[(https://www.bigdatauniversity.com)](https://www.bigdatauniversity.com)

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 [90]:
```
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 [3]:
```
!wget -O loan_train.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_train.csv
```

```
--2019-10-14 04:37:56--  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 0.002s

2019-10-14 04:37:56 (13.6 MB/s) - 'loan_train.csv' saved [23101/23101]
```

Load Data From CSV File

In [4]:
```
df = pd.read_csv('loan_train.csv')
df.head()
```

Out[4]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education
0	0	0	PAIDOFF	1000	30	9/8/2016	10/7/2016	45	High School or Below
1	2	2	PAIDOFF	1000	30	9/8/2016	10/7/2016	33	Bechalor
2	3	3	PAIDOFF	1000	15	9/8/2016	9/22/2016	27	college
3	4	4	PAIDOFF	1000	30	9/9/2016	10/8/2016	28	college
4	6	6	PAIDOFF	1000	30	9/9/2016	10/8/2016	29	college

In [5]:
```
df.shape
```

Out[5]: (346, 10)

Convert to date time object

In [6]:
```
df['due_date'] = pd.to_datetime(df['due_date'])
df['effective_date'] = pd.to_datetime(df['effective_date'])
df.head()
```

Out[6]:

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

Data visualization and pre-processing

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

In [7]:
```
df['loan_status'].value_counts()
```

Out[7]:
```
PAIDOFF       260
COLLECTION     86
Name: loan_status, dtype: int64
```

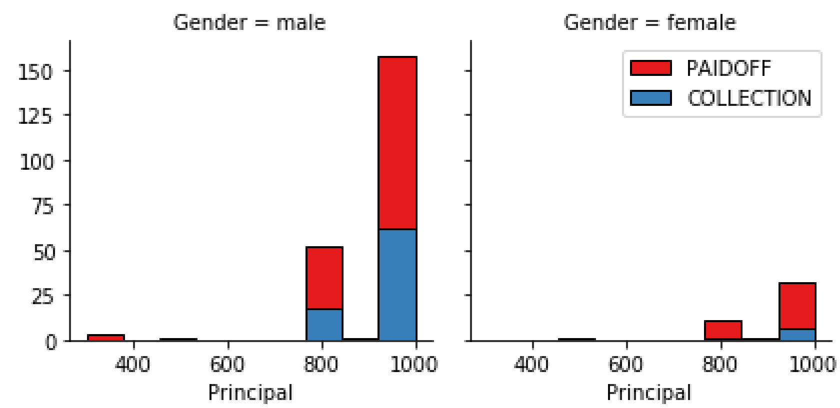
260 people have paid off the loan on time while 86 have gone into collection

Lets plot some columns to underestand data better:

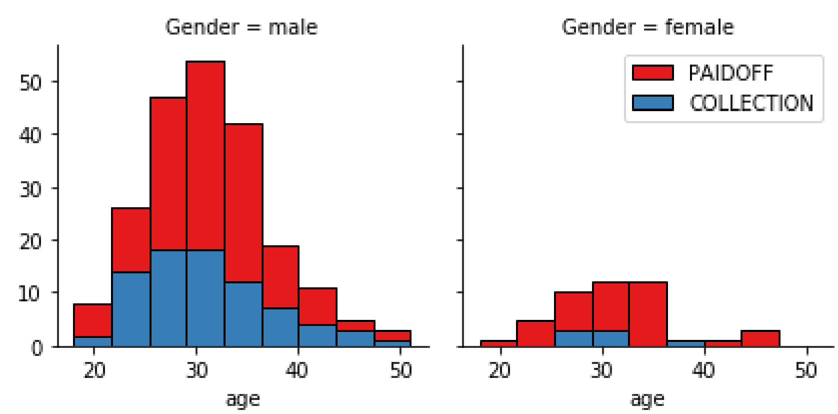
In [7]:
```
# notice: installing seaborn might takes a few minutes
!conda install -c anaconda seaborn -y
```

```
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  anaco
nda
    ca-certificates-2019.8.28  |                0         132 KB  anaco
nda
    certifi-2019.9.11          |           py36_0         154 KB  anaco
nda
    seaborn-0.9.0              |           py36_0         379 KB  anaco
nda
    ------------------------------------------------------------
                                           Total:         5.7 MB

The following packages will be UPDATED:

    ca-certificates: 2019.8.28-0        --> 2019.8.28-0      anaconda
    certifi:         2019.9.11-py36_0   --> 2019.9.11-py36_0 anaconda
    openssl:         1.1.1d-h7b6447c_2 --> 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    | ####################################
# | 100%
ca-certificates-2019 | 132 KB    | ####################################
# | 100%
certifi-2019.9.11    | 154 KB    | ####################################
# | 100%
seaborn-0.9.0        | 379 KB    | ####################################
# | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

In [8]:
```
import seaborn as sns

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

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



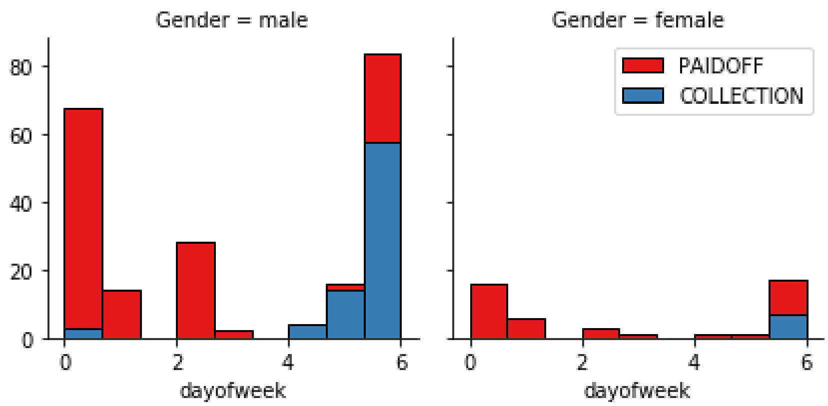
In [9]:
```
bins = np.linspace(df.age.min(), df.age.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", c
ol_wrap=2)
g.map(plt.hist, 'age', bins=bins, ec="k")

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



Pre-processing: Feature selection/extraction

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

In [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", c
ol_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

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

Out[11]:

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

Convert Categorical features to numerical values

Lets look at gender:

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

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

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

Lets convert male to 0 and female to 1:

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

Out[13]:

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

One Hot Encoding

How about education?

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

Out[14]:
```
education             loan_status
Bechalor              PAIDOFF        0.750000
                      COLLECTION     0.250000
High School or Below  PAIDOFF        0.741722
                      COLLECTION     0.258278
Master or Above       COLLECTION     0.500000
                      PAIDOFF        0.500000
college               PAIDOFF        0.765101
                      COLLECTION     0.234899
Name: loan_status, dtype: float64
```

Feature befor One Hot Encoding

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

Out[15]:

	Principal	terms	age	Gender	education
0	1000	30	45	0	High School or Below
1	1000	30	33	1	Bechalor
2	1000	15	27	0	college
3	1000	30	28	1	college
4	1000	30	29	0	college

Use one hot encoding technique to conver categorical varables to binary variables and append them to the feature Data Frame

In [16]:
```
Feature = df[['Principal','terms','age','Gender','weekend']]
Feature = pd.concat([Feature,pd.get_dummies(df['education'])], axis=1)
Feature.drop(['Master or Above'], axis = 1,inplace=True)
Feature.head()
```

Out[16]:

	Principal	terms	age	Gender	weekend	Bechalor	High School or Below	college
0	1000	30	45	0	0	0	1	0
1	1000	30	33	1	0	1	0	0
2	1000	15	27	0	0	0	0	1
3	1000	30	28	1	1	0	0	1
4	1000	30	29	0	1	0	0	1

Feature selection

Lets defind feature sets, X:

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

Out[17]:

	Principal	terms	age	Gender	weekend	Bechalor	High School or Below	college
0	1000	30	45	0	0	0	1	0
1	1000	30	33	1	0	1	0	0
2	1000	15	27	0	0	0	0	1
3	1000	30	28	1	1	0	0	1
4	1000	30	29	0	1	0	0	1

What are our lables?

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

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

Normalize Data

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

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

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/preprocess
ing/data.py:645: DataConversionWarning: Data with input dtype uint8, in
t64 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[19]:
```
array([[ 0.51578458,  0.92071769,  2.33152555, -0.42056004, -1.2057780
5,
        -0.38170062,  1.13639374, -0.86968108],
       [ 0.51578458,  0.92071769,  0.34170148,  2.37778177, -1.2057780
5,
         2.61985426, -0.87997669, -0.86968108],
       [ 0.51578458, -0.95911111, -0.65321055, -0.42056004, -1.2057780
5,
        -0.38170062, -0.87997669,  1.14984679],
       [ 0.51578458,  0.92071769, -0.48739188,  2.37778177,  0.8293400
3,
        -0.38170062, -0.87997669,  1.14984679],
       [ 0.51578458,  0.92071769, -0.3215732 , -0.42056004,  0.8293400
3,
        -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 [21]:
```
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 [27]:
```
k = 3
neigh3 = KNeighborsClassifier(n_neighbors = k).fit(X_train, y_train)
yhat3 = neigh3.predict(X_test)
print("Train set Accuracy:", metrics.accuracy_score(y_train, neigh3.pred
ict(X_train)))
print("Test set Accuracy:", metrics.accuracy_score(y_test, yhat3))
```

```
Train set Accuracy: 0.8333333333333334
Test set Accuracy: 0.7142857142857143
```

In [94]:
```
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 = KNeighborsClassifier(n_neighbors=n).fit(X_train,y_train)
    yhat = KNN.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])
print(mean_acc)
from sklearn.neighbors import KNeighborsClassifier
k = 7
#Train Model and Predict
neigh7 = KNeighborsClassifier(n_neighbors=k).fit(X_train,y_train)
yhat7 = neigh7.predict(X_test)
```

```
[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       ]
```

When k = 7 in KNN classification, I got a better result.

In [95]:
```
from sklearn.metrics import jaccard_similarity_score
print("Jaccard Accuracy of Train set:", jaccard_similarity_score(y_train
, neigh7.predict(X_train)))
print("Jaccard Accuracy of Test set:", jaccard_similarity_score(y_test,
yhat7))
```

```
Jaccard Accuracy of Train set: 0.8079710144927537
Jaccard Accuracy of Test set: 0.7857142857142857
```

In [97]:
```
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score
print("Jaccard Accuracy of Testset by KNN:", jaccard_similarity_score(y_
test, yhat7))
print("F1_Score Accuracy of Test set by KNN:", f1_score(y_test, yhat7, a
verage = 'weighted'))
```

```
Jaccard Accuracy of Testset by KNN: 0.7857142857142857
F1_Score Accuracy of Test set by KNN: 0.7766540244416351
```

Decision Tree

In [29]:
```
from sklearn.model_selection import train_test_split
X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y, te
st_size = 0.3, random_state = 3)
print('Train set:', X_trainset.shape, y_trainset.shape)
print('Test set:', X_testset.shape, y_testset.shape)
```

```
Train set: (242, 8) (242,)
Test set: (104, 8) (104,)
```

In [31]:
```
from sklearn.tree import DecisionTreeClassifier
dTree = DecisionTreeClassifier(criterion = "entropy", max_depth = 4)
dTree.fit(X_trainset, y_trainset)
predTree = dTree.predict(X_testset)
print(predTree[0:5])
print(y_testset[0:5])
```

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

In [32]:
```
print("DecisionTrees's Accuracy:", metrics.accuracy_score(y_testset, pre
dTree))
```

```
DecisionTrees's Accuracy: 0.6538461538461539
```

In [56]:
```
print("Jaccard Accuracy of Testset by DecisionTree:", jaccard_similarity
_score(y_testset, predTree))
print("F1_Score Accuracy of Testset by DecisionTree:",f1_score(y_testset
, predTree, average = 'weighted'))
```

```
Jaccard Accuracy of Testset by DecisionTree: 0.6538461538461539
F1_Score Accuracy of Testset by DecisionTree: 0.6666949930317142
              precision    recall  f1-score   support

  COLLECTION       0.37      0.48      0.42        27
     PAIDOFF       0.80      0.71      0.75        77

   micro avg       0.65      0.65      0.65       104
   macro avg       0.58      0.60      0.59       104
weighted avg       0.69      0.65      0.67       104
```

Support Vector Machine

In [53]:
```
from sklearn import svm
clf = svm.SVC(kernel = 'rbf')
clf.fit(X_train,y_train)
yhat_svm = clf.predict(X_test)
yhat [0:5]
```

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/svm/base.p
y:196: FutureWarning: The default value of gamma will change from 'aut
o' 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[53]:
```
array(['PAIDOFF', 'COLLECTION', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF'],
      dtype=object)
```

In [63]:
```
print("Jaccard Accuracy of Testset by SVM:", jaccard_similarity_score(y_
test, yhat_svm))
print("F1_Score Accuracy of Test set by SVM:", f1_score(y_test, yhat_svm
, average='weighted'))
```

```
Jaccard Accuracy of Testset by SVM: 0.7428571428571429
F1_Score Accuracy of Test set by SVM: 0.7275882012724117
```

Logistic Regression

In [59]:
```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(C = 0.01, solver = 'liblinear').fit(X_train, y_t
rain)
yhat_lr = LR.predict(X_test)
yhat_prob = LR.predict_proba(X_test)
```

```
Jaccard Accuracy of Testset by KNN: 0.6857142857142857
F1_Score Accuracy of Test set by KNN: 0.6670522459996144
```

In [62]:
```
from sklearn.metrics import log_loss
print("Jaccard Accuracy of Testset by LogisticRegression:", jaccard_simi
larity_score(y_test, yhat_lr))
print("F1_Score Accuracy of Test set by LogisticRegression:", f1_score(y
_test, yhat_lr, average='weighted'))
print("Log_Loss Accuracy of Test set by LogisticRegression:", log_loss(y
_test, yhat_prob))
```

```
Jaccard Accuracy of Testset by LogisticRegression: 0.6857142857142857
F1_Score Accuracy of Test set by LogisticRegression: 0.6670522459996144
Log_Loss Accuracy of Test set by LogisticRegression: 0.5772287609479654
```

Model Evaluation using Test set

In [64]:
```
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 [65]:
```
!wget -O loan_test.csv https://s3-api.us-geo.objectstorage.softlayer.ne
t/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_test.csv
```

```
--2019-10-14 05:31:55--  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.obje
ctstorage.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

2019-10-14 05:31:55 (320 MB/s) - ‘loan_test.csv’ saved [3642/3642]
```

Load Test set for evaluation

In [66]:
```
test_df = pd.read_csv('loan_test.csv')
test_df.head()
```

Out[66]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education
0	1	1	PAIDOFF	1000	30	9/8/2016	10/7/2016	50	Bechalor
1	5	5	PAIDOFF	300	7	9/9/2016	9/15/2016	35	Master or Above
2	21	21	PAIDOFF	1000	30	9/10/2016	10/9/2016	43	High School or Below
3	24	24	PAIDOFF	1000	30	9/10/2016	10/9/2016	26	college
4	35	35	PAIDOFF	800	15	9/11/2016	9/25/2016	29	Bechalor

In [81]:
```
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
bins = np.linspace(test_df.dayofweek.min(), test_df.dayofweek.max(), 10)
test_df['weekend'] = test_df['dayofweek'].apply(lambda x: 1 if (x>3)  el
se 0)
test_df.groupby(['Gender'])['loan_status'].value_counts(normalize=True)
#test_df['Gender'].replace(to_replace=['male','female'], value=[0,1],inp
lace=True)
test_df.groupby(['education'])['loan_status'].value_counts(normalize=Tru
e)
Feature = test_df[['Principal','terms','age','Gender','weekend']]
Feature = pd.concat([Feature,pd.get_dummies(test_df['education'])], axis
=1)
Feature.drop(['Master or Above'], axis = 1,inplace=True)
Xt = Feature
yt = test_df['loan_status'].values
Xt = preprocessing.StandardScaler().fit(Xt).transform(Xt)
```

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/preprocess
ing/data.py:645: DataConversionWarning: Data with input dtype uint8, in
t64 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:14: DataConversionWarning: Data with input dtype uint8, int64 were
all converted to float64 by StandardScaler.
```

In [102]:
```
y_knn = neigh7.predict(Xt)
y_dTree = dTree.predict(Xt)
y_svm = clf.predict(Xt)
y_lr = LR.predict(Xt)
y_prob = LR.predict_proba(Xt)
```

In [105]:
```python
jc1 = round(jaccard_similarity_score(yt, y_knn),2)
jc2 = round(jaccard_similarity_score(yt, y_dTree),2)
jc3 = round(jaccard_similarity_score(yt, y_svm),2)
jc4 = round(jaccard_similarity_score(yt, y_lr),2)
f11 = round(f1_score(yt, y_knn, average = 'weighted'),2)
f12 = round(f1_score(yt, y_dTree, average = 'weighted'),2)
f13 = round(f1_score(yt, y_svm, average='weighted'),2)
f14 = round(f1_score(yt, y_lr, average='weighted'),2)
log = round(log_loss(yt, y_prob),2)
print("Jaccard Accuracy of Testset by KNN:", jc1)
print("F1_Score Accuracy of Test set by KNN:", f11)
print("Jaccard Accuracy of Testset by DecisionTree:", jc2)
print("F1_Score Accuracy of Testset by DecisionTree:",f12)
print("Jaccard Accuracy of Testset by SVM:", jc3)
print("F1_Score Accuracy of Test set by SVM:", f13)
print("Jaccard Accuracy of Testset by LogisticRegression:", jc4)
print("F1_Score Accuracy of Test set by LogisticRegression:", f14)
print("Log_Loss Accuracy of Test set by LogisticRegression:", log)
```

```
Jaccard Accuracy of Testset by KNN: 0.67
F1_Score Accuracy of Test set by KNN: 0.63
Jaccard Accuracy of Testset by DecisionTree: 0.78
F1_Score Accuracy of Testset by DecisionTree: 0.78
Jaccard Accuracy of Testset by SVM: 0.8
F1_Score Accuracy of Test set by SVM: 0.76
Jaccard Accuracy of Testset by LogisticRegression: 0.74
F1_Score Accuracy of Test set by LogisticRegression: 0.66
Log_Loss Accuracy of Test set by LogisticRegression: 0.57
```

In [107]:
```python
import pandas as pd

list_jc = [jc1, jc2, jc3, jc4]
list_fs = [f11, f12, f13, f14]
list_ll = ['NA', 'NA', 'NA', log]

df = pd.DataFrame(list_jc, index=['KNN','Decision Tree','SVM','Logistic
 Regression'])
df.columns = ['Jaccard']
df.insert(loc=1, column='F1-score', value=list_fs)
df.insert(loc=2, column='LogLoss', value=list_ll)
df.columns.name = 'Algorithm'
df
```

Out[107]:

Algorithm	Jaccard	F1-score	LogLoss
KNN	0.67	0.63	NA
Decision Tree	0.78	0.78	NA
SVM	0.80	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	0.70	0.70	NA
Decision Tree	0.78	0.78	NA
SVM	0.80	0.76	NA
LogisticRegression	0.74	0.66	0.57

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 (http://cocl.us/ML0101EN-SPSSModeler)

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 (https://cocl.us/ML0101EN_DSX)

Thanks for completing this lesson!

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

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