predict_loan_case

May 2, 2020

1 In this notebook we will build a classifier to predict whether a loan case will be paid off or not.

We load a historical dataset from previous loan applications, clean the data, and apply different classification algorithm on the data. We are expected to use the following algorithms to build your models:

```
k-Nearest Neighbour
Decision Tree
Support Vector Machine
Logistic Regression
```

The results is reported as the accuracy of each classifier, using the following metrics when these are applicable:

```
Jaccard index
F1-score
LogLoass
```

The Result you will find at the end of the code-block

2 Enjoy

3 RS

[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
```

3.0.1 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

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

```
--2018-06-12 16:16:44--  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… 67.228.254.193
Connecting to 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’

loan_train.csv      100%[===================>]  22.56K  63.3KB/s    in 0.4s

2018-06-12 16:16:45 (63.3 KB/s) - ‘loan_train.csv’ saved [23101/23101]
```

3.0.2 Load Data From CSV File

[3]:
```
df = pd.read_csv('loan_train.csv')
df.head()
```

[3]:
```
   Unnamed: 0  Unnamed: 0.1 loan_status  Principal  terms effective_date  \
0           0             0     PAIDOFF       1000     30       9/8/2016
1           2             2     PAIDOFF       1000     30       9/8/2016
2           3             3     PAIDOFF       1000     15       9/8/2016
3           4             4     PAIDOFF       1000     30       9/9/2016
4           6             6     PAIDOFF       1000     30       9/9/2016
```

```
    due_date  age             education  Gender
0  10/7/2016   45  High School or Below    male
1  10/7/2016   33              Bechalor  female
2  9/22/2016   27               college    male
3  10/8/2016   28               college  female
4  10/8/2016   29               college    male
```

[4]:
```
df.shape
```

[4]:
```
(346, 10)
```

3.0.3 Convert to date time object

[5]:
```
df['due_date'] = pd.to_datetime(df['due_date'])
df['effective_date'] = pd.to_datetime(df['effective_date'])
df.head()
```

[5]:
```
   Unnamed: 0  Unnamed: 0.1 loan_status  Principal  terms effective_date  \
0           0             0     PAIDOFF       1000     30     2016-09-08
1           2             2     PAIDOFF       1000     30     2016-09-08
2           3             3     PAIDOFF       1000     15     2016-09-08
3           4             4     PAIDOFF       1000     30     2016-09-09
4           6             6     PAIDOFF       1000     30     2016-09-09

    due_date  age             education  Gender
0 2016-10-07   45  High School or Below    male
1 2016-10-07   33              Bechalor  female
2 2016-09-22   27               college    male
3 2016-10-08   28               college  female
4 2016-10-08   29               college    male
```

4 Data visualization and pre-processing

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

[6]:
```
df['loan_status'].value_counts()
```

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

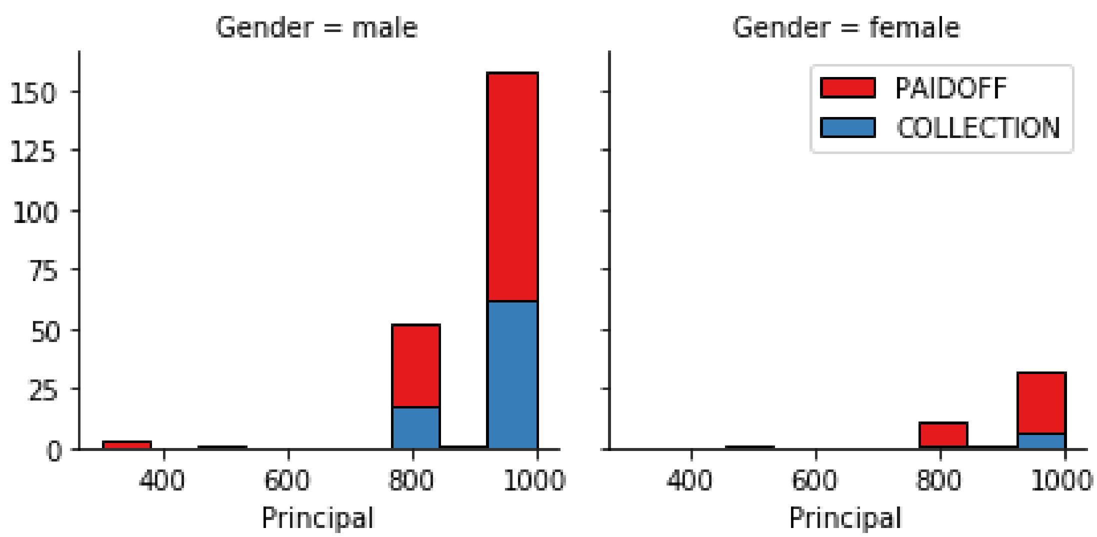
260 people have paid off the loan on time while 86 have gone into collection

Lets plot some columns to underestand data better:

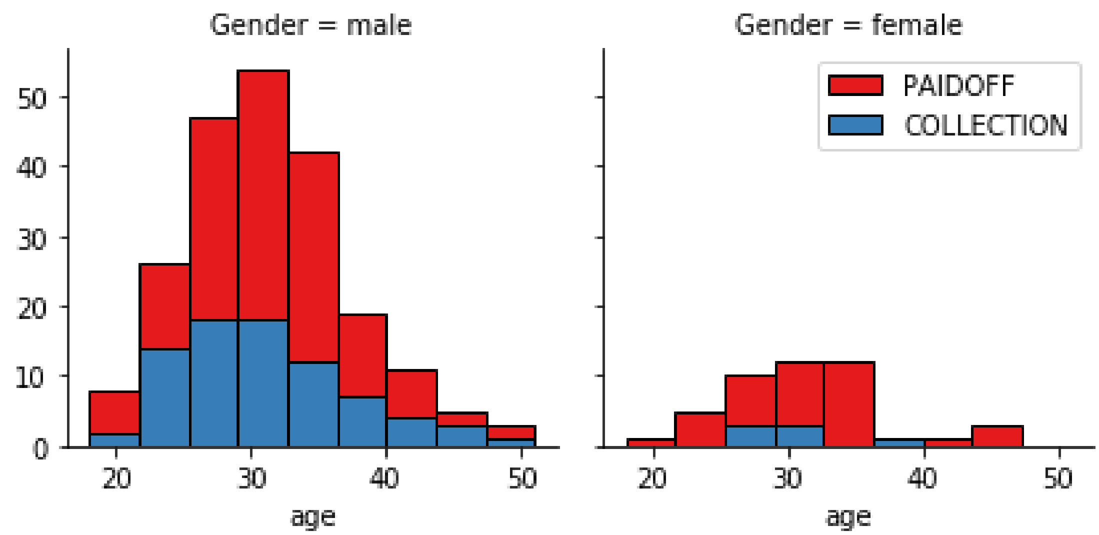
[7]:
```
# notice: installing seaborn might takes a few minutes
!conda install -c anaconda seaborn -y
```

```
Solving environment: done

# All requested packages already installed.
```

[8]:
```
import seaborn as sns

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

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



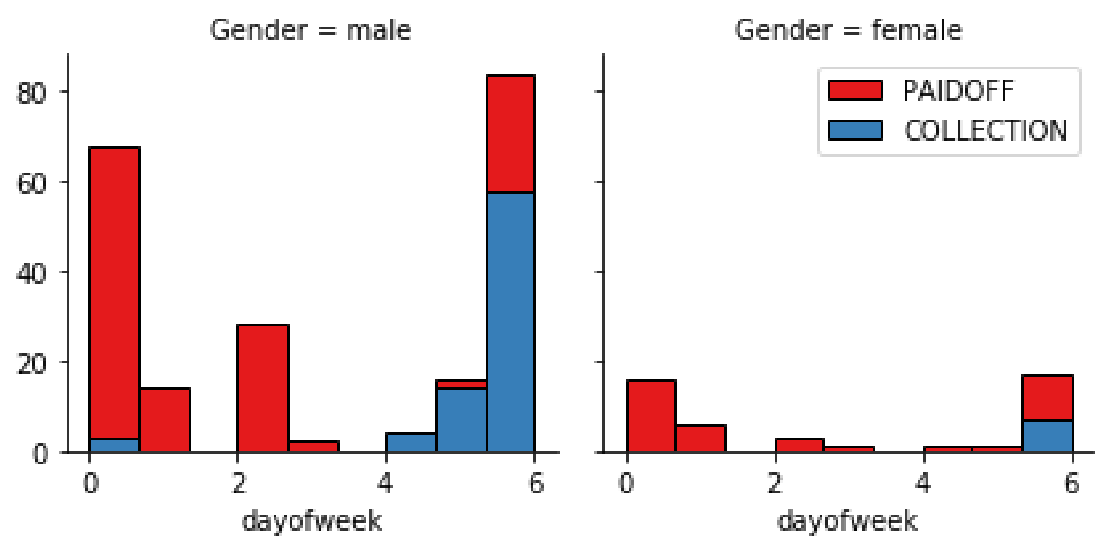
[9]:
```
bins=np.linspace(df.age.min(), df.age.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1",␣
↪col_wrap=2)
g.map(plt.hist, 'age', bins=bins, ec="k")

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



5 Pre-processing: Feature selection/extraction

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

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



We see that people who get the loan at the end of the week dont pay it off, so lets use Feature binarization to set a threshold values less then day 4

[11]:
```
df['weekend']= df['dayofweek'].apply(lambda x: 1 if (x>3)  else 0)
df.head()
```

[11]:
```
   Unnamed: 0  Unnamed: 0.1 loan_status  Principal  terms effective_date  \
0           0             0     PAIDOFF       1000     30     2016-09-08
1           2             2     PAIDOFF       1000     30     2016-09-08
2           3             3     PAIDOFF       1000     15     2016-09-08
3           4             4     PAIDOFF       1000     30     2016-09-09
4           6             6     PAIDOFF       1000     30     2016-09-09

    due_date  age             education  Gender  dayofweek  weekend
0 2016-10-07   45  High School or Below    male          3        0
1 2016-10-07   33              Bechalor  female          3        0
2 2016-09-22   27               college    male          3        0
3 2016-10-08   28               college  female          4        1
4 2016-10-08   29               college    male          4        1
```

5.1 Convert Categorical features to numerical values

Lets look at gender:

[12]:
```
df.groupby(['Gender'])['loan_status'].value_counts(normalize=True)
```

[12]:
```
Gender  loan_status
female  PAIDOFF        0.865385
        COLLECTION     0.134615
male    PAIDOFF        0.731293
        COLLECTION     0.268707
Name: loan_status, dtype: float64
```

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

Lets convert male to 0 and female to 1:

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

[13]:
```
   Unnamed: 0  Unnamed: 0.1 loan_status  Principal  terms effective_date  \
0           0             0     PAIDOFF       1000     30     2016-09-08
1           2             2     PAIDOFF       1000     30     2016-09-08
2           3             3     PAIDOFF       1000     15     2016-09-08
3           4             4     PAIDOFF       1000     30     2016-09-09
```

```
4            6              6     PAIDOFF       1000     30     2016-09-09

     due_date  age             education  Gender  dayofweek  weekend
0  2016-10-07   45  High School or Below       0          3        0
1  2016-10-07   33              Bechalor       1          3        0
2  2016-09-22   27               college       0          3        0
3  2016-10-08   28               college       1          4        1
4  2016-10-08   29               college       0          4        1
```

5.2 One Hot Encoding

How about education?

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

[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

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

[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

[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()
```

[16]:
```
   Principal  terms  age  Gender  weekend  Bechalor  High School or Below  \
0       1000     30   45       0        0         0                     1
1       1000     30   33       1        0         1                     0
2       1000     15   27       0        0         0                     0
3       1000     30   28       1        1         0                     0
4       1000     30   29       0        1         0                     0

   college
0        0
1        0
2        1
3        1
4        1
```

5.2.1 Feature selection

Lets defind feature sets, X:

[17]:
```
X = Feature
X[0:5]
```

[17]:
```
   Principal  terms  age  Gender  weekend  Bechalor  High School or Below  \
0       1000     30   45       0        0         0                     1
1       1000     30   33       1        0         1                     0
2       1000     15   27       0        0         0                     0
3       1000     30   28       1        1         0                     0
4       1000     30   29       0        1         0                     0

   college
0        0
1        0
2        1
3        1
4        1
```

What are our lables?

[18]:
```
y = df['loan_status'].values
y[0:5]
```

[18]:
```
array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF'], dtype=object)
```

5.3 Normalize Data

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

[19]:
```
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

[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]])
```

6 Classification

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

7 K Nearest Neighbor(KNN)

We should find the best k to build the model with the best accuracy.
We should split our train_loan.csv into train and test to find the best **k**.

[21]:
```
# 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,)
```

[45]:
```
# Modeling
from sklearn.neighbors import KNeighborsClassifier
k = 3
#Train Model and Predict
kNN_model = KNeighborsClassifier(n_neighbors=k).fit(X_train,y_train)
kNN_model
```

[45]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=3, p=2,
           weights='uniform')
```

[46]:
```
# just for sanity chaeck
yhat = kNN_model.predict(X_test)
yhat[0:5]
```

[46]:
```
array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF'], dtype=object)
```

[67]:
```
# 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
```

[67]:
```
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       ])
```

[68]:
```
# 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
```

[68]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=7, p=2,
           weights='uniform')
```

8 Decision Tree

[84]:
```
from sklearn.tree import DecisionTreeClassifier
DT_model = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
DT_model.fit(X_train,y_train)
DT_model
```

[84]:
```
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')
```

[85]:
```
yhat = DT_model.predict(X_test)
yhat
```

[85]:
```
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)
```

9 Support Vector Machine

[77]:
```
from sklearn import svm
SVM_model = svm.SVC()
SVM_model.fit(X_train, y_train)
```

[77]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

[82]:
```
yhat = SVM_model.predict(X_test)
yhat
```

```
[82]: 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)
```

10 Logistic Regression

```
[80]: from sklearn.linear_model import LogisticRegression
      LR_model = LogisticRegression(C=0.01).fit(X_train,y_train)
      LR_model
```

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

```
[81]: yhat = LR_model.predict(X_test)
      yhat
```

```
[81]: 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)
```

11 Model Evaluation using Test set

[93]:
```
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:

[94]:
```
!wget -O loan_test.csv https://s3-api.us-geo.objectstorage.softlayer.net/
↪cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_test.csv
```

```
--2018-06-12 17:42:41--  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… 67.228.254.193
Connecting to 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’

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

2018-06-12 17:42:42 (105 MB/s) - ‘loan_test.csv’ saved [3642/3642]
```

11.0.1 Load Test set for evaluation

[100]:
```
test_df = pd.read_csv('loan_test.csv')
test_df.head()
```

[100]:
```
   Unnamed: 0  Unnamed: 0.1 loan_status  Principal  terms effective_date  \
0           1             1     PAIDOFF       1000     30       9/8/2016
1           5             5     PAIDOFF        300      7       9/9/2016
2          21            21     PAIDOFF       1000     30      9/10/2016
3          24            24     PAIDOFF       1000     30      9/10/2016
4          35            35     PAIDOFF        800     15      9/11/2016

    due_date  age             education  Gender
0  10/7/2016   50              Bechalor  female
1  9/15/2016   35      Master or Above    male
2  10/9/2016   43  High School or Below  female
3  10/9/2016   26               college    male
4  9/25/2016   29              Bechalor    male
```

[101]:
```
## Preprocessing
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=True)
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_X = preprocessing.StandardScaler().fit(test_Feature).
↪transform(test_Feature)
test_X[0:5]
```

[101]:
```
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]])
```

[102]:
```
test_y = test_df['loan_status'].values
test_y[0:5]
```

[102]:
```
array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF'], dtype=object)
```

[103]:
```
knn_yhat = kNN_model.predict(test_X)
print("KNN Jaccard index: %.2f" % jaccard_similarity_score(test_y, knn_yhat))
print("KNN F1-score: %.2f" % f1_score(test_y, knn_yhat, average='weighted') )
```

```
KNN Jaccard index: 0.67
KNN F1-score: 0.63
```

[104]:
```
DT_yhat = DT_model.predict(test_X)
print("DT Jaccard index: %.2f" % jaccard_similarity_score(test_y, DT_yhat))
print("DT F1-score: %.2f" % f1_score(test_y, DT_yhat, average='weighted') )
```

```
DT Jaccard index: 0.72
DT F1-score: 0.74
```

[105]:
```
SVM_yhat = SVM_model.predict(test_X)
print("SVM Jaccard index: %.2f" % jaccard_similarity_score(test_y, SVM_yhat))
print("SVM F1-score: %.2f" % f1_score(test_y, SVM_yhat, average='weighted') )
```

```
SVM Jaccard index: 0.80
SVM F1-score: 0.76
```

[106]:
```
LR_yhat = LR_model.predict(test_X)
LR_yhat_prob = LR_model.predict_proba(test_X)
print("LR Jaccard index: %.2f" % jaccard_similarity_score(test_y, LR_yhat))
print("LR F1-score: %.2f" % f1_score(test_y, LR_yhat, average='weighted') )
print("LR LogLoss: %.2f" % log_loss(test_y, LR_yhat_prob))
```

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

12 Report

Report the accuracy of the built model using different evaluation metrics:

Algorithm	Jaccard	F1-score	LogLoss
KNN	0.67	0.63	NA
Decision Tree	0.72	0.74	NA
SVM	0.80	0.76	NA
LogisticRegression	0.74	0.66	0.57

[ ]:
```

```