Classification with Python

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:

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-dat
a/CognitiveClass/ML0101ENv3/labs/loan_train.csv
```

```
--2020-08-01 06:27:34--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-dat
a/CognitiveClass/ML0101ENv3/labs/loan_train.csv
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlaye
r.net)... 67.228.254.196
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.soft
layer.net)|67.228.254.196|: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

2020-08-01 06:27:34 (305 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 [4]:

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

Out[4]:

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

Data visualization and pre-processing

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

In [5]:

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

Out[5]:

```
PAIDOFF       260
COLLECTION     86
```

```
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:

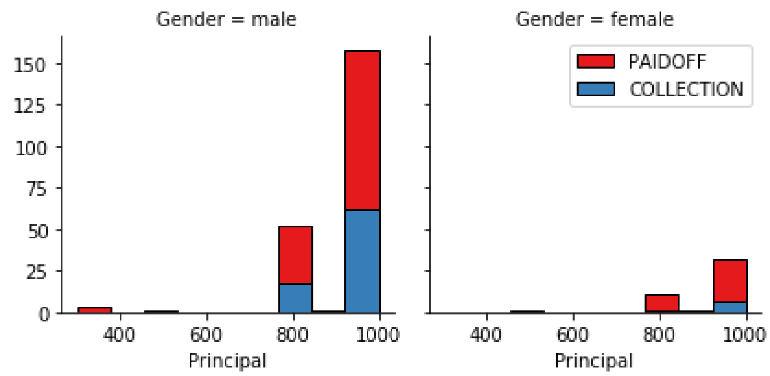
In [7]:

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

```
Solving environment: done

## Package Plan ##

  environment location: /Users/Saeed/anaconda/envs/python3.6

  added / updated specs:
    - seaborn


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    openssl-1.0.2o             |       h26aff7b_0         3.4 MB  anaconda
    ca-certificates-2018.03.07 |                0         124 KB  anaconda
    ------------------------------------------------------------
                                           Total:         3.5 MB

The following packages will be UPDATED:

    ca-certificates: 2018.03.07-0      --> 2018.03.07-0      anaconda
    openssl:         1.0.2o-h26aff7b_0 --> 1.0.2o-h26aff7b_0 anaconda


Downloading and Extracting Packages
openssl-1.0.2o       |  3.4 MB | ####################################### | 100%
ca-certificates-2018 |  124 KB | ####################################### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

In [7]:

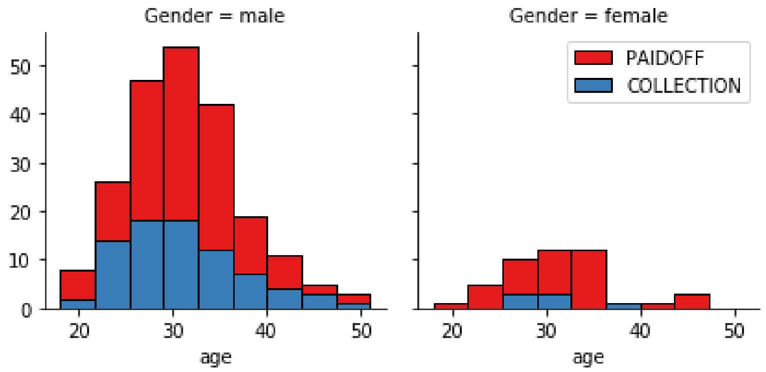
```
import seaborn as sns

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 [8]:

```
bins = np.linspace(df.age.min(), df.age.max(), 10)
```

```
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()
```

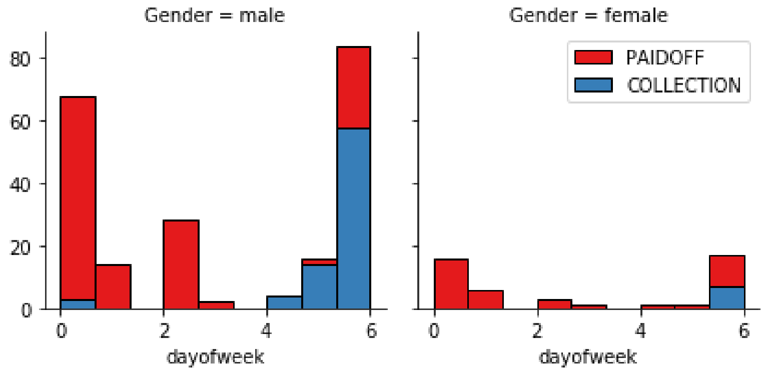


Pre-processing: Feature selection/extraction

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

In [9]:

```
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()
```



In [ ]:

```

```

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 [10]:

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

Out[10]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education	Gender	dayofweek	weeken
0	0	0	PAIDOFF	1000	30	2016-09-08	2016-10-07	45	High School or Below	male	3	
1	2	2	PAIDOFF	1000	30	2016-09-08	2016-10-07	33	Bechalor	female	3	

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education	Gender	dayofweek	weeken
2	3	3	PAIDOFF	1000	15	2016-09-08	2016-09-22	27	college	male	3	
3	4	4	PAIDOFF	1000	30	2016-09-09	2016-10-08	28	college	female	4	
4	6	6	PAIDOFF	1000	30	2016-09-09	2016-10-08	29	college	male	4	

Convert Categorical features to numerical values

Lets look at gender:

In [11]:

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

Out[11]:

```
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 [12]:

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

Out[12]:

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

One Hot Encoding

How about education?

In [13]:

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

Out[13]:

```
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 [14]:

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

Out[14]:

	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 [15]:

```
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[15]:

	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 [16]:

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

Out[16]:

	Principal	terms	age	Gender	weekend	Bechalor	High School or Below	college
0	1000	30	45	0	0	0	1	0

	Principal	terms	age	Gender	weekend	Bechalor	High School or Below	college
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 [17]:

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

Out[17]:

```
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 [18]:

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

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/preprocessing/data.py:645: D
ataConversionWarning: 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: DataConvers
ionWarning: Data with input dtype uint8, int64 were all converted to float64 by StandardS
caler.
  if __name__ == '__main__':
```

Out[18]:

```
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.**

Train Test split

In [19]:

```
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,)
```

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.

Importing libraries

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

Checking for the best value of K

In [74]:

```
for k in range(1, 10):
    knn_model  = KNeighborsClassifier(n_neighbors = k).fit(x_train, y_train)
    knn_yhat = knn_model.predict(x_test)
    print("For K = {} accuracy = {}".format(k,accuracy_score(y_test,knn_yhat)))
```

```
For K = 1 accuracy = 0.6714285714285714
For K = 2 accuracy = 0.6571428571428571
For K = 3 accuracy = 0.7142857142857143
For K = 4 accuracy = 0.6857142857142857
For K = 5 accuracy = 0.7571428571428571
For K = 6 accuracy = 0.7142857142857143
For K = 7 accuracy = 0.7857142857142857
For K = 8 accuracy = 0.7571428571428571
For K = 9 accuracy = 0.7571428571428571
```

In [75]:

```
print("We can see that the KNN model is the best for K=7")
```

```
We can see that the KNN model is the best for K=7
```

Building the model with the best value of K = 7

In [29]:

```
best_knn_model = KNeighborsClassifier(n_neighbors = 7).fit(x_train, y_train)
best_knn_model
```

Out[29]:

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

In [91]:

```
## Evaluation Metrics
# jaccard score and f1 score
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score

print("Train set Accuracy (Jaccard): ", jaccard_similarity_score(y_train, best_knn_model.
predict(x_train)))
print("Test set Accuracy (Jaccard): ", jaccard_similarity_score(y_test, best_knn_model.pr
edict(x_test)))

print("Train set Accuracy (F1): ", f1_score(y_train, best_knn_model.predict(x_train), av
erage='weighted'))
print("Test set Accuracy (F1): ", f1_score(y_test, best_knn_model.predict(x_test), avera
ge='weighted'))
```

```
Train set Accuracy (Jaccard):  0.8079710144927537
Test set Accuracy (Jaccard):  0.7857142857142857
Train set Accuracy (F1):  0.8000194668761034
Test set Accuracy (F1):  0.7766540244416351
```

Decision Tree

In [31]:

```
# importing libraries
from sklearn.tree import DecisionTreeClassifier
```

In [77]:

```
for d in range(1,10):
    dt = DecisionTreeClassifier(criterion = 'entropy', max_depth = d).fit(x_train, y_tra
in)
    dt_yhat = dt.predict(x_test)
    print("For depth = {}  the accuracy score is {} ".format(d, accuracy_score(y_test, d
t_yhat)))
```

```
For depth = 1  the accuracy score is 0.7857142857142857
For depth = 2  the accuracy score is 0.7857142857142857
For depth = 3  the accuracy score is 0.6142857142857143
For depth = 4  the accuracy score is 0.6142857142857143
For depth = 5  the accuracy score is 0.6428571428571429
For depth = 6  the accuracy score is 0.7714285714285715
For depth = 7  the accuracy score is 0.7571428571428571
For depth = 8  the accuracy score is 0.7571428571428571
For depth = 9  the accuracy score is 0.6571428571428571
```

In [79]:

```
print("The best value of depth is d = 2 ")
```

```
The best value of depth is d = 2
```

In [60]:

```
## Creating the best model for decision tree with best value of depth 2

best_dt_model = DecisionTreeClassifier(criterion = 'entropy', max_depth = 2).fit(x_train
, y_train)
best_dt_model
```

Out[60]:

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=2,
            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 [92]:

```
## Evaluation Metrics
# jaccard score and f1 score
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score

print("Train set Accuracy (Jaccard): ", jaccard_similarity_score(y_train, best_dt_model.p
redict(x_train)))
print("Test set Accuracy (Jaccard): ", jaccard_similarity_score(y_test, best_dt_model.pre
dict(x_test)))

print("Train set Accuracy (F1): ", f1_score(y_train, best_dt_model.predict(x_train), ave
rage='weighted'))
print("Test set Accuracy (F1): ", f1_score(y_test, best_dt_model.predict(x_test), averag
e='weighted'))
```

```
Train set Accuracy (Jaccard):  0.7427536231884058
Test set Accuracy (Jaccard):  0.7857142857142857
Train set Accuracy (F1):  0.6331163939859591
Test set Accuracy (F1):  0.6914285714285714
```

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/metrics/classification.py:11
43: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/metrics/classification.py:11
43: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
```

Support Vector Machine

In [63]:

```
#importing svm
from sklearn import svm
from sklearn.metrics import f1_score
```

In [81]:

```
for k in ('linear', 'poly', 'rbf','sigmoid'):
    svm_model = svm.SVC( kernel = k).fit(x_train,y_train)
    svm_yhat = svm_model.predict(x_test)
    print("For kernel: {}, the f1 score is: {}".format(k,f1_score(y_test,svm_yhat, avera
ge='weighted')))
```

```
For kernel: linear, the f1 score is: 0.6914285714285714
For kernel: poly, the f1 score is: 0.7064793130366899
For kernel: rbf, the f1 score is: 0.7275882012724117
For kernel: sigmoid, the f1 score is: 0.6892857142857144
```

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/metrics/classification.py:11
43: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/metrics/classification.py:11
43: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/svm/base.py:196: FutureWarni
ng: The default value of gamma will change from 'auto' to 'scale' in version 0.22 to acco
unt better for unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid this
warning.
  "avoid this warning.", FutureWarning)
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/svm/base.py:196: FutureWarni
ng: The default value of gamma will change from 'auto' to 'scale' in version 0.22 to acco
unt better for unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid this
warning.
```

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

In [68]:

```
print("We can see the rbf has the best f1 score ")
```

```
We can see the rbf has the best f1 score of 0.7275882012724117
```

In [69]:

```
## building best SVM with kernel = rbf
best_svm = svm.SVC(kernel='rbf').fit(x_train,y_train)
best_svm
```

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

Out[69]:

```
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 [93]:

```
## Evaluation Metrics
# jaccard score and f1 score
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score

print("Train set Accuracy (Jaccard): ", jaccard_similarity_score(y_train, best_svm.predic
t(x_train)))
print("Test set Accuracy (Jaccard): ", jaccard_similarity_score(y_test, best_svm.predict(
x_test)))

print("Train set Accuracy (F1): ", f1_score(y_train, best_svm.predict(x_train), average=
'weighted'))
print("Test set Accuracy (F1): ", f1_score(y_test, best_svm.predict(x_test), average='we
ighted'))
```

```
Train set Accuracy (Jaccard):  0.782608695652174
Test set Accuracy (Jaccard):  0.7428571428571429
Train set Accuracy (F1):  0.7682165861513688
Test set Accuracy (F1):  0.7275882012724117
```

Logistic Regression

In [84]:

```
# importing libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss
```

In [85]:

```
for k in ('lbfgs', 'saga', 'liblinear', 'newton-cg', 'sag'):
    lr_model = LogisticRegression(C = 0.01, solver = k).fit(x_train, y_train)
    lr_yhat = lr_model.predict(x_test)
    y_prob = lr_model.predict_proba(x_test)
    print('When Solver is {}, logloss is : {}'.format(k, log_loss(y_test, y_prob)))
```

```
When Solver is lbfgs, logloss is : 0.4920179847937498
When Solver is saga, logloss is : 0.4920191758227281
When Solver is liblinear, logloss is : 0.5772287609479654
When Solver is newton-cg, logloss is : 0.4920178014679269
When Solver is sag, logloss is : 0.4920038148985641
```

In [86]:

```
print("We can see that the best solver is liblinear")
```

```
We can see that the best solver is liblinear
```

In [87]:

```
# Best logistic regression model with liblinear solver

best_lr_model = LogisticRegression(C = 0.01, solver = 'liblinear').fit(x_train, y_train)
best_lr_model
```

Out[87]:

```
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='liblinear',
          tol=0.0001, verbose=0, warm_start=False)
```

In [94]:

```
## Evaluation Metrics
# jaccard score and f1 score
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score

print("Train set Accuracy (Jaccard): ", jaccard_similarity_score(y_train, best_lr_model.p
redict(x_train)))
print("Test set Accuracy (Jaccard): ", jaccard_similarity_score(y_test, best_lr_model.pre
dict(x_test)))

print("Train set Accuracy (F1): ", f1_score(y_train, best_lr_model.predict(x_train), ave
rage='weighted'))
print("Test set Accuracy (F1): ", f1_score(y_test, best_lr_model.predict(x_test), averag
e='weighted'))
```

```
Train set Accuracy (Jaccard):  0.7572463768115942
Test set Accuracy (Jaccard):  0.6857142857142857
Train set Accuracy (F1):  0.7341146337750953
Test set Accuracy (F1):  0.6670522459996144
```

Model Evaluation using Test set

In [20]:

```
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 [95]:

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

```
--2020-08-01 09:35:47--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-dat
a/CognitiveClass/ML0101ENv3/labs/loan_test.csv
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlaye
r.net)... 67.228.254.196
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.soft
layer.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'

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

2020-08-01 09:35:47 (312 MB/s) - 'loan_test.csv' saved [3642/3642]
```

Load Test set for evaluation

In [108]:

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

Out[108]:

	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 [109]:

```
# data processing
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)

Feature1 = test_df[['Principal','terms','age','Gender','weekend']]
Feature1 = pd.concat([Feature1,pd.get_dummies(test_df['education'])], axis=1)
Feature1.drop(['Master or Above'], axis = 1,inplace=True)


x_loan_test = Feature1
x_loan_test = preprocessing.StandardScaler().fit(x_loan_test).transform(x_loan_test)

y_loan_test = test_df['loan_status'].values
```

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/preprocessing/data.py:645: D
ataConversionWarning: 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: DataConver
sionWarning: Data with input dtype uint8, int64 were all converted to float64 by Standard
Scaler.
```

In [110]:

```
# Jaccard

# KNN
knn_yhat = best_knn_model.predict(x_loan_test)
jacc1 = round(jaccard_similarity_score(y_loan_test, knn_yhat), 2)

# Decision Tree
dt_yhat = best_dt_model.predict(x_loan_test)
jacc2 = round(jaccard_similarity_score(y_loan_test, dt_yhat), 2)

# Support Vector Machine
svm_yhat = best_svm.predict(x_loan_test)
```

```
jacc3 = round(jaccard_similarity_score(y_loan_test, svm_yhat), 2)

# Logistic Regression
lr_yhat = best_lr_model.predict(x_loan_test)
jacc4 = round(jaccard_similarity_score(y_loan_test, lr_yhat), 2)

jss = [jacc1, jacc2, jacc3, jacc4]
jss
```

Out[110]:

```
[0.67, 0.74, 0.8, 0.74]
```

In [111]:

```
# F1_score

# KNN
knn_yhat = best_knn_model.predict(x_loan_test)
f1 = round(f1_score(y_loan_test, knn_yhat, average = 'weighted'), 2)

# Decision Tree
dt_yhat = best_dt_model.predict(x_loan_test)
f2 = round(f1_score(y_loan_test, dt_yhat, average = 'weighted'), 2)

# Support Vector Machine
svm_yhat = best_svm.predict(x_loan_test)
f3 = round(f1_score(y_loan_test, svm_yhat, average = 'weighted'), 2)

# Logistic Regression
lr_yhat = best_lr_model.predict(x_loan_test)
f4 = round(f1_score(y_loan_test, lr_yhat, average = 'weighted'), 2)

f1_list = [f1, f2, f3, f4]
f1_list
```

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/metrics/classification.py:11
43: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
```

Out[111]:

```
[0.63, 0.63, 0.76, 0.66]
```

In [113]:

```
# log loss

# Logistic Regression
lr_prob = best_lr_model.predict_proba(x_loan_test)
ll_list = ['NA','NA','NA', round(log_loss(y_loan_test, lr_prob), 2)]
ll_list
```

Out[113]:

```
['NA', 'NA', 'NA', 0.57]
```

In [114]:

```
columns = ['KNN', 'Decision Tree', 'SVM', 'Logistic Regression']
index = ['Jaccard', 'F1-score', 'Logloss']

accuracy_df = pd.DataFrame([jss, f1_list, ll_list], index = index, columns = columns)
accuracy_df1 = accuracy_df.transpose()
accuracy_df1.columns.name = 'Algorithm'
accuracy_df1
```

Out[114]:

Algorithm	Jaccard	F1-score	Logloss
KNN	0.67	0.63	NA

Algorithm	Jaccard	F1-score	Logloss
Decision Tree	0.74	0.63	NA
SVM	0.8	0.76	NA
Logistic Regression	0.74	0.66	0.57

Report

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

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

Want to learn more?

IBM SPSS Modeler is a comprehensive analytics platform that has many machine learning algorithms. It has been designed to bring predictive intelligence to decisions made by individuals, by groups, by systems – by your enterprise as a whole. A free trial is available through this course, available here: SPSS Modeler

Also, you can use Watson Studio to run these notebooks faster with bigger datasets. Watson Studio is IBM's leading cloud solution for data scientists, built by data scientists. With Jupyter notebooks, RStudio, Apache Spark and popular libraries pre-packaged in the cloud, Watson Studio enables data scientists to collaborate on their projects without having to install anything. Join the fast-growing community of Watson Studio users today with a free account at Watson Studio

Thanks for completing this lesson!

Author: Saeed Aghabozorgi

Saeed Aghabozorgi, PhD is a Data Scientist in IBM with a track record of developing enterprise level applications that substantially increases clients' ability to turn data into actionable knowledge. He is a researcher in data mining field and expert in developing advanced analytic methods like machine learning and statistical modelling on large datasets.
