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 [315]:
```
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
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 [168]:
```
!wget -O loan_train.csv https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/FinalModule_Coursera/data/loan_train.csv
```

```
--2022-03-07 17:50:11--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/FinalModule_Coursera/data/loan_train.csv
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud)... 169.63.118.104
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud)|169.63.118.104|: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 0.07s

2022-03-07 17:50:11 (304 KB/s) - 'loan_train.csv' saved [23101/23101]
```

Load Data From CSV File

In [169]:
```
df = pd.read_csv('loan_train.csv')
df.head()
```

Out[169]:

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

In [170]:
```
df.shape
```

Out[170]: (346, 10)

Convert to date time object

In [171]:
```
df['due_date'] = pd.to_datetime(df['due_date'])
df['effective_date'] = pd.to_datetime(df['effective_date'])
df.head()
```

Out[171]:

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

Data visualization and pre-processing

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

In [172]:
```
df['loan_status'].value_counts()
```

Out[172]:
```
PAIDOFF       260
COLLECTION     86
Name: loan_status, dtype: int64
```

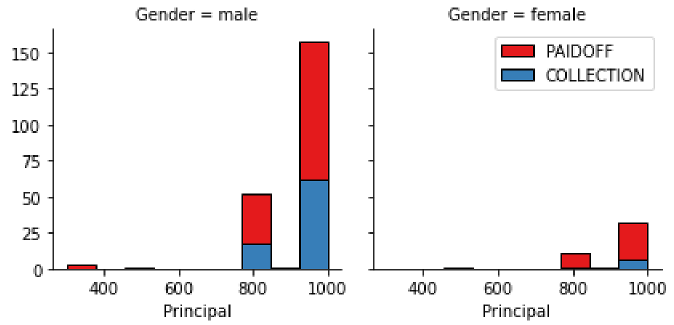
260 people have paid off the loan on time while 86 have gone into collection

Let's plot some columns to underestand data better:

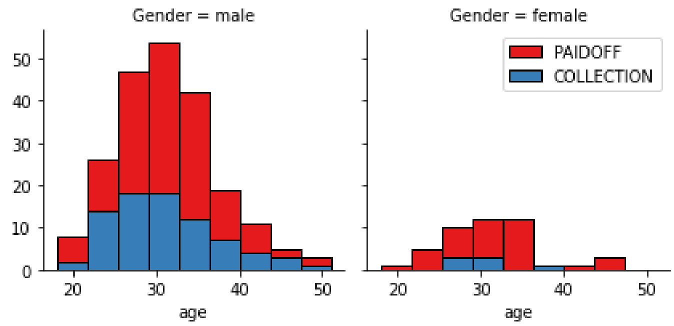
In [173]:
```
# notice: installing seaborn might takes a few minutes
!conda install -c anaconda seaborn -y
```

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

# All requested packages already installed.
```

In [174]:
```
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()
```



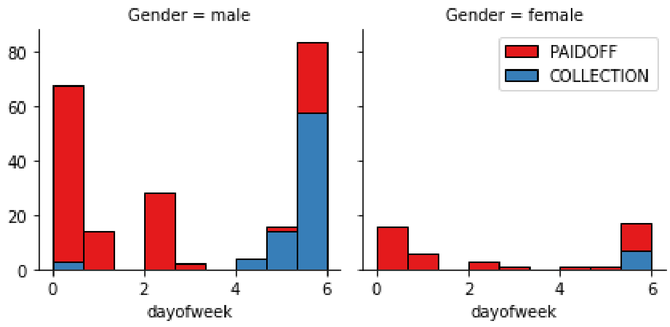
In [175]:
```
bins = np.linspace(df.age.min(), df.age.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wrap=2)
g.map(plt.hist, 'age', bins=bins, ec="k")

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 [176]:
```
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 [177]:
```
df['weekend'] = df['dayofweek'].apply(lambda x: 1 if (x>3)  else 0)
df.head()
```

Out[177]:

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

Convert Categorical features to numerical values

Let's look at gender:

In [178]:
```
df.groupby(['Gender'])['loan_status'].value_counts(normalize=True)
```

Out[178]:
```
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 [179]:
```
df['Gender'].replace(to_replace=['male','female'], value=[0,1],inplace=True)
df.head()
```

Out[179]:

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

One Hot Encoding

How about education?

In [180]:
```
df.groupby(['education'])['loan_status'].value_counts(normalize=True)
```

Out[180]:
```
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 [181]:
```
df[['Principal','terms','age','Gender','education']].head()
```

Out[181]:

	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
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 [182]:
```
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[182]:

	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 [204]:
```
X = Feature
X[0:5]
```

Out[204]:

	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 [205]:
```
y = df['loan_status'].values
y[0:5]
```

Out[205]:
```
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 [206]:
```
X= preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

Out[206]:
```
array([[ 0.52,  0.92,  2.33, -0.42, -1.21, -0.38,  1.14, -0.87],
       [ 0.52,  0.92,  0.34,  2.38, -1.21,  2.62, -0.88, -0.87],
       [ 0.52, -0.96, -0.65, -0.42, -1.21, -0.38, -0.88,  1.15],
       [ 0.52,  0.92, -0.49,  2.38,  0.83, -0.38, -0.88,  1.15],
       [ 0.52,  0.92, -0.32, -0.42,  0.83, -0.38, -0.88,  1.15]])
```

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.

In [289]:
```
scores = {}
```

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 [283]:
```
# Train Test Split
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_state=4)
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 [287]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics

k = 4

# Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train, y_train)

# Prediction
yhat = neigh.predict(X_test)
yhat[0:5]

# Accuracy evaluation
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:  0.8152173913043478
Test set Accuracy:  0.7
```

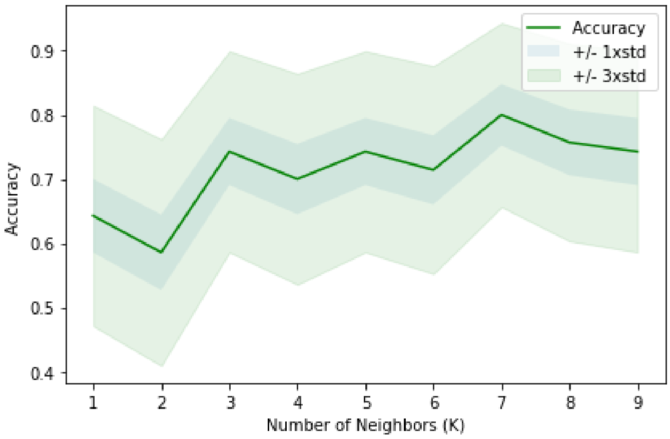
In [290]:
```
from sklearn.metrics import f1_score
from sklearn.metrics import jaccard_score

# Evaluation
scores['KNN_f1-score'] = f1_score(y_test, yhat, average='weighted')
scores['KNN_jaccard'] = jaccard_score(y_test, yhat,pos_label='PAIDOFF')
print("KNN's f1 score: ", scores['KNN_f1-score'])
print("KNN's jaccard score: ", scores['KNN_jaccard'])
print("KNN's Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
KNN's f1 score:  0.7212490479817212
KNN's jaccard score:  0.6557377049180327
KNN's Accuracy:  0.7
```

In [291]:
```python
# Calculate the accuracy of KNN for different values of k.
Ks = 10
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[291]:
```
array([0.64, 0.59, 0.74, 0.7 , 0.74, 0.71, 0.8 , 0.76, 0.74])
```

In [292]:
```python
# Plot the model accuracy for a different number of neighbors.
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=0.10)
plt.fill_between(range(1,Ks),mean_acc - 3 * std_acc,mean_acc + 3 * std_acc, alpha=0.10,color="green")
plt.legend(('Accuracy ', '+/- 1xstd','+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
```



In [293]:
```python
print( "The best accuracy was with", mean_acc.max(), "with k=", mean_acc.argmax()+1)
```

```
The best accuracy was with 0.8 with k= 7
```

Decision Tree

In [294]:
```python
# Train Test Split
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_state=3)
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 [295]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics

# Train Model and Predict
Dec_Tree = DecisionTreeClassifier(criterion="entropy", max_depth = 2)
Dec_Tree.fit(X_train,y_train)

# Prediction
predTree = Dec_Tree.predict(X_test)
print (predTree [0:5])
print (y_test [0:5])
```

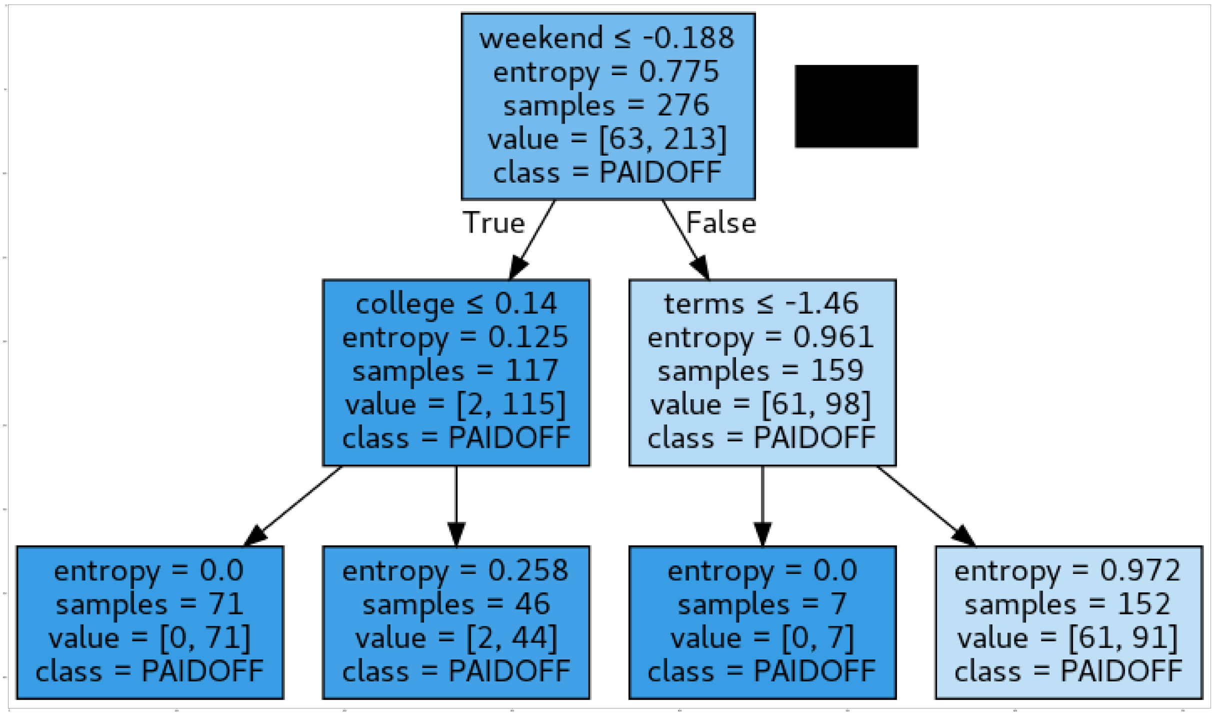
```
['PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF']
['PAIDOFF' 'PAIDOFF' 'COLLECTION' 'COLLECTION' 'PAIDOFF']
```

In [296]:
```
from sklearn.metrics import f1_score
from sklearn.metrics import jaccard_score

# Evaluation
scores['Decision_Tree_f1-score'] = f1_score(y_test, yhat, average='weighted')
scores['Decision_Tree_jaccard'] = jaccard_score(y_test, yhat,pos_label='PAIDOFF')
print("Decision Tree's f1 score: ", scores['Decision_Tree_f1-score'])
print("Decision Tree's jaccard score: ", scores['Decision_Tree_jaccard'])
print("Decision Tree's Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
Decision Tree's f1 score:  0.6043650793650793
Decision Tree's jaccard score:  0.6363636363636364
Decision Tree's Accuracy:  0.6571428571428571
```

In [297]:
```
from  io import StringIO
import pydotplus
import matplotlib.image as mpimg
from sklearn import tree
%matplotlib inline

# Visualization
dot_data = StringIO()
filename = "Dec_tree.png"
featureNames = Feature.columns
out=tree.export_graphviz(Dec_Tree,feature_names=featureNames, out_file=dot_data, class_names= np.unique(y_trai
n), filled=True,  special_characters=True,rotate=False)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png(filename)
img = mpimg.imread(filename)
plt.figure(figsize=(100, 200))
plt.imshow(img,interpolation='nearest')
```

Out[297]: <matplotlib.image.AxesImage at 0x7f6b15c88e50>



Support Vector Machine

In [298]:
```
# Train Test Split
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_state=4)
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 [299]:
```python
from sklearn import svm

# Modeling SVM
# clf = svm.SVC(kernel='rbf')
clf = svm.SVC()
svm_model = clf.fit(X_train, y_train)
svm_model

# Prediction
yhat = svm_model.predict(X_test)
yhat
```

Out[299]:
```
array(['COLLECTION', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'COLLECTION', 'COLLECTION', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'COLLECTION', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'COLLECTION', 'COLLECTION', 'PAIDOFF', 'COLLECTION',
       'COLLECTION', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'COLLECTION', 'PAIDOFF', 'PAIDOFF', '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 [301]:
```python
from sklearn.metrics import f1_score
from sklearn.metrics import jaccard_score

# Evaluation
scores['SVM_f1-score'] = f1_score(y_test, yhat, average='weighted')
scores['SVM_jaccard'] = jaccard_score(y_test, yhat,pos_label='PAIDOFF')
print("SVM's f1 score: ", scores['SVM_f1-score'])
print("SVM's jaccard score: ", scores['SVM_jaccard'])
print("SVM's Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

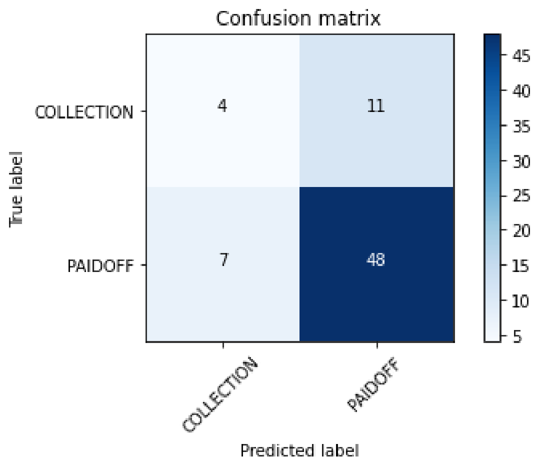
```
SVM's f1 score:  0.7275882012724117
SVM's jaccard score:  0.7272727272727273
SVM's Accuracy:  0.7428571428571429
```

In [302]:

```
from sklearn.metrics import classification_report, confusion_matrix
import itertools

# Evaluation
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [303]:
```
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, yhat, labels=None)
np.set_printoptions(precision=2)

print (classification_report(y_test, yhat))

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['COLLECTION','PAIDOFF'],normalize= False,  title='Confusion matri
x')
```

```
              precision    recall  f1-score   support

  COLLECTION       0.36      0.27      0.31        15
     PAIDOFF       0.81      0.87      0.84        55

    accuracy                           0.74        70
   macro avg       0.59      0.57      0.57        70
weighted avg       0.72      0.74      0.73        70

Confusion matrix, without normalization
[[ 4 11]
 [ 7 48]]
```



Logistic Regression

In [373]:
```
# Train Test Split
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_state=9)
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 [374]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

# Modeling
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
yhat = LR.predict(X_test)
yhat
```

Out[374]:
```
array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', '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',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'COLLECTION', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'COLLECTION', 'PAIDOFF',
       'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF',
       'PAIDOFF'], dtype=object)
```

In [375]:

```
# predict proba
yhat_prob = LR.predict_proba(X_test)
yhat_prob
```

```
Out[375]: array([[0.48, 0.52],
                 [0.33, 0.67],
                 [0.46, 0.54],
                 [0.49, 0.51],
                 [0.47, 0.53],
                 [0.5 , 0.5 ],
                 [0.46, 0.54],
                 [0.43, 0.57],
                 [0.5 , 0.5 ],
                 [0.49, 0.51],
                 [0.33, 0.67],
                 [0.31, 0.69],
                 [0.33, 0.67],
                 [0.32, 0.68],
                 [0.31, 0.69],
                 [0.48, 0.52],
                 [0.34, 0.66],
                 [0.31, 0.69],
                 [0.41, 0.59],
                 [0.47, 0.53],
                 [0.47, 0.53],
                 [0.48, 0.52],
                 [0.32, 0.68],
                 [0.31, 0.69],
                 [0.48, 0.52],
                 [0.49, 0.51],
                 [0.39, 0.61],
                 [0.34, 0.66],
                 [0.31, 0.69],
                 [0.33, 0.67],
                 [0.33, 0.67],
                 [0.31, 0.69],
                 [0.5 , 0.5 ],
                 [0.47, 0.53],
                 [0.31, 0.69],
                 [0.34, 0.66],
                 [0.47, 0.53],
                 [0.27, 0.73],
                 [0.47, 0.53],
                 [0.43, 0.57],
                 [0.31, 0.69],
                 [0.48, 0.52],
                 [0.48, 0.52],
                 [0.31, 0.69],
                 [0.3 , 0.7 ],
                 [0.46, 0.54],
                 [0.26, 0.74],
                 [0.31, 0.69],
                 [0.46, 0.54],
                 [0.46, 0.54],
                 [0.5 , 0.5 ],
                 [0.31, 0.69],
                 [0.47, 0.53],
                 [0.48, 0.52],
                 [0.48, 0.52],
                 [0.3 , 0.7 ],
                 [0.29, 0.71],
                 [0.33, 0.67],
                 [0.45, 0.55],
                 [0.49, 0.51],
                 [0.48, 0.52],
                 [0.51, 0.49],
                 [0.48, 0.52],
                 [0.46, 0.54],
                 [0.44, 0.56],
                 [0.5 , 0.5 ],
                 [0.47, 0.53],
                 [0.26, 0.74],
                 [0.45, 0.55],
                 [0.49, 0.51]])
```

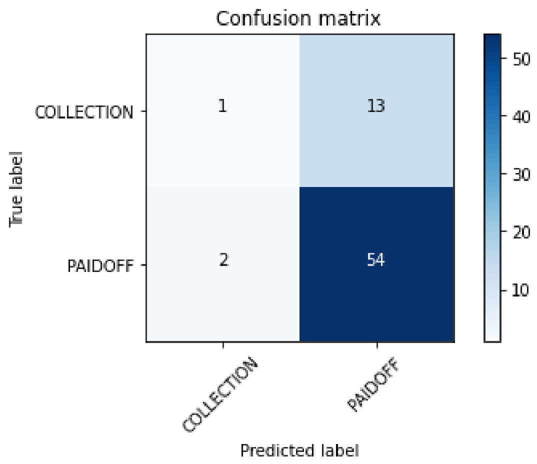
In [376]:
```
from sklearn.metrics import f1_score
from sklearn.metrics import jaccard_score

# Evaluation
scores['lr_f1-score'] = f1_score(y_test, yhat, average='weighted')
scores['lr_jaccard'] = jaccard_score(y_test, yhat,pos_label='PAIDOFF')
print("logistic regression's f1 score: ", scores['lr_f1-score'])
print("logistic regression's jaccard score: ", scores['lr_jaccard'])
print("logistic regression's Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
logistic regression's f1 score:  0.7259684361549498
logistic regression's jaccard score:  0.782608695652174
logistic regression's Accuracy:  0.7857142857142857
```

In [377]:
```
from sklearn.metrics import classification_report, confusion_matrix
import itertools

# Evaluation
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [378]:
```
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, yhat, labels=None)
np.set_printoptions(precision=2)

print (classification_report(y_test, yhat))

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['COLLECTION','PAIDOFF'],normalize= False,  title='Confusion matri
x')
```

```
              precision    recall  f1-score   support

  COLLECTION       0.33      0.07      0.12        14
     PAIDOFF       0.81      0.96      0.88        56

    accuracy                           0.79        70
   macro avg       0.57      0.52      0.50        70
weighted avg       0.71      0.79      0.73        70

Confusion matrix, without normalization
[[ 1 13]
 [ 2 54]]
```



Model Evaluation using Test set

In [311]:
```
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 [312]:
```
!wget -O loan_test.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101
ENv3/labs/loan_test.csv
```

```
--2022-03-07 18:11:27--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML
0101ENv3/labs/loan_test.csv
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.net)... 67.228.25
4.196
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.net)|67.228.2
54.196|: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

2022-03-07 18:11:28 (96.9 MB/s) - 'loan_test.csv' saved [3642/3642]
```

Load Test set for evaluation

In [379]:
```
test_df = pd.read_csv('loan_test.csv')
test_df.head()
```

Out[379]:

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

In [380]:
```
# Convert to date time object
df['due_date'] = pd.to_datetime(df['due_date'])
df['effective_date'] = pd.to_datetime(df['effective_date'])
df['dayofweek'] = df['effective_date'].dt.dayofweek
df['weekend'] = df['dayofweek'].apply(lambda x: 1 if (x>3)  else 0)

# class counts
df.groupby(['Gender'])['loan_status'].value_counts(normalize=True)
df['Gender'].replace(to_replace=['male','female'], value=[0,1],inplace=True)

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

# Use one hot encoding technique to conver categorical varables to binary variables and append them to the fea
ture Data Frame
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)

X_test = Feature
y_test = df['loan_status'].values
X_test = preprocessing.StandardScaler().fit(X_test).transform(X_test)
```

In [387]:
```
test_scores = {}

yhat_knn_test = neigh.predict(X_test)
test_scores['KNN f1-score'] = f1_score(y_test, yhat_knn_test, average='weighted')
test_scores['KNN jaccord'] = jaccard_score(y_test, yhat_knn_test, pos_label='PAIDOFF')

yhat_DT_test = Dec_Tree.predict(X_test)
test_scores['Decision Tree f1-score'] = f1_score(y_test, yhat_DT_test, average='weighted')
test_scores['Decision Tree jaccord'] = jaccard_score(y_test, yhat_DT_test, pos_label='PAIDOFF')

yhat_svm_test = svm_model.predict(X_test)
test_scores['SVM f1-score'] = f1_score(y_test, yhat_svm_test, average='weighted')
test_scores['SVM jaccord'] = jaccard_score(y_test, yhat_svm_test, pos_label='PAIDOFF')

yhat_lr_test = LR.predict(X_test)
y_proba = LR.predict_proba(X_test)
test_scores['lr f1-score'] = f1_score(y_test, yhat_lr_test, average='weighted')
test_scores['lr jaccord'] = jaccard_score(y_test, yhat_lr_test, pos_label='PAIDOFF')
test_scores['lr logloss'] = log_loss(y_test, y_proba)
```

In [388]:
```
test_scores
```

Out[388]:
```
{'KNN f1-score': 0.6736355806123249,
 'KNN jaccord': 0.6862745098039216,
 'Decision Tree f1-score': 0.6304176516942475,
 'Decision Tree jaccord': 0.7407407407407407,
 'SVM f1-score': 0.7583503077293734,
 'SVM jaccord': 0.78,
 'lr f1-score': 0.6717642373556352,
 'lr jaccord': 0.7547169811320755,
 'lr logloss': 0.57423384799027}
```

In [395]:
```
col_names = ['Algorithm','Jaccard', 'F1-score','LogLoss']
report_df = pd.DataFrame(columns = col_names)
report_df.loc[len(report_df)] = ['KNN', test_scores['KNN jaccord'],test_scores['KNN f1-score'], 'NA']
report_df.loc[len(report_df)] = ['Decision Tree', test_scores['Decision Tree jaccord'],test_scores['Decision Tree f1-score'], 'NA']
report_df.loc[len(report_df)] = ['SVM', test_scores['SVM jaccord'],test_scores['SVM f1-score'], 'NA']
report_df.loc[len(report_df)] = ['LogisticRegression', test_scores['lr jaccord'],test_scores['lr f1-score'], test_scores['lr logloss'] ]
report_df.style.hide_index()
```

Out[395]:

Algorithm	Jaccard	F1-score	LogLoss
KNN	0.686275	0.673636	NA
Decision Tree	0.740741	0.630418	NA
SVM	0.780000	0.758350	NA
LogisticRegression	0.754717	0.671764	0.574234

Report

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

Algorithm	Jaccard	F1-score	LogLoss
KNN	?	?	NA
Decision Tree	?	?	NA
SVM	?	?	NA
LogisticRegression	?	?	?

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 (http://cocl.us/ML0101EN-SPSSModeler?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkML0101ENSkillsNetwork20718538-2021-01-01)

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 (https://cocl.us/ML0101EN_DSX?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkML0101ENSkillsNetwork20718538-2021-01-01)

Thanks for completing this lesson!

Author: Saeed Aghabozorgi (https://ca.linkedin.com/in/saeedaghabozorgi?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkML0101ENSkillsNetwork20718538-2021-01-01?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkML0101ENSkillsNetwork20718538-2021-01-01)

Saeed Aghabozorgi (https://ca.linkedin.com/in/saeedaghabozorgi), 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
