In this notebook we try to practice all the classification algorithms that we 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.

Lets first load required libraries:

```
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

Lets download the dataset

In [6]:

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

```
--2019-09-22 12:15:21--  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.objectstorage.softlayer.net)... 67.228.254.193
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.net)|67.228.254.193|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 23101 (23K) [text/csv]
Saving to: 'loan_train.csv'

100%[======================================>] 23,101      --.-K/s   in 0.07s

2019-09-22 12:15:21 (303 KB/s) - 'loan_train.csv' saved [23101/23101]
```

Load Data From CSV File

In [7]:

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

Out[7]:

In [8]:

```
df.shape
```

Out[8]:

```
(346, 10)
```

Convert to date time object

In [9]:

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

Out[9]:

Data visualization and pre-processing

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

In [10]:

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

Out[10]:

```
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 [11]:

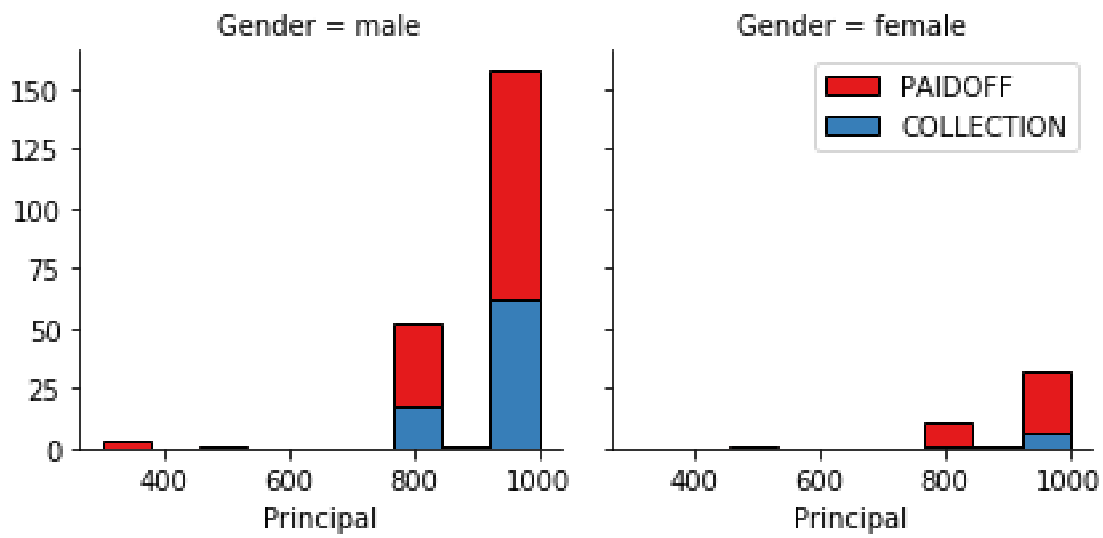
```
# notice: installing seaborn might takes a few minutes
!conda install -c anaconda seaborn -y
```

```
Solving environment: done

## Package Plan ##
```

```
  environment location: /opt/conda/envs/Python36

  added / updated specs:
    - seaborn


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    openssl-1.1.1              |       h7b6447c_0         5.0 MB  anaconda
    seaborn-0.9.0              |           py36_0         379 KB  anaconda
    ca-certificates-2019.5.15  |                1         134 KB  anaconda
    certifi-2019.6.16          |           py36_1         156 KB  anaconda
    ------------------------------------------------------------
                                           Total:         5.7 MB

The following packages will be UPDATED:

    ca-certificates: 2019.5.15-1       --> 2019.5.15-1      anaconda
    certifi:         2019.6.16-py36_1  --> 2019.6.16-py36_1 anaconda
    openssl:         1.1.1d-h7b6447c_1 --> 1.1.1-h7b6447c_0 anaconda
    seaborn:         0.9.0-py36_0      --> 0.9.0-py36_0     anaconda


Downloading and Extracting Packages
openssl-1.1.1        | 5.0 MB    | ##################################### | 10
0%
seaborn-0.9.0        | 379 KB    | ##################################### | 10
0%
ca-certificates-2019 | 134 KB    | ##################################### | 10
0%
certifi-2019.6.16    | 156 KB    | ##################################### | 10
0%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

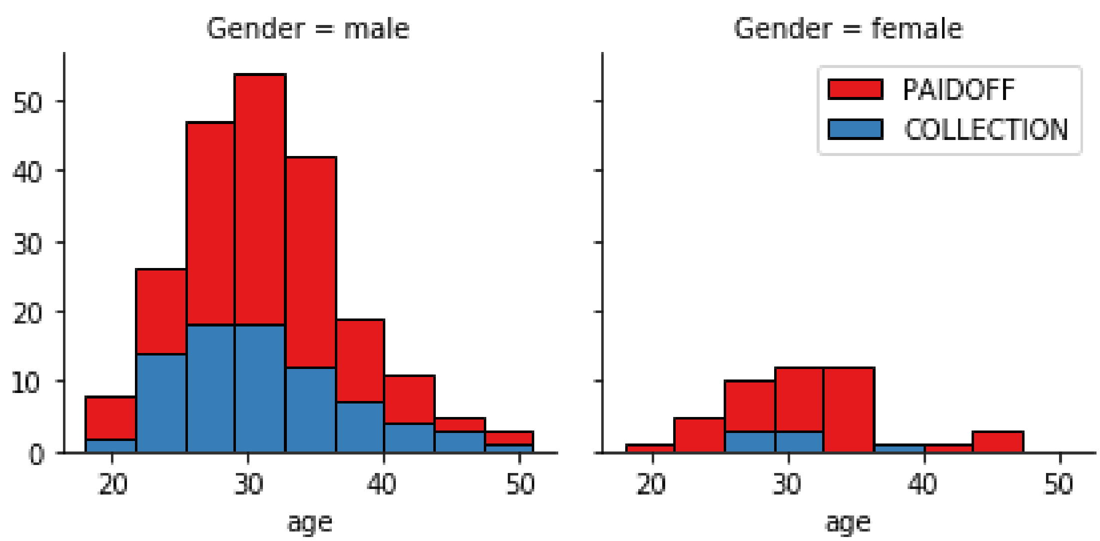
In [12]:

```python
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_wr
ap=2)
g.map(plt.hist, 'Principal', bins=bins, ec="k")

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



In [13]:

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

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



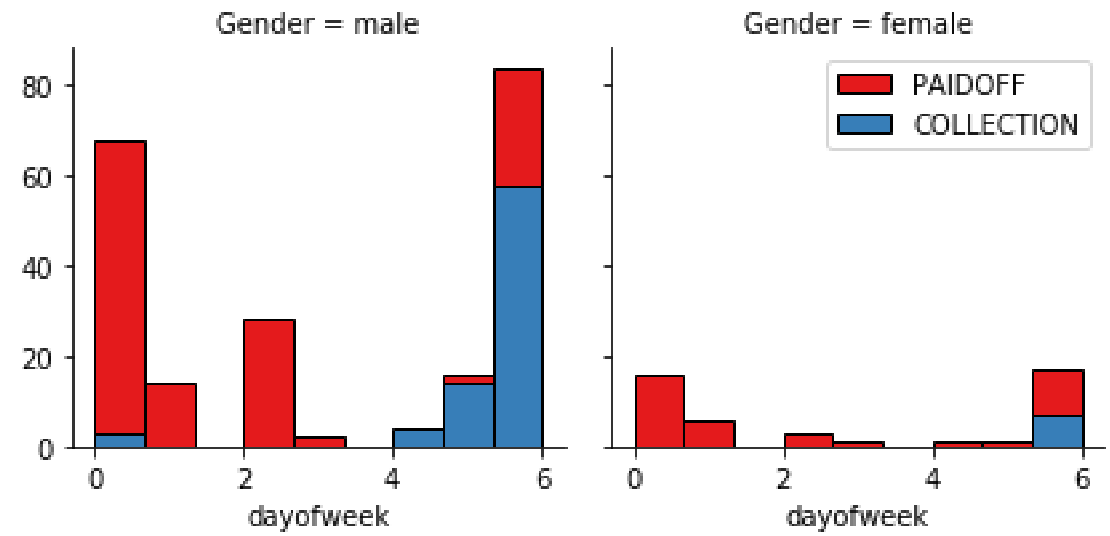
Pre-processing: Feature selection/extraction

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

In [14]:

```
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_wr
ap=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 [15]:

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

Out[15]:

Convert Categorical features to numerical values

Lets look at gender:

In [16]:

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

Out[16]:

```
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

Lets convert male to 0 and female to 1:

In [17]:

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

Out[17]:

One Hot Encoding

How about education?

In [18]:

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

Out[18]:

```
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 [19]:

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

Out[19]:

	Principal	terms	age	Gender	education
0	1000	30	45	0	High School or Below
1	1000	30	33	1	Bechalor

	Principal	terms	age	Gender	education
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 [20]:

```
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[20]:

		Principal			Gender	weekend	Bechalor	[illegible]	[illegible]
		1000			0	0	0		[illegible]
		1000			1	0	1	[illegible]	[illegible]
		1000			0	0	0	[illegible]	
		1000			1	1	0	[illegible]	

	Principal				[illegible]	weekend	Bechalor	[illegible]	[illegible]
	1000				0	1	0	[illegible]	

Feature selection

Lets defind feature sets, X:

In [21]:

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

Out[21]:

	Principal			Gender	weekend	Bechalor	[illegible]	[illegible]
	1000			0	0	0		[illegible]
	1000			1	0	1	[illegible]	[illegible]
	1000			0	0	0	[illegible]	
	1000			1	1	0	[illegible]	

	Principal				[illegible]	weekend	Bechalor	[illegible]	[illegible]
	1000				0	1	0	0	

What are our lables?

In [22]:

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

Out[22]:

```
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 [23]:

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

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/preprocessing/da
ta.py:645: DataConversionWarning: Data with input dtype uint8, int64 were all
converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/opt/conda/envs/Python36/lib/python3.6/site-packages/ipykernel/__main__.py:1:
DataConversionWarning: Data with input dtype uint8, int64 were all converted
to float64 by StandardScaler.
```

```
if __name__ == '__main__':
```

Out[23]:

```
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 [ ]:

```
# Answers for each classification at the end.
```

In [24]:

```
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, rand
om_state=4)
```

In [30]:

```
from sklearn.neighbors import KNeighborsClassifier
k = 5
kNN_type = KNeighborsClassifier(n_neighbors=k).fit(X_train,y_train)
kNN_type
```

Out[30]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform')
```

In [31]:

```
Ks=10
mean_acc=np.zeros((Ks-1))
std_acc=np.zeros((Ks-1))
ConfustionMx=[];
for n in range(1,Ks):


    kNN_type = KNeighborsClassifier(n_neighbors=n).fit(X_train,y_train)
    yhat = kNN_type.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[31]:

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

In [32]:

```
from sklearn.neighbors import KNeighborsClassifier
k = 7

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

Out[32]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=7, p=2,
           weights='uniform')
```

Decision Tree

In [39]:

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

Out[39]:

```
DecisionTreeClassifier(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=False, random_state=None,
            splitter='best')
```

In [40]:

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

Out[40]:

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

Support Vector Machine

In [42]:

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

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'auto' to 'scale'
in version 0.22 to account better for unscaled features. Set gamma explicitly
to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

Out[42]:

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

In [43]:

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

Out[43]:

```
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 [45]:

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

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/linear_model/log
istic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
```

Out[45]:

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [46]:

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

Out[46]:

```
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)
```

Model Evaluation using Test set

In [108]:

```
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 [109]:

```
!wget -O loan_test.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-c
ourses-data/CognitiveClass/ML0101ENv3/labs/loan_test.csv
```

```
--2019-09-22 13:18:07--  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)... 67.228.254.193
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.net)|67.228.254.193|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3642 (3.6K) [text/csv]
Saving to: ‘loan_test.csv’

100%[=====================================>] 3,642       --.-K/s   in 0s

2019-09-22 13:18:07 (468 MB/s) - ‘loan_test.csv’ saved [3642/3642]
```

Load Test set for evaluation

In [110]:

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

Out[110]:

In [111]:

```
#Pretraining dataset to only keep relevant/important information

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
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=T
rue)

#These are the relevant columns
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)

#Including variable for these columns
test_X = preprocessing.StandardScaler().fit(test_Feature).transform(test_Feat
ure)
test_X[0:5]
```

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/preprocessing/da
ta.py:645: DataConversionWarning: Data with input dtype uint8, int64 were all
converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/opt/conda/envs/Python36/lib/python3.6/site-packages/ipykernel/__main__.py:15
: DataConversionWarning: Data with input dtype uint8, int64 were all converte
d to float64 by StandardScaler.
```

Out[111]:

```
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 [112]:

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

Out[112]:

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

In [ ]:

```
# Now is the results for each classifier
```

In [ ]:

```
# Result for K Nearest Neighbor
```

In [113]:

```
knn_result = kNN_type.predict(test_X)

print("K Nearest Neighbor F1-score: %.2f" % f1_score(test_y, knn_result, aver
age='weighted'))
print("K Nearest Neighbor Jaccard index: %.2f" % jaccard_similarity_score(tes
t_y, knn_result))
```

```
K Nearest Neighbor F1-score: 0.63
K Nearest Neighbor Jaccard index: 0.67
```

In [ ]:

```
# Result for Decision Tree
```

In [114]:

```
DT_result = DT_type.predict(test_X)

print("Decision Tree F1-score: %.2f" % f1_score(test_y, DT_result, average='w
eighted'))
print("Decision Tree Jaccard index: %.2f" % jaccard_similarity_score(test_y,
DT_result))
```

```
Decision Tree F1-score: 0.74
Decision Tree Jaccard index: 0.72
```

In [ ]:

```
# Result for Logistic Regression
```

In [115]:

```
LR_result = LR_type.predict(test_X)
```

```python
#for logloss
LR_result_prob = LR_type.predict_proba(test_X)

print("Logistic Regression F1-score: %.2f" % f1_score(test_y, LR_result, average='weighted'))
print("Logistic Regression Jaccard index: %.2f" % jaccard_similarity_score(test_y, LR_result))
#for logloss
print("Logistic Regression LogLoss: %.2f" % log_loss(test_y, LR_result_prob))
```

```
Logistic Regression F1-score: 0.66
Logistic Regression Jaccard index: 0.74
Logistic Regression LogLoss: 0.57
```

In [ ]:

```python
# Support Vector Machine
```

In [116]:

```python
SVM_result = SVM_type.predict(test_X)

print("Support Vector Machine F1-score: %.2f" % f1_score(test_y, SVM_result, average='weighted'))
print("Support Vector Machine Jaccard index: %.2f" % jaccard_similarity_score(test_y, SVM_result))
```

```
Support Vector Machine F1-score: 0.76
Support Vector Machine Jaccard index: 0.80
```

Report

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

Algorithm	**Jaccard**	**F1-score**	**LogLoss**
KNN	0.67	0.63	NA

Algorithm	**Jaccard**	**F1-score**	**LogLoss**
Decision Tree	0.72	0.74	NA
SVM	0.80	0.76	NA
Logistic Regression	0.74	0.66	0.57