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
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

Field	Description
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.cloud
```

Load Data From CSV File

In [13]:

```
top\Programming\Data Science\IBM\Machine Learning with Python\Week 6\Guide\loan_train.csv')
```

Out[13]:

	loan_status	Principal	terms	effective_date	due_date	age	education	Gender
0	PAIDOFF	1000	30	9/8/2016	10/7/2016	45	High School or Below	male
1	PAIDOFF	1000	30	9/8/2016	10/7/2016	33	Bechalor	female
2	PAIDOFF	1000	15	9/8/2016	9/22/2016	27	college	male
3	PAIDOFF	1000	30	9/9/2016	10/8/2016	28	college	female
4	PAIDOFF	1000	30	9/9/2016	10/8/2016	29	college	male
...	...	...	...	...	...	...	...	...
341	COLLECTION	800	15	9/11/2016	9/25/2016	32	High School or Below	male
342	COLLECTION	1000	30	9/11/2016	10/10/2016	25	High School or Below	male
343	COLLECTION	800	15	9/12/2016	9/26/2016	39	college	male
344	COLLECTION	1000	30	9/12/2016	11/10/2016	28	college	male
345	COLLECTION	1000	30	9/12/2016	10/11/2016	26	college	male

346 rows × 8 columns

In [14]:

```
df.shape
```

Out[14]:

```
(346, 8)
```

Convert to date time object

In [15]:

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

Out[15]:

	loan_status	Principal	terms	effective_date	due_date	age	education	Gender
0	PAIDOFF	1000	30	2016-09-08	2016-10-07	45	High School or Below	male
1	PAIDOFF	1000	30	2016-09-08	2016-10-07	33	Bechalor	female
2	PAIDOFF	1000	15	2016-09-08	2016-09-22	27	college	male
3	PAIDOFF	1000	30	2016-09-09	2016-10-08	28	college	female
4	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 [16]:

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

Out[16]:

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

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

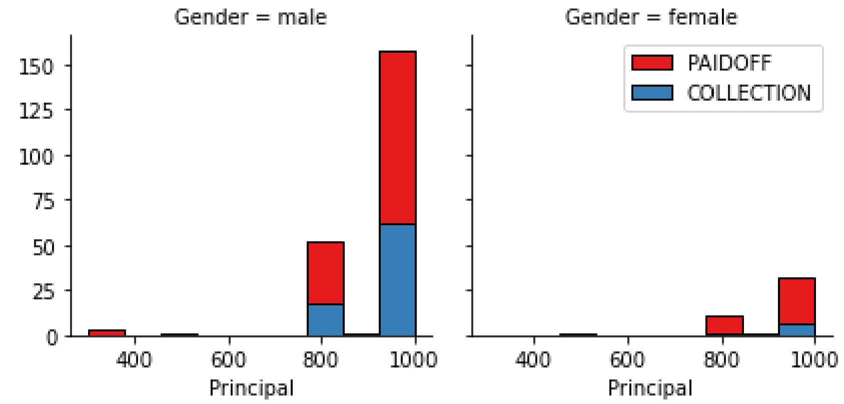
Let's plot some columns to underestand data better:

In [17]:

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

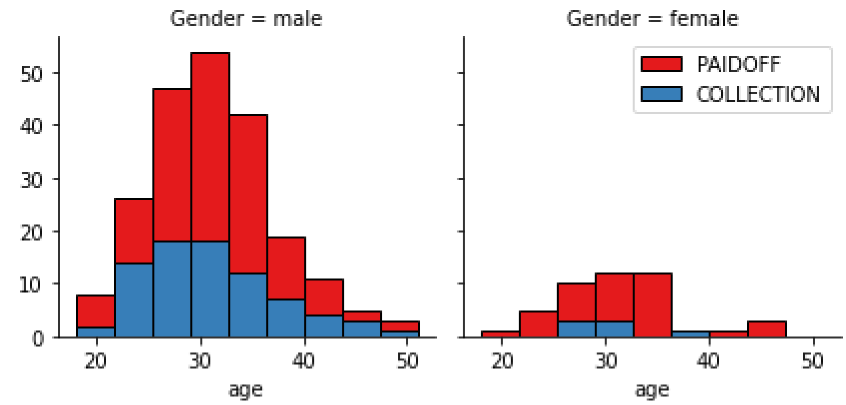
In [18]:

```
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 [19]:

```
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()
```

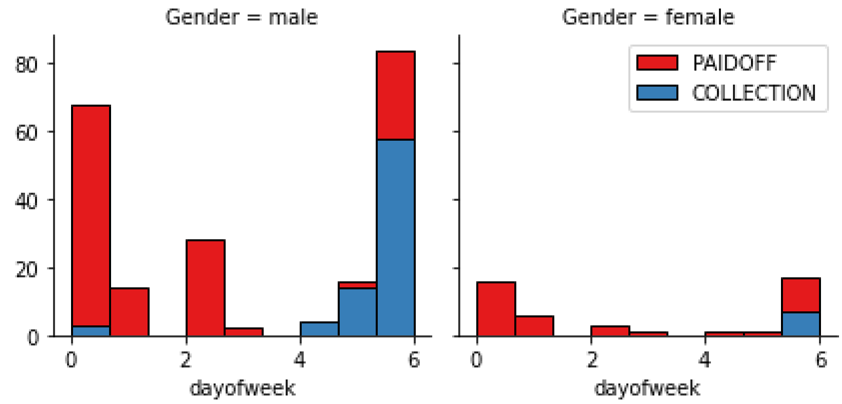


Pre-processing: Feature selection/extraction

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

In [20]:

```
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 [21]:

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

Out[21]:

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

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

Out[22]:

```
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 [23]:

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

Out[23]:

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

One Hot Encoding

How about education?

In [24]:

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

Out[24]:

```
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 [25]:

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

Out[25]:

	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 [26]:

```
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[26]:

	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 [27]:

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

Out[27]:

	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 [28]:

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

Out[28]:

```
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 [29]:

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

Out[29]:

```
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 [30]:

```
# We split the X into train and test to find the best k
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 [94]:

```
from sklearn.neighbors import KNeighborsClassifier
k = 3
#Train Model and Predict
knn_model = KNeighborsClassifier(n_neighbors=k).fit(X_train,y_train)
knn_model
```

Out[94]:

```
KNeighborsClassifier(n_neighbors=3)
```

In [95]:

```
yhat = knn_model.predict(X_test)
yhat[0:5]
```

Out[95]:

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

In [96]:

```
# Best k
Ks=15
mean_acc=np.zeros((Ks-1))
std_acc=np.zeros((Ks-1))
ConfustionMx=[];
for n in range(1,Ks):

    #Train Model and Predict
    knn_model = KNeighborsClassifier(n_neighbors=n).fit(X_train,y_train)
    yhat = knn_model.predict(X_test)


    mean_acc[n-1]=np.mean(yhat==y_test);

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

Out[96]:

```
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 [97]:

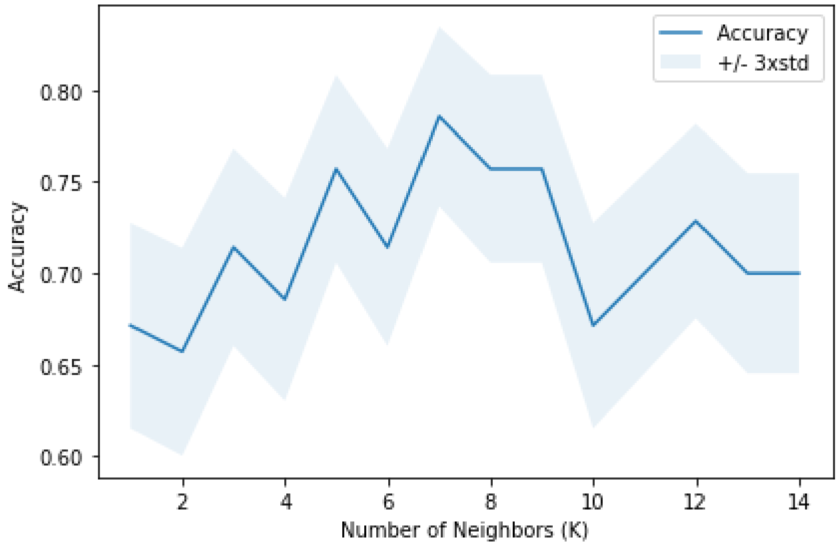
```
# Building the model again, using k=7
from sklearn.neighbors import KNeighborsClassifier
k = 7
#Train Model and Predict
knn_model = KNeighborsClassifier(n_neighbors=k).fit(X_train,y_train)
knn_model
```

Out[97]:

```
KNeighborsClassifier(n_neighbors=7)
```

In [98]:

```
plt.plot(range(1,Ks),mean_acc)
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()

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

neigh = KNeighborsClassifier(n_neighbors=mean_acc.argmax()+1).fit(X_train, y_train)
```




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

In [99]:

```
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
```

Decision Tree

In [102]:

```
from sklearn.tree import DecisionTreeClassifier
tree= DecisionTreeClassifier(criterion="entropy", max_depth = 4)
tree.fit(X_train,y_train)
tree
```

Out[102]:

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

In [103]:

```
yhat = tree.predict(X)
yhat
```

Out[103]:

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

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

Support Vector Machine

In [104]:

```
from sklearn import svm
SVM = svm.SVC()
SVM.fit(X_train, y_train)
```

Out[104]:

```
SVC()
```

In [105]:

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

Out[105]:

```
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)
```

Logistic Regression

In [108]:

```
from sklearn.linear_model import LogisticRegression
LogRmodel = LogisticRegression(C=0.01).fit(X_train,y_train)
LogRmodel
```

Out[108]:

```
LogisticRegression(C=0.01)
```

In [109]:

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

Out[109]:

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

Model Evaluation using Test set

In [110]:

```
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 [111]:

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

Load Test set for evaluation

In [112]:

```
test_df = pd.read_csv(r'C:\Users\Lucas.Tan\Desktop\Programming\Data Science\IBM\Machine Lea
test_df
```

Out[112]:

	loan_status	Principal	terms	effective_date	due_date	age	education	Gender
0	PAIDOFF	1000	30	9/8/2016	10/7/2016	50	Bechalor	female
1	PAIDOFF	300	7	9/9/2016	9/15/2016	35	Master or Above	male
2	PAIDOFF	1000	30	9/10/2016	10/9/2016	43	High School or Below	female
3	PAIDOFF	1000	30	9/10/2016	10/9/2016	26	college	male
4	PAIDOFF	800	15	9/11/2016	9/25/2016	29	Bechalor	male
5	PAIDOFF	700	15	9/11/2016	9/25/2016	33	High School or Below	male
6	PAIDOFF	1000	15	9/11/2016	9/25/2016	24	college	male
7	PAIDOFF	1000	30	9/11/2016	10/10/2016	32	Bechalor	male
8	PAIDOFF	800	15	9/11/2016	9/25/2016	27	college	female
9	PAIDOFF	1000	15	9/11/2016	9/25/2016	37	college	male
10	PAIDOFF	800	15	9/11/2016	9/25/2016	24	High School or Below	male
11	PAIDOFF	300	7	9/11/2016	9/17/2016	35	college	male
12	PAIDOFF	1000	30	9/11/2016	10/10/2016	31	Bechalor	male
13	PAIDOFF	1000	30	9/11/2016	10/10/2016	37	college	female
14	PAIDOFF	1000	30	9/11/2016	10/10/2016	37	High School or Below	female
15	PAIDOFF	1000	30	9/11/2016	11/9/2016	33	college	male
16	PAIDOFF	800	15	9/11/2016	9/25/2016	43	Bechalor	male
17	PAIDOFF	1000	7	9/11/2016	9/17/2016	32	Bechalor	female
18	PAIDOFF	1000	15	9/11/2016	9/25/2016	26	High School or Below	male
19	PAIDOFF	1000	7	9/11/2016	9/17/2016	29	High School or Below	male
20	PAIDOFF	1000	30	9/11/2016	10/10/2016	30	college	male
21	PAIDOFF	1000	7	9/11/2016	9/17/2016	27	High School or Below	male
22	PAIDOFF	300	7	9/12/2016	9/18/2016	37	Master or Above	male
23	PAIDOFF	1000	15	9/12/2016	10/26/2016	29	college	male
24	PAIDOFF	1000	15	9/12/2016	9/26/2016	26	Bechalor	male
25	PAIDOFF	800	30	9/12/2016	10/11/2016	28	college	male
26	PAIDOFF	1000	30	9/12/2016	10/11/2016	38	college	male
27	PAIDOFF	1000	30	9/12/2016	10/11/2016	46	college	male
28	PAIDOFF	1000	30	9/12/2016	10/11/2016	33	Bechalor	male

	loan_status	Principal	terms	effective_date	due_date	age	education	Gender
29	PAIDOFF	1000	30	9/12/2016	11/10/2016	29	college	male
30	PAIDOFF	1000	30	9/12/2016	10/11/2016	29	college	male
31	PAIDOFF	1000	15	9/12/2016	9/26/2016	36	High School or Below	male
32	PAIDOFF	1000	30	9/12/2016	11/10/2016	29	college	male
33	PAIDOFF	1000	30	9/12/2016	10/11/2016	30	college	male
34	PAIDOFF	1000	15	9/12/2016	9/26/2016	36	High School or Below	male
35	PAIDOFF	1000	30	9/13/2016	10/12/2016	29	college	male
36	PAIDOFF	1000	30	9/13/2016	10/12/2016	28	High School or Below	male
37	PAIDOFF	800	15	9/13/2016	9/27/2016	23	college	male
38	PAIDOFF	1000	30	9/14/2016	10/13/2016	38	High School or Below	female
39	PAIDOFF	1000	30	9/14/2016	10/13/2016	30	college	female
40	COLLECTION	1000	30	9/9/2016	10/8/2016	33	High School or Below	male
41	COLLECTION	1000	15	9/10/2016	9/24/2016	31	High School or Below	female
42	COLLECTION	800	15	9/10/2016	9/24/2016	41	college	male
43	COLLECTION	1000	30	9/10/2016	10/9/2016	30	college	male
44	COLLECTION	800	15	9/10/2016	9/24/2016	26	High School or Below	female
45	COLLECTION	1000	30	9/10/2016	10/9/2016	20	High School or Below	male
46	COLLECTION	1000	15	9/10/2016	10/9/2016	26	High School or Below	male
47	COLLECTION	1000	30	9/11/2016	10/10/2016	24	High School or Below	female
48	COLLECTION	800	15	9/11/2016	9/25/2016	27	college	male
49	COLLECTION	1000	30	9/11/2016	10/10/2016	32	High School or Below	male
50	COLLECTION	800	15	9/11/2016	9/25/2016	29	college	male
51	COLLECTION	1000	30	9/11/2016	10/10/2016	37	High School or Below	male
52	COLLECTION	800	15	9/11/2016	9/25/2016	36	High School or Below	male
53	COLLECTION	1000	30	9/12/2016	10/11/2016	33	High School or Below	male

In [115]:

```
# convert date time
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 [116]:

```
# evaulate weekend field
test_df['weekend'] = test_df['dayofweek'].apply(lambda x: 1 if (x>3)  else 0)
```

In [117]:

```
# work out education level
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[117]:

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

In [123]:

```
# normalize the test data
try:
    test_X = preprocessing.StandardScaler().fit(test_feature).transform(test_feature)
    test_X[0:5]
except:
    pass
```

In [124]:

```
# and target result
test_y = test_df['loan_status'].values
test_y[0:5]
```

Out[124]:

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

In [130]:

```
knn_pred=neigh.predict(TestX)
jc1=jaccard_score(TestY, knn_pred,average=None)
fs1=f1_score(TestY, knn_pred, average='weighted')

tree_pred=DT_model.predict(TestX)
jc2=jaccard_score(TestY, tree_pred,average=None)
fs2=f1_score(TestY, tree_pred, average='weighted')

svm_pred=SVM_model.predict(TestX)
jc3=jaccard_score(TestY, svm_pred,average=None)
fs3=f1_score(TestY, svm_pred, average='weighted')

log_pred=LR_model.predict(TestX)
proba=LR_model.predict_proba(TestX)
jc4=jaccard_score(TestY, log_pred,average=None)
fs4=f1_score(TestY, log_pred, average='weighted')
ll4=log_loss(TestY, proba)

jclist = [jc1, jc2, jc3, jc4]
fslist = [fs1, fs2, fs3, fs4]
lllist = ['NA', 'NA', 'NA', ll4]


import pandas as pd

# fomulate the report format
df = pd.DataFrame(jclist, index=['KNN','Decision Tree','SVM','Logistic Regression'],columns
df.insert(loc=1, column='F1-score', value=fslist)
df.insert(loc=2, column='LogLoss', value=lllist)
df.columns.name = 'Algorithm'
df.drop(columns=['extra'])
df = df[['Jaccard','F1-score','LogLoss']]
```

In [131]:

```
df
```

Out[131]:

Algorithm	Jaccard	F1-score	LogLoss
KNN	0.653846	0.632840	NA
Decision Tree	0.659091	0.736682	NA
SVM	0.780000	0.758350	NA
Logistic Regression	0.740741	0.630418	0.516366

Report

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

Algorithm	Jaccard	F1-score	LogLoss
KNN	?	?	NA

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