Machine Learning Classification with Python

14 Final Project - Loan Prediction

This notebook provides practice of all the classification algorithms learned in the IBM Machine Learning with Python (https://www.coursera.org/learn/machine-learning-with-python/) 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.

Jump to Sections

- Classification Models
- Model Evaluation using Test Set
- Final Report of Model Accuracy Comparisons

Initial Analysis

Lets 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
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

Lets download the dataset

In [2]:
```
!wget -O loan_train.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf
-courses-data/CognitiveClass/ML0101ENv3/labs/loan_train.csv
```

```
--2020-06-10 10:56:01--  https://s3-api.us-geo.objectstorage.softlayer.net/cf
-courses-data/CognitiveClass/ML0101ENv3/labs/loan_train.csv
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstor
age.softlayer.net)... 67.228.254.196
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.object
storage.softlayer.net)|67.228.254.196|: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 0.1s

2020-06-10 10:56:02 (180 KB/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	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 [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	Gende
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 [6]:
```
df['loan_status'].value_counts()
```

Out[6]:
```
PAIDOFF       260
COLLECTION     86
Name: loan_status, dtype: int64
```

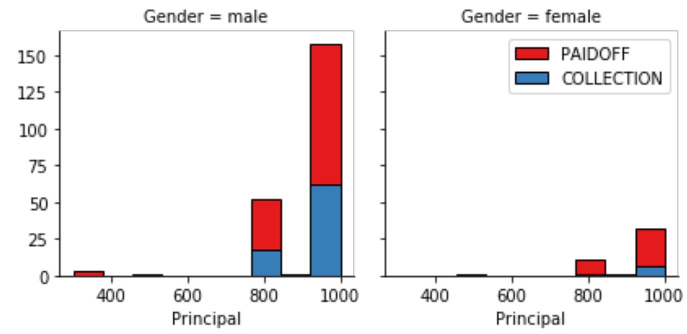
260 people have paid off the loan on time while 86 have gone into collection

Lets plot some columns to underestand data better:

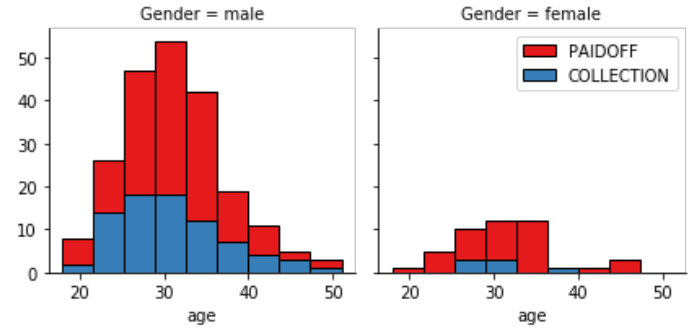
In [7]:
```
# notice: installing seaborn might takes a few minutes
!conda install -c anaconda seaborn -y
```

```
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_w
rap=2)
g.map(plt.hist, 'Principal', bins=bins, ec="k")

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



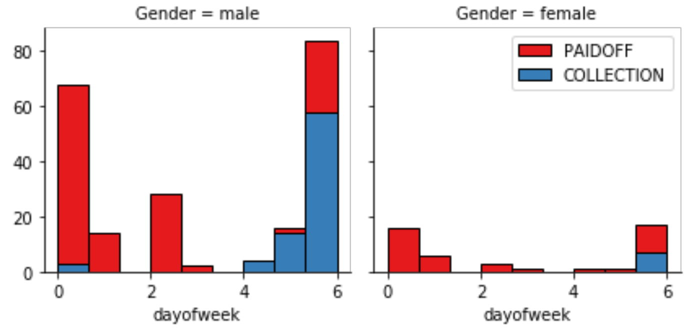
In [9]:
```
bins = np.linspace(df.age.min(), df.age.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_w
rap=2)
g.map(plt.hist, 'age', bins=bins, ec="k")

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



Pre-processing: Feature selection/extraction

Lets look at the day of the week people get the loan

In [10]:
```python
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_w
rap=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 dont pay it off, so lets use Feature binarization to set a threshold values less then day 4

In [11]:
```python
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	Gende
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

Convert Categorical features to numerical values

Lets look at gender:

In [12]:
```python
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 their loan

Lets 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	Gende
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	1
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	1
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
```

Feature befor 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
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

Lets defind 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

Training set to be used to train the model, we then model the accuracy of our model through our test set.

The following algorithms are utilised:

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

Training and Test Set Splitting

In [20]:
```
from sklearn.model_selection import train_test_split
X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y, test_s
ize=0.2, random_state=4)
print ('Train set shapes:', X_trainset.shape,  y_trainset.shape)
print ('Test set shapes:', X_testset.shape,  y_testset.shape)
```

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

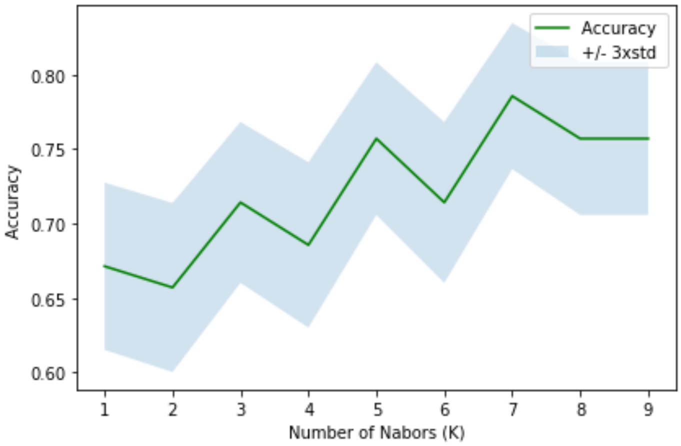
K Nearest Neighbor (KNN)

We calculate the accuracy of KNN for different K values

In [21]:
```
# classification

from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics

# calculate the accuracy of KNN for different K's
Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfustionMx = [];
for n in range(1,Ks):

    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_trainset, y_trainse
t)
    yhat = neigh.predict(X_testset)
    #print('yhat for n =', n, ':', yhat)
    mean_acc[n-1] = metrics.accuracy_score(y_testset, yhat)


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

mean_acc
```

Out[21]:
```
array([0.67142857, 0.65714286, 0.71428571, 0.68571429, 0.75714286,
       0.71428571, 0.78571429, 0.75714286, 0.75714286])
```

In [22]:
```
# plot model accuracy for different k values

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=0.20)
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Nabors (K)')
plt.tight_layout()
plt.show()

# determine the best k value and its accuracy
print( "The best accuracy was with", mean_acc.max(), "with k =", mean_acc.ar
gmax()+1)
```




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

Decision Tree

In [23]:
```
# set up decision tree

from sklearn.tree import DecisionTreeClassifier
```

In [24]:
```
# model - create instance of DecisionTreeClassifier
loanTree = DecisionTreeClassifier(criterion='entropy', max_depth=4)
loanTree # shows the default parameters

# model - fit the data with the training feature matrix and the training res
ponse vector
loanTree.fit(X_trainset, y_trainset)
```

Out[24]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=4, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [25]:
```
# predict on testing dataset
predictionTree = loanTree.predict(X_testset)
print(predictionTree) # print predictions

# evaluate

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

```
['COLLECTION' 'COLLECTION' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF'
 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'COLLECTION' 'PAIDOFF' 'COLLECTION'
 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'COLLECTION' 'PAIDOFF'
 'COLLECTION' 'PAIDOFF' 'PAIDOFF' 'COLLECTION' 'COLLECTION' 'COLLECTION'
 'PAIDOFF' 'COLLECTION' 'COLLECTION' 'PAIDOFF' 'COLLECTION' 'PAIDOFF'
 'COLLECTION' 'COLLECTION' 'COLLECTION' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF'
 'COLLECTION' 'PAIDOFF' 'COLLECTION' 'PAIDOFF' 'COLLECTION' 'PAIDOFF'
 'PAIDOFF' 'COLLECTION' 'PAIDOFF' 'COLLECTION' 'COLLECTION' 'COLLECTION'
 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF'
 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'COLLECTION' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF'
 'PAIDOFF' 'COLLECTION' 'PAIDOFF' 'COLLECTION' 'PAIDOFF' 'COLLECTION'
 'PAIDOFF' 'PAIDOFF']
DecisionTrees's Accuracy:  0.6142857142857143
```

Support Vector Machine

In [26]:
```
# model - fit the data
from sklearn import svm
clf = svm.SVC(kernel='rbf')
clf.fit(X_trainset, y_trainset)
```

Out[26]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

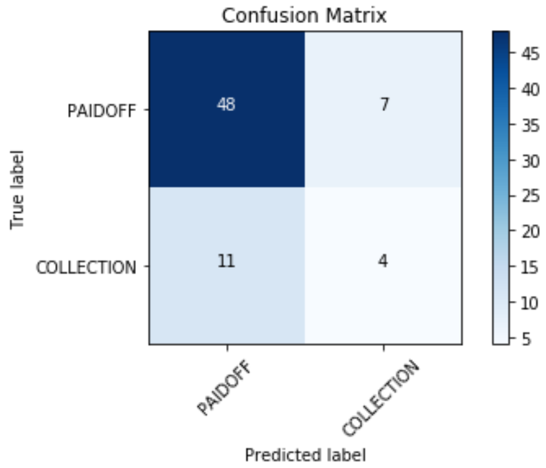
In [60]:
```python
# predict on testing dataset
yhat = clf.predict(X_testset)
print(yhat) # print predictions

# evaluate
from sklearn.metrics import f1_score
from sklearn.metrics import jaccard_similarity_score

print("Avg F1-score: %.4f" % f1_score(y_testset, yhat, average='weighted'))
print("Jaccard score: %.4f" % jaccard_similarity_score(y_testset, yhat))
```

```
['PAIDOFF' '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' 'PAIDOFF'
 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF'
 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'COLLECTION' 'PAIDOFF'
 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'COLLECTION' 'PAIDOFF'
 'COLLECTION' 'PAIDOFF' 'COLLECTION' 'PAIDOFF' 'PAIDOFF']
Avg F1-score: 0.7584
Jaccard score: 0.7963
```

In [28]:

```python
# visualise confusion matrix

from sklearn.metrics import classification_report, confusion_matrix
import itertools

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')

# Compute confusion matrix
cnf_matrix = confusion_matrix(y_testset, yhat, labels=['PAIDOFF','COLLECTION
'])
np.set_printoptions(precision=2)

print (classification_report(y_testset, yhat))

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

```
              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
[[48  7]
 [11  4]]
```



Logistic Regression

In [29]:
```
from sklearn.linear_model import LogisticRegression
lrm = LogisticRegression(C=0.01, solver='liblinear').fit(X_trainset,y_trains
et)
```

In [30]:
```
# predict
yhat = lrm.predict(X_testset)
print(yhat) # print predictions

# predict estimates for all classes
yhat_probabilities = lrm.predict_proba(X_testset)
```

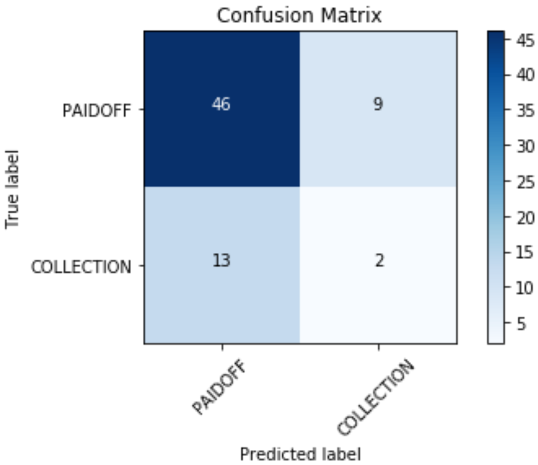
```
['COLLECTION' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF'
 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF' 'PAIDOFF'
 'PAIDOFF' 'PAIDOFF' 'COLLECTION' 'PAIDOFF' 'COLLECTION' 'PAIDOFF'
 'PAIDOFF' 'PAIDOFF' 'COLLECTION' 'PAIDOFF' 'PAIDOFF' 'COLLECTION'
 'COLLECTION' 'PAIDOFF'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

In [56]:
```
# evaluate
from sklearn.metrics import f1_score
from sklearn.metrics import jaccard_similarity_score

print("Avg F1-score: %.4f" % f1_score(y_testset, yhat, average='weighted'))
print("Jaccard score: %.4f" % jaccard_similarity_score(y_testset, yhat))
```

```
Avg F1-score: 0.7584
Jaccard score: 0.7963
```

In [32]:
```
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_testset, yhat, labels=['PAIDOFF','COLLECTION
'])
np.set_printoptions(precision=2)

print (classification_report(y_testset, yhat))

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

```
              precision    recall  f1-score   support

  COLLECTION       0.18      0.13      0.15        15
     PAIDOFF       0.78      0.84      0.81        55

    accuracy                           0.69        70
   macro avg       0.48      0.48      0.48        70
weighted avg       0.65      0.69      0.67        70

Confusion matrix, without normalization
[[46  9]
 [13  2]]
```



Model Evaluation using Test Set

In [33]:
```
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
```

First, download and load the test set:

In [34]:
```
!wget -O loan_test.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_test.csv
```

```
--2020-06-10 10:56:17--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_test.csv
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.net)... 67.228.254.196
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.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

2020-06-10 10:56:17 (1.27 GB/s) - 'loan_test.csv' saved [3642/3642]
```

Load Test set for evaluation

In [35]:
```
test_df = pd.read_csv('loan_test.csv')
test_df.head()
```

Out[35]:

	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 [36]:
```
# perform preprocessing

# convert dates
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
# change gender to numeric
test_df['Gender'].replace(to_replace=['male','female'], value=[0, 1], inplac
e=True)
# determine if weekend or not
test_df['weekend'] = test_df['dayofweek'].apply(lambda x: 1 if (x>3)  else
0)
# get education from test dataframe
education_dummy = pd.get_dummies(test_df['education'])
education_dummy = education_dummy[['Bechalor','High School or Below','colleg
e']]

# create a test feature
test_feature = test_df[['Principal','terms','age','Gender','weekend']]
test_feature = pd.concat([test_feature, education_dummy], axis=1) # add educ
ation dummy columns

X_testset = test_feature # setup X values for testset
# fit to standard scalar
X_testset = preprocessing.StandardScaler().fit(X_testset).transform(X_testse
t)
y_testset = test_df['loan_status'] # setup y values for testset
```

Run Model Evaluations

KNN

In [37]:
```
n_accuracies = []
for n in range(1, 75):
    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_trainset, y_trainse
t)
    yhat = neigh.predict(X_testset)
    # get and store accuracy
    accuracy = metrics.accuracy_score(yhat, y_testset)
    n_accuracies.append(accuracy)

# print details of the maximum
max_accuracy = max(n_accuracies)
max_accuracy_k = n_accuracies.index(max_accuracy) + 1
print('K Value', max_accuracy_k, 'gives accuracy', max_accuracy)
```

```
K Value 24 gives accuracy 0.7962962962962963
```

In [38]:
```
# run model on this K Value
neigh = KNeighborsClassifier(n_neighbors = max_accuracy_k).fit(X_trainset, y
_trainset)
yhat = neigh.predict(X_testset)
```

In [55]:
```
# get accuracy scores from this run
print("KNN Avg F1-score: %.3f" % f1_score(y_testset, yhat, average='weighted
'))
print("KNN Jaccard score: %.3f" % jaccard_similarity_score(y_testset, yhat))
```

```
KNN Avg F1-score: 0.758
KNN Jaccard score: 0.796
```

Decision Tree

In [40]:
```
# predict on our testset
yhat = loanTree.predict(X_testset)
```

In [54]:
```
# get accuracy scores
print("Decision Tree Avg F1-score: %.3f" % f1_score(y_testset, yhat, average
='weighted'))
print("Decision Tree Jaccard score: %.3f" % jaccard_similarity_score(y_tests
et, yhat))
```

```
Decision Tree Avg F1-score: 0.758
Decision Tree Jaccard score: 0.796
```

Support Vector Machine

In [42]:
```
# predict on our testset
yhat = clf.predict(X_testset)
```

In [53]:
```
# get accuracy scores
print("SVM Avg F1-score: %.3f" % f1_score(y_testset, yhat, average='weighted
'))
print("SVM Jaccard score: %.3f" % jaccard_similarity_score(y_testset, yhat))
```

```
SVM Avg F1-score: 0.758
SVM Jaccard score: 0.796
```

Logistic Regression

In [44]:
```
# predict on our testset
yhat = lrm.predict(X_testset)
# predict estimates for all classes
yhat_probabilities = lrm.predict_proba(X_testset)
```

In [51]:
```
# get accuracy scores
print("Logistic Regression Avg F1-score: %.3f" % f1_score(y_testset, yhat, a
verage='weighted'))
print("Logistic Regression Jaccard score: %.3f" % jaccard_similarity_score(y
_testset, yhat))
print("Logistic Regression LogLoss score: %.3f" % log_loss(y_testset, yhat_p
robabilities))
```

```
Logistic Regression Avg F1-score: 0.758
Logistic Regression Jaccard score: 0.796
Logistic Regression LogLoss score: 0.567
```

Report

Accuracy of the built models using different evaluation metrics:

Algorithm	Jaccard	F1-score	LogLoss
KNN	0.741	0.660	NA
Decision Tree	0.722	0.737	NA
SVM	0.796	0.758	NA
Logistic Regression	0.741	0.660	0.567

Course Credits

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