IBM Developer
SKILLS NETWORK

Classification with Python

In this notebook we try to practice all the classification algorithms that we have learned in this course.

We load a dataset using Pandas library, and apply the following algorithms, and find the best one for this specific dataset by accuracy evaluation methods.

Let's first load required libraries:

In [1]:
```
import itertools
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
import pandas as pd
import numpy as np
import matplotlib.ticker as ticker
from sklearn import preprocessing
%matplotlib inline
```

About dataset

This dataset is about past loans. The **Loan_train.csv** data set includes details of 346 customers whose loan are already paid off or defaulted. It includes following fields:

Field	Description
Loan_status	Whether a loan is paid off on in collection
Principal	Basic principal loan amount at the
Terms	Origination terms which can be weekly (7 days), biweekly, and monthly payoff schedule

Field	Description
Effective_date	When the loan got originated and took effects
Due_date	Since it's one-time payoff schedule, each loan has one single due date
Age	Age of applicant
Education	Education of applicant
Gender	The gender of applicant

Let's download the dataset

In [2]:

```
!wget -O loan_train.csv https://cf-courses-data.s3.us.cloud-object-storage.appdomain
```

```
--2021-10-06 13:46:23--  https://cf-courses-data.s3.us.cloud-object-storage.appdomai
n.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/FinalModule_Coursera/d
ata/loan_train.csv
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-dat
a.s3.us.cloud-object-storage.appdomain.cloud)... 198.23.119.245
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses
-data.s3.us.cloud-object-storage.appdomain.cloud)|198.23.119.245|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 23101 (23K) [text/csv]
Saving to: 'loan_train.csv'

loan_train.csv      100%[===================>]  22.56K  --.-KB/s    in 0.002s

2021-10-06 13:46:23 (14.0 MB/s) - 'loan_train.csv' saved [23101/23101]
```

Load Data From CSV File

In [3]:

```
df = pd.read_csv('loan_train.csv')
df.head()
```

Out[3]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education
0	0	0	PAIDOFF	1000	30	9/8/2016	10/7/2016	45	High School or Below
1	2	2	PAIDOFF	1000	30	9/8/2016	10/7/2016	33	Bechalor
2	3	3	PAIDOFF	1000	15	9/8/2016	9/22/2016	27	college
3	4	4	PAIDOFF	1000	30	9/9/2016	10/8/2016	28	college
4	6	6	PAIDOFF	1000	30	9/9/2016	10/8/2016	29	college

In [4]:

```
df.shape
```

Out[4]:

```
(346, 10)
```

Convert to date time object

In [5]:

```
df['due_date'] = pd.to_datetime(df['due_date'])
df['effective_date'] = pd.to_datetime(df['effective_date'])
df.head()
```

Out[5]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education	G
0	0	0	PAIDOFF	1000	30	2016-09-08	2016-10-07	45	High School or Below	
1	2	2	PAIDOFF	1000	30	2016-09-08	2016-10-07	33	Bechalor	
2	3	3	PAIDOFF	1000	15	2016-09-08	2016-09-22	27	college	
3	4	4	PAIDOFF	1000	30	2016-09-09	2016-10-08	28	college	
4	6	6	PAIDOFF	1000	30	2016-09-09	2016-10-08	29	college	

Data visualization and pre-processing

Let’s see how many of each class is in our data set

In [6]:

```
df['loan_status'].value_counts()
```

Out[6]:

```
PAIDOFF       260
COLLECTION     86
Name: loan_status, dtype: int64
```

260 people have paid off the loan on time while 86 have gone into collection

Let's plot some columns to underestand data better:

In [7]:

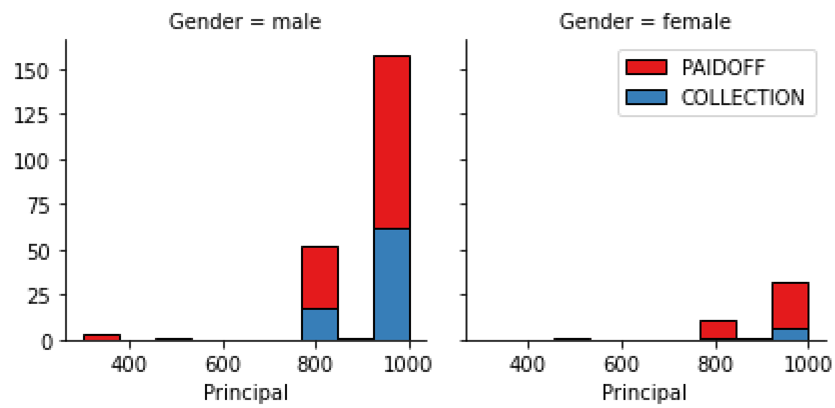
```
# notice: installing seaborn might takes a few minutes
!conda install -c anaconda seaborn -y
```

```
Collecting package metadata (current_repodata.json): done
Solving environment: done

# All requested packages already installed.
```

In [8]:

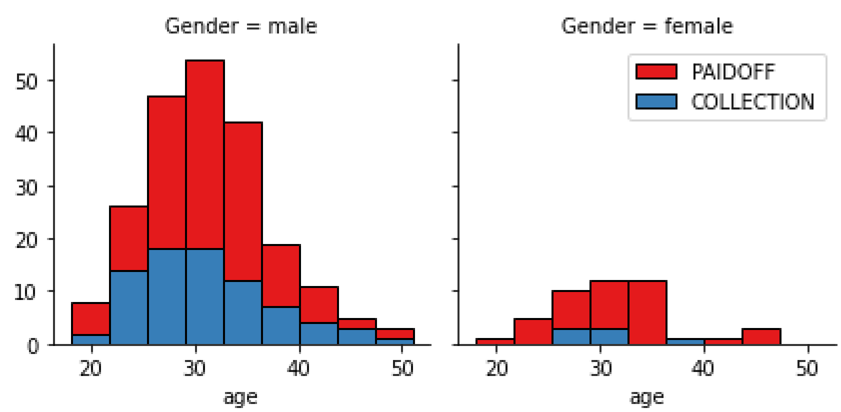
```
import seaborn as sns

bins = np.linspace(df.Principal.min(), df.Principal.max(), 10)
```

```
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wrap=2)
g.map(plt.hist, 'Principal', bins=bins, ec="k")

g.axes[-1].legend()
plt.show()
```



In [9]:

```
bins = np.linspace(df.age.min(), df.age.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wrap=2)
g.map(plt.hist, 'age', bins=bins, ec="k")

g.axes[-1].legend()
plt.show()
```

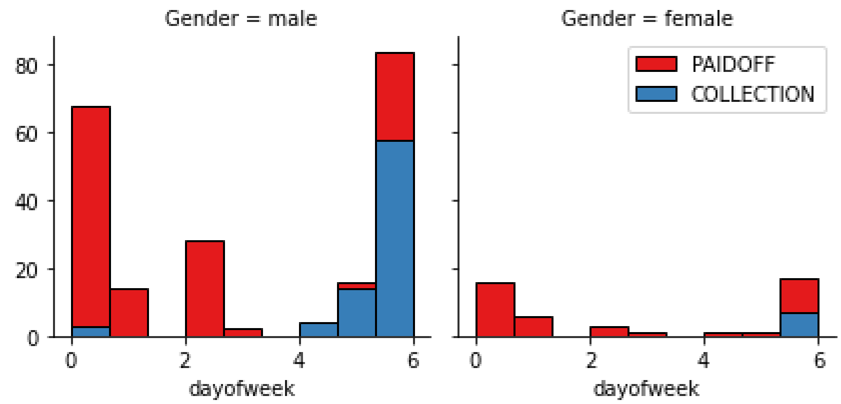


Pre-processing: Feature selection/extraction

Let's look at the day of the week people get the loan

In [10]:

```
df['dayofweek'] = df['effective_date'].dt.dayofweek
bins = np.linspace(df.dayofweek.min(), df.dayofweek.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wrap=2)
g.map(plt.hist, 'dayofweek', bins=bins, ec="k")
g.axes[-1].legend()
plt.show()
```



We see that people who get the loan at the end of the week don't pay it off, so let's use Feature binarization to set a threshold value less than day 4

In [11]:

```
df['weekend'] = df['dayofweek'].apply(lambda x: 1 if (x>3)  else 0)
df.head()
```

Out[11]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education	G
0	0	0	PAIDOFF	1000	30	2016-09-08	2016-10-07	45	High School or Below	
1	2	2	PAIDOFF	1000	30	2016-09-08	2016-10-07	33	Bechalor	
2	3	3	PAIDOFF	1000	15	2016-09-08	2016-09-22	27	college	
3	4	4	PAIDOFF	1000	30	2016-09-09	2016-10-08	28	college	
4	6	6	PAIDOFF	1000	30	2016-09-09	2016-10-08	29	college	

Convert Categorical features to numerical values

Let's look at gender:

In [12]:

```
df.groupby(['Gender'])['loan_status'].value_counts(normalize=True)
```

Out[12]:

```
Gender  loan_status
female  PAIDOFF        0.865385
        COLLECTION     0.134615
male    PAIDOFF        0.731293
        COLLECTION     0.268707
Name: loan_status, dtype: float64
```

86 % of female pay there loans while only 73 % of males pay there loan

Let's convert male to 0 and female to 1:

In [13]:
```
df['Gender'].replace(to_replace=['male','female'], value=[0,1],inplace=True)
df.head()
```

Out[13]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education	G
0	0	0	PAIDOFF	1000	30	2016-09-08	2016-10-07	45	High School or Below	
1	2	2	PAIDOFF	1000	30	2016-09-08	2016-10-07	33	Bechalor	
2	3	3	PAIDOFF	1000	15	2016-09-08	2016-09-22	27	college	
3	4	4	PAIDOFF	1000	30	2016-09-09	2016-10-08	28	college	
4	6	6	PAIDOFF	1000	30	2016-09-09	2016-10-08	29	college	

One Hot Encoding

How about education?

In [14]:
```
df.groupby(['education'])['loan_status'].value_counts(normalize=True)
```

Out[14]:
```
education             loan_status
Bechalor              PAIDOFF        0.750000
                      COLLECTION     0.250000
High School or Below  PAIDOFF        0.741722
                      COLLECTION     0.258278
Master or Above       COLLECTION     0.500000
                      PAIDOFF        0.500000
college               PAIDOFF        0.765101
                      COLLECTION     0.234899
Name: loan_status, dtype: float64
```

Features before One Hot Encoding

In [15]:
```
df[['Principal','terms','age','Gender','education']].head()
```

Out[15]:

	Principal	terms	age	Gender	education
0	1000	30	45	0	High School or Below
1	1000	30	33	1	Bechalor
2	1000	15	27	0	college
3	1000	30	28	1	college

	Principal	terms	age	Gender	education
4	1000	30	29	0	college

Use one hot encoding technique to conver categorical varables to binary variables and append them to the feature Data Frame

In [16]:

```
Feature = df[['Principal','terms','age','Gender','weekend']]
Feature = pd.concat([Feature,pd.get_dummies(df['education'])], axis=1)
Feature.drop(['Master or Above'], axis = 1,inplace=True)
Feature.head()
```

Out[16]:

	Principal	terms	age	Gender	weekend	Bechalor	High School or Below	college
0	1000	30	45	0	0	0	1	0
1	1000	30	33	1	0	1	0	0
2	1000	15	27	0	0	0	0	1
3	1000	30	28	1	1	0	0	1
4	1000	30	29	0	1	0	0	1

Feature Selection

Let's define feature sets, X:

In [17]:

```
X = Feature
X[0:5]
```

Out[17]:

	Principal	terms	age	Gender	weekend	Bechalor	High School or Below	college
0	1000	30	45	0	0	0	1	0
1	1000	30	33	1	0	1	0	0
2	1000	15	27	0	0	0	0	1
3	1000	30	28	1	1	0	0	1
4	1000	30	29	0	1	0	0	1

What are our lables?

In [18]:

```
y = df['loan_status'].values
y[0:5]
```

Out[18]:

```
array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF'],
      dtype=object)
```

Normalize Data

Data Standardization give data zero mean and unit variance (technically should be done after train test split)

In [19]:

```
X= preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

Out[19]:

```
array([[ 0.51578458,  0.92071769,  2.33152555, -0.42056004, -1.20577805,
        -0.38170062,  1.13639374, -0.86968108],
       [ 0.51578458,  0.92071769,  0.34170148,  2.37778177, -1.20577805,
         2.61985426, -0.87997669, -0.86968108],
       [ 0.51578458, -0.95911111, -0.65321055, -0.42056004, -1.20577805,
        -0.38170062, -0.87997669,  1.14984679],
       [ 0.51578458,  0.92071769, -0.48739188,  2.37778177,  0.82934003,
        -0.38170062, -0.87997669,  1.14984679],
       [ 0.51578458,  0.92071769, -0.3215732 , -0.42056004,  0.82934003,
        -0.38170062, -0.87997669,  1.14984679]])
```

Classification

Now, it is your turn, use the training set to build an accurate model. Then use the test set to report the accuracy of the model You should use the following algorithm:

- K Nearest Neighbor(KNN)
- Decision Tree
- Support Vector Machine
- Logistic Regression

__ Notice:__

- You can go above and change the pre-processing, feature selection, feature-extraction, and so on, to make a better model.
- You should use either scikit-learn, Scipy or Numpy libraries for developing the classification algorithms.
- You should include the code of the algorithm in the following cells.

K Nearest Neighbor(KNN)

Notice: You should find the best k to build the model with the best accuracy.\ **warning:** You should not use the **loan_test.csv** for finding the best k, however, you can split your train_loan.csv into train and test to find the best **k**.

In [20]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_sta
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (276, 8) (276,)
Test set: (70, 8) (70,)
```

In [101…

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
k = 3
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
yhat = neigh.predict(X_test)
yhat[0:5]
```

Out[101…

```
array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF'],
      dtype=object)
```

In [102…

```
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_train)
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
Train set Accuracy:  0.8333333333333334
Test set Accuracy:  0.7142857142857143
```

In [90]:

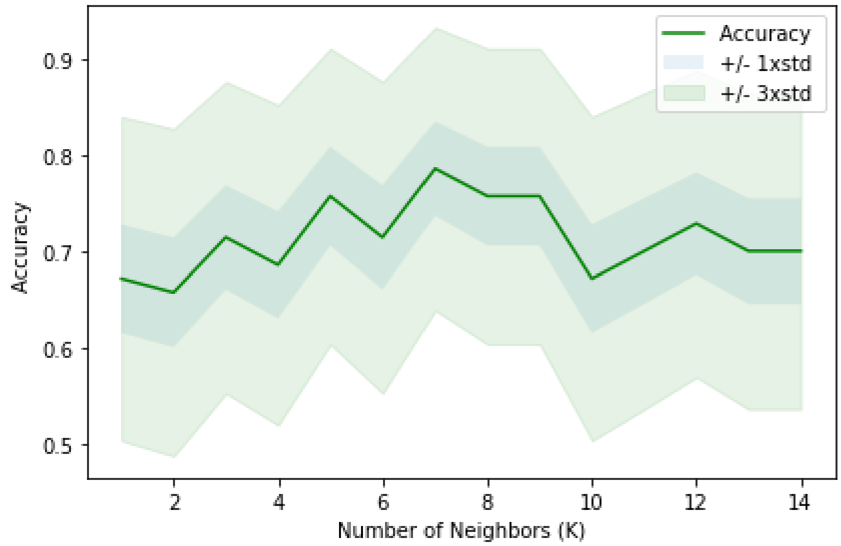
```
Ks = 15
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))

for n in range(1,Ks):

    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)


    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

mean_acc
```

Out[90]:

```
array([0.67142857, 0.65714286, 0.71428571, 0.68571429, 0.75714286,
       0.71428571, 0.78571429, 0.75714286, 0.75714286, 0.67142857,
       0.7       , 0.72857143, 0.7       , 0.7       ])
```

In [103…

```
plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.
plt.fill_between(range(1,Ks),mean_acc - 3 * std_acc,mean_acc + 3 * std_acc, alpha=0.
plt.legend(('Accuracy ', '+/- 1xstd','+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
```



In [104…

```
print( "The best accuracy was with", mean_acc.max(), "with k=", mean_acc.argmax()+1)
```

```
The best accuracy was with 0.7857142857142857 with k= 7
```

In [106…

```
k = 7
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
```

Decision Tree

In [26]:

```
from sklearn.tree import DecisionTreeClassifier
X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y, test_size=0.3,
print('Shape of X training set {}'.format(X_trainset.shape),'&',' Size of Y training
print('Shape of X test set {}'.format(X_testset.shape),'&',' Size of Y test set {}'.
```

```
Shape of X training set (242, 8) &  Size of Y training set (242,)
Shape of X test set (104, 8) &  Size of Y test set (104,)
```

In [27]:

```
loanTree = DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

In [28]:

```
loanTree.fit(X_trainset, y_trainset)
```

Out[28]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

In [39]:

```
predTree = loanTree.predict(X_testset)
predTree
```

Out[39]:

```
array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'COLLECTION', 'COLLECTION', 'PAIDOFF', 'COLLECTION', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'COLLECTION', 'COLLECTION',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'COLLECTION',
       'COLLECTION', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'COLLECTION',
       'PAIDOFF', 'COLLECTION', 'PAIDOFF', 'COLLECTION', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'COLLECTION', 'COLLECTION', 'COLLECTION',
```

```
       'COLLECTION', 'COLLECTION', 'COLLECTION', 'PAIDOFF', 'COLLECTION',
       'COLLECTION', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'COLLECTION', 'PAIDOFF', 'PAIDOFF', 'COLLECTION',
       'PAIDOFF', 'COLLECTION', 'COLLECTION', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'COLLECTION',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'COLLECTION', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'COLLECTION', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'COLLECTION', 'PAIDOFF', 'COLLECTION',
       'COLLECTION', 'COLLECTION', 'COLLECTION', 'PAIDOFF', 'PAIDOFF',
       'COLLECTION', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'COLLECTION', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'COLLECTION', 'COLLECTION', 'PAIDOFF',
       'PAIDOFF', 'COLLECTION'], dtype=object)
```

In [30]:

```
from sklearn import metrics
import matplotlib.pyplot as plt
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_testset, predTree))
```

```
DecisionTrees's Accuracy:  0.6538461538461539
```

Support Vector Machine

In [108…

```
from sklearn import svm
clf = svm.SVC(kernel="sigmoid")
clf.fit(X_train, y_train)
```

Out[108…

```
SVC(kernel='sigmoid')
```

In [93]:

```
yhat = clf.predict(X_test)
yhat
```

Out[93]:

```
array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'COLLECTION', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'COLLECTION',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'COLLECTION',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'COLLECTION', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'COLLECTION', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF'], dtype=object)
```

In [94]:

```
from sklearn.metrics import f1_score
f1_score(y_test,yhat, average="weighted")
```

Out[94]:

```
0.6892857142857144
```

Logistic Regression

In [109…

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
LR
```

Out[109…

```
LogisticRegression(C=0.01, solver='liblinear')
```

In [52]:

```
yhat = LR.predict(X_test)
yhat
```

Out[52]:

```
array(['COLLECTION', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'COLLECTION', 'PAIDOFF', 'COLLECTION', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'COLLECTION', 'PAIDOFF', 'PAIDOFF', 'COLLECTION',
       'COLLECTION', 'PAIDOFF', 'COLLECTION', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'COLLECTION', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'COLLECTION',
       'PAIDOFF', 'PAIDOFF', 'COLLECTION', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'COLLECTION', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF'], dtype=object)
```

In [53]:

```
yhat_prob = LR.predict_proba(X_test)
yhat_prob
```

Out[53]:

```
array([[0.5034238 , 0.4965762 ],
       [0.45206111, 0.54793889],
       [0.30814132, 0.69185868],
       [0.34259428, 0.65740572],
       [0.32025894, 0.67974106],
       [0.31680537, 0.68319463],
       [0.48830185, 0.51169815],
       [0.47823073, 0.52176927],
       [0.34259428, 0.65740572],
       [0.4934056 , 0.5065944 ],
       [0.33806706, 0.66193294],
       [0.49662231, 0.50337769],
       [0.24891907, 0.75108093],
       [0.3419095 , 0.6580905 ],
       [0.43751789, 0.56248211],
       [0.25760497, 0.74239503],
       [0.52357188, 0.47642812],
       [0.30450278, 0.69549722],
       [0.50166363, 0.49833637],
       [0.3195971 , 0.6804029 ],
       [0.44276988, 0.55723012],
       [0.49410185, 0.50589815],
       [0.51350333, 0.48649667],
       [0.47203498, 0.52796502],
       [0.40944694, 0.59055306],
       [0.50846442, 0.49153558],
       [0.51098415, 0.48901585],
       [0.37457647, 0.62542353],
```

```
       [0.50418423, 0.49581577],
       [0.25299635, 0.74700365],
       [0.46824113, 0.53175887],
       [0.46024688, 0.53975312],
       [0.46206917, 0.53793083],
       [0.48402425, 0.51597575],
       [0.38818191, 0.61181809],
       [0.45821326, 0.54178674],
       [0.50166363, 0.49833637],
       [0.28973585, 0.71026415],
       [0.4569882 , 0.5430118 ],
       [0.45494718, 0.54505282],
       [0.50670462, 0.49329538],
       [0.32179362, 0.67820638],
       [0.45245776, 0.54754224],
       [0.50846442, 0.49153558],
       [0.30664231, 0.69335769],
       [0.49515584, 0.50484416],
       [0.47075244, 0.52924756],
       [0.49662231, 0.50337769],
       [0.45571125, 0.54428875],
       [0.45567623, 0.54432377],
       [0.27794059, 0.72205941],
       [0.46744865, 0.53255135],
       [0.30501081, 0.69498919],
       [0.48906194, 0.51093806],
       [0.28058426, 0.71941574],
       [0.24921106, 0.75078894],
       [0.31522806, 0.68477194],
       [0.43036995, 0.56963005],
       [0.46824113, 0.53175887],
       [0.33513632, 0.66486368],
       [0.41925226, 0.58074774],
       [0.33133167, 0.66866833],
       [0.45821326, 0.54178674],
       [0.52608635, 0.47391365],
       [0.32399805, 0.67600195],
       [0.49410185, 0.50589815],
       [0.33133167, 0.66866833],
       [0.41737926, 0.58262074],
       [0.44996108, 0.55003892],
       [0.32399805, 0.67600195]])
```

In [63]:

```
from sklearn.metrics import jaccard_score
jaccard_score(y_test, yhat, pos_label="PAIDOFF")
```

Out[63]:

```
0.6764705882352942
```

Model Evaluation using Test set

In [64]:

```
from sklearn.metrics import jaccard_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
```

First, download and load the test set:

In [65]:

```
!wget -O loan_test.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-
```

```
--2021-10-06 18:38:42--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-course
s-data/CognitiveClass/ML0101ENv3/labs/loan_test.csv
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.sof
tlayer.net)... 67.228.254.196
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorag
e.softlayer.net)|67.228.254.196|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3642 (3.6K) [text/csv]
Saving to: ‘loan_test.csv’

loan_test.csv       100%[===================>]   3.56K  --.-KB/s    in 0s

2021-10-06 18:38:42 (41.2 MB/s) - ‘loan_test.csv’ saved [3642/3642]
```

Load Test set for evaluation

In [66]:

```
test_df = pd.read_csv('loan_test.csv')
test_df.head()
```

Out[66]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education
0	1	1	PAIDOFF	1000	30	9/8/2016	10/7/2016	50	Bechalor
1	5	5	PAIDOFF	300	7	9/9/2016	9/15/2016	35	Master or Above
2	21	21	PAIDOFF	1000	30	9/10/2016	10/9/2016	43	High School or Below
3	24	24	PAIDOFF	1000	30	9/10/2016	10/9/2016	26	college
4	35	35	PAIDOFF	800	15	9/11/2016	9/25/2016	29	Bechalor

In [72]:

```
test_df['due_date'] = pd.to_datetime(test_df['due_date'])
test_df['effective_date'] = pd.to_datetime(test_df['effective_date'])
test_df['dayofweek'] = test_df['effective_date'].dt.dayofweek
```

In [77]:

```
test_df['weekend'] = test_df['dayofweek'].apply(lambda x: 1 if (x>3)  else 0)
test_df['Gender'].replace(to_replace=['male','female'], value=[0,1],inplace=True)
```

In [78]:

```
test_feature = test_df[['Principal','terms','age','Gender','weekend']]
test_feature = pd.concat([test_feature,pd.get_dummies(test_df['education'])], axis=1
test_feature.drop(['Master or Above'], axis = 1,inplace=True)
test_feature.head()
```

Out[78]:

	Principal	terms	age	Gender	weekend	Bechalor	High School or Below	college

	Principal	terms	age	Gender	weekend	Bechalor	High School or Below	college
0	1000	30	50	1	0	1	0	0
1	300	7	35	0	1	0	0	0
2	1000	30	43	1	1	0	1	0
3	1000	30	26	0	1	0	0	1
4	800	15	29	0	1	1	0	0

In [79]:

```
test_X = preprocessing.StandardScaler().fit(test_feature).transform(test_feature)
test_X[0:5]
```

Out[79]:

```
array([[ 0.49362588,  0.92844966,  3.05981865,  1.97714211, -1.30384048,
         2.39791576, -0.79772404, -0.86135677],
       [-3.56269116, -1.70427745,  0.53336288, -0.50578054,  0.76696499,
        -0.41702883, -0.79772404, -0.86135677],
       [ 0.49362588,  0.92844966,  1.88080596,  1.97714211,  0.76696499,
        -0.41702883,  1.25356634, -0.86135677],
       [ 0.49362588,  0.92844966, -0.98251057, -0.50578054,  0.76696499,
        -0.41702883, -0.79772404,  1.16095912],
       [-0.66532184, -0.78854628, -0.47721942, -0.50578054,  0.76696499,
         2.39791576, -0.79772404, -0.86135677]])
```

In [80]:

```
test_y = test_df['loan_status'].values
test_y[0:5]
```

Out[80]:

```
array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF'],
      dtype=object)
```

In [110…

```
knn_pred = neigh.predict(test_X)
print("KNN Jaccard index: %.2f" % jaccard_score(test_y, knn_pred, pos_label="PAIDOFF
print("KNN F1-score: %.2f" % f1_score(test_y, knn_pred, average='weighted') )
```

```
KNN Jaccard index: 0.65
KNN F1-score: 0.63
```

In [111…

```
tree_pred = loanTree.predict(test_X)
print("DT Jaccard index: %.2f" % jaccard_score(test_y, tree_pred, pos_label="PAIDOFF
print("DT F1-score: %.2f" % f1_score(test_y, tree_pred, average='weighted') )
```

```
DT Jaccard index: 0.73
DT F1-score: 0.78
```

In [112…

```
svm_pred = clf.predict(test_X)
print("SVM Jaccard index: %.2f" % jaccard_score(test_y, svm_pred, pos_label="PAIDOFF
print("SVM F1-score: %.2f" % f1_score(test_y, svm_pred, average='weighted') )
```

```
SVM Jaccard index: 0.70
SVM F1-score: 0.64
```

In [113…

```
log_pred = LR.predict(test_X)
log_pred_prob = LR.predict_proba(test_X)
print("LR Jaccard index: %.2f" % jaccard_score(test_y, log_pred, pos_label="PAIDOFF"
```

```
print("LR F1-score: %.2f" % f1_score(test_y, log_pred, average='weighted') )
print("LR LogLoss: %.2f" % log_loss(test_y, log_pred_prob))
```

```
LR Jaccard index: 0.74
LR F1-score: 0.66
LR LogLoss: 0.57
```

Report

You should be able to report the accuracy of the built model using different evaluation metrics:

Algorithm	Jaccard	F1-score	LogLoss
KNN	0.65	0.63	NA
Decision Tree	0.73	0.78	NA
SVM	0.70	0.64	NA
LogisticRegression	0.74	0.66	0.57

Want to learn more?

IBM SPSS Modeler is a comprehensive analytics platform that has many machine learning algorithms. It has been designed to bring predictive intelligence to decisions made by individuals, by groups, by systems – by your enterprise as a whole. A free trial is available through this course, available here: SPSS Modeler

Also, you can use Watson Studio to run these notebooks faster with bigger datasets. Watson Studio is IBM's leading cloud solution for data scientists, built by data scientists. With Jupyter notebooks, RStudio, Apache Spark and popular libraries pre-packaged in the cloud, Watson Studio enables data scientists to collaborate on their projects without having to install anything. Join the fast-growing community of Watson Studio users today with a free account at Watson Studio

Thanks for completing this lesson!

Author: Saeed Aghabozorgi

Saeed Aghabozorgi, PhD is a Data Scientist in IBM with a track record of developing enterprise level applications that substantially increases clients' ability to turn data into actionable knowledge. He is a researcher in data mining field and expert in developing advanced analytic methods like machine learning and statistical modelling on large datasets.

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-10-27	2.1	Lakshmi Holla	Made changes in import statement due to updates in version of sklearn library
2020-08-27	2.0	Malika Singla	Added lab to GitLab