0, parse_
         sdata.head(3)
```

```
Out[69]:              Lag1   Lag2   Lag3   Lag4   Lag5  Volume  Today Direction
         Year
         2001-01-01  0.381 -0.192 -2.624 -1.055  5.010  1.1913  0.959        Up
         2001-01-01  0.959  0.381 -0.192 -2.624 -1.055  1.2965  1.032        Up
         2001-01-01  1.032  0.959  0.381 -0.192 -2.624  1.4112 -0.623      Down
```

```
In [70]: X_train = sdata[:'2004'][['Lag1','Lag2']]
         y_train = sdata[:'2004']['Direction']

         X_test = sdata['2005':][['Lag1','Lag2']]
         y_test = sdata['2005':]['Direction']

         lda = LinearDiscriminantAnalysis()
         pred = lda.fit(X_train, y_train).predict(X_test)
```

```
In [71]: lda.priors_
```

```
Out[71]: array([0.49198397, 0.50801603])
```

```
In [73]: lda.means_
         # group means
         #Col: Lag1, Lag2
```

```
          #Row:
          #Dwon
          #Up

Out[73]: array([[ 0.04279022,  0.03389409],
                [-0.03954635, -0.03132544]])

In [75]: # Coefficients of linear discriminants: Lag1 and Lag2
         lda.coef_

Out[75]: array([[-0.05544078, -0.0443452 ]])

In [77]: confusion_matrix(y_test, pred).T
         #True: Down Up
         #Pred: Down Up

Out[77]: array([[ 35,  35],
                [ 76, 106]], dtype=int64)

In [78]: print(classification_report(y_test, pred, digits=3))

              precision    recall  f1-score   support

        Down      0.500     0.315     0.387       111
          Up      0.582     0.752     0.656       141

   micro avg      0.560     0.560     0.560       252
   macro avg      0.541     0.534     0.522       252
weighted avg      0.546     0.560     0.538       252



In [80]: pred_p = lda.predict_proba(X_test)
         np.unique(pred_p[:,1]>0.5, return_counts=True)

Out[80]: (array([False,  True]), array([ 70, 182], dtype=int64))

In [81]: np.unique(pred_p[:,1]>0.9, return_counts=True)

Out[81]: (array([False]), array([252], dtype=int64))
```

0.0.1 4.6.4 Quadratic Discriminant Analysis

```
In [82]: qda = QuadraticDiscriminantAnalysis()
         pred = qda.fit(X_train, y_train).predict(X_test)

In [84]: qda.priors_

Out[84]: array([0.49198397, 0.50801603])
```

```
In [85]: qda.means_

Out[85]: array([[ 0.04279022,  0.03389409],
                [-0.03954635, -0.03132544]])

In [86]: confusion_matrix(y_test, pred).T

Out[86]: array([[ 30,  20],
                [ 81, 121]], dtype=int64)

In [87]: print(classification_report(y_test, pred, digits=3))

             precision    recall  f1-score   support

       Down      0.600     0.270     0.373       111
         Up      0.599     0.858     0.706       141

   micro avg      0.599     0.599     0.599       252
   macro avg      0.600     0.564     0.539       252
weighted avg      0.599     0.599     0.559       252
```

0.0.2 4.6.5 K-Nearest Neighbors

```
In [88]: knn = neighbors.KNeighborsClassifier(n_neighbors=1)
         pred = knn.fit(X_train, y_train).predict(X_test)
         print(confusion_matrix(y_test, pred).T)
         print(classification_report(y_test, pred, digits=3))

[[43 58]
 [68 83]]
             precision    recall  f1-score   support

       Down      0.426     0.387     0.406       111
         Up      0.550     0.589     0.568       141

   micro avg      0.500     0.500     0.500       252
   macro avg      0.488     0.488     0.487       252
weighted avg      0.495     0.500     0.497       252
```

```
In [89]: knn = neighbors.KNeighborsClassifier(n_neighbors=3)
         pred = knn.fit(X_train, y_train).predict(X_test)
         print(confusion_matrix(y_test, pred).T)
         print(classification_report(y_test, pred, digits=3))
```

```
[[48 55]
 [63 86]]
              precision    recall  f1-score   support

        Down      0.466     0.432     0.449       111
          Up      0.577     0.610     0.593       141

   micro avg      0.532     0.532     0.532       252
   macro avg      0.522     0.521     0.521       252
weighted avg      0.528     0.532     0.529       252
```