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

Classification with Python

In this notebook I 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 [6]:

```
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

Load Data From CSV File

In [7]:

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

Out[7]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	ag
0	0	0	PAIDOFF	1000	30	9/8/2016	10/7/2016	45
1	2	2	PAIDOFF	1000	30	9/8/2016	10/7/2016	33
2	3	3	PAIDOFF	1000	15	9/8/2016	9/22/2016	27
3	4	4	PAIDOFF	1000	30	9/9/2016	10/8/2016	28
4	6	6	PAIDOFF	1000	30	9/9/2016	10/8/2016	29

In [24]:

```
df.shape
```

Out[24]: (346, 10)

In [25]:
```
df.dtypes
```

Out[25]:
```
Unnamed: 0                 int64
Unnamed: 0.1               int64
loan_status               object
Principal                  int64
terms                      int64
effective_date    datetime64[ns]
due_date          datetime64[ns]
age                        int64
education                 object
Gender                    object
dtype: object
```

Convert to date time object

In [26]:
```
df['due_date'] = pd.to_datetime(df['due_date'])
df['effective_date'] = pd.to_datetime(df['effective_date'])
df.head()
```

Out[26]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	ag
0	0	0	PAIDOFF	1000	30	2016-09-08	2016-10-07	45
1	2	2	PAIDOFF	1000	30	2016-09-08	2016-10-07	33
2	3	3	PAIDOFF	1000	15	2016-09-08	2016-09-22	27
3	4	4	PAIDOFF	1000	30	2016-09-09	2016-10-08	28
4	6	6	PAIDOFF	1000	30	2016-09-09	2016-10-08	29

Data visualization and pre-processing

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

In [27]:
```
df['loan_status'].value_counts()
```

Out[27]:
```
PAIDOFF       260
COLLECTION     86
Name: loan_status, dtype: int64
```

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

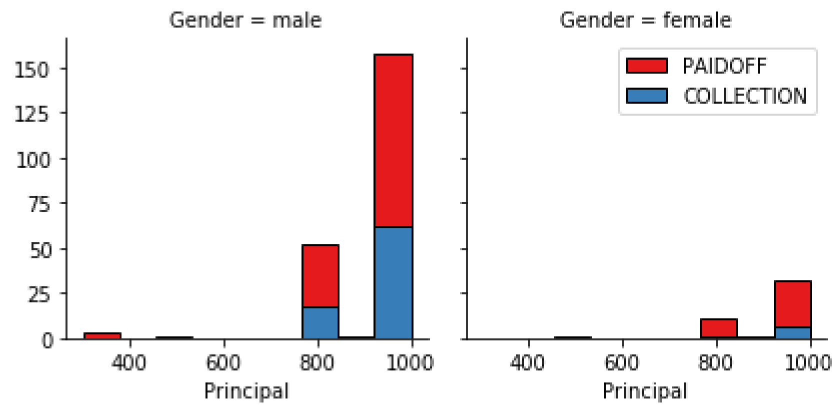
Lets plot some columns to underestand data better:

In [28]:
```
# notice: installing seaborn might takes a few minutes
!conda install -c anaconda seaborn -y
```

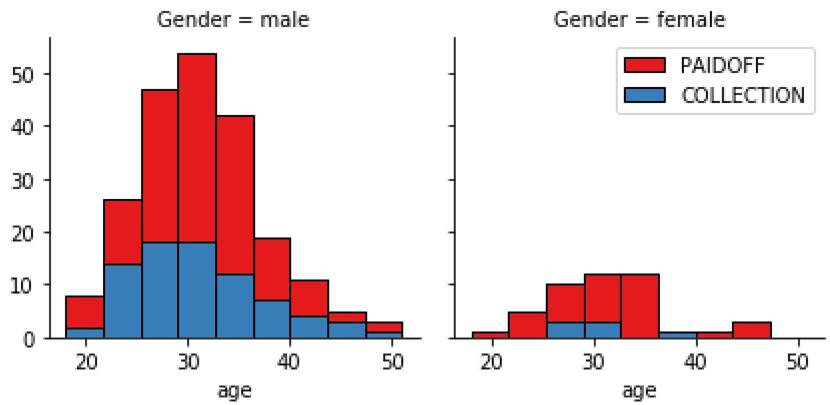
```
Collecting package metadata (repodata.json): ...working... done
Solving environment: ...working... done

# All requested packages already installed.
```

In [29]:

```
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 [31]:
```
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()
```

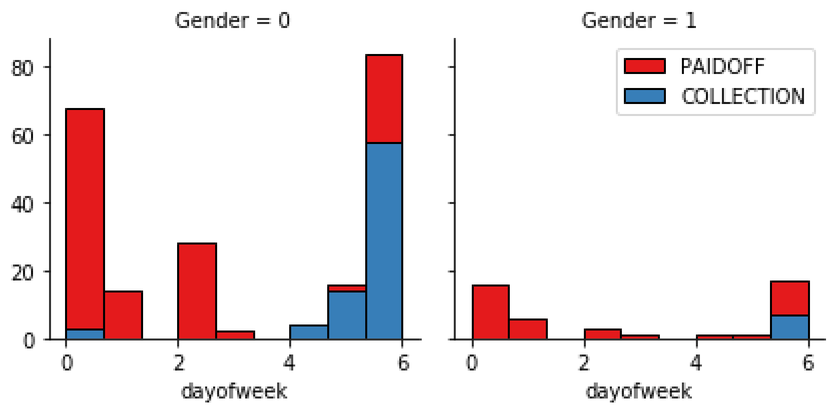


Pre-processing: Feature selection/extraction

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

In [118]:

```
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

In [33]:

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

Out[33]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	ag
0	0	0	PAIDOFF	1000	30	2016-09-08	2016-10-07	45
1	2	2	PAIDOFF	1000	30	2016-09-08	2016-10-07	33
2	3	3	PAIDOFF	1000	15	2016-09-08	2016-09-22	27
3	4	4	PAIDOFF	1000	30	2016-09-09	2016-10-08	28
4	6	6	PAIDOFF	1000	30	2016-09-09	2016-10-08	29

Convert Categorical features to numerical values

Lets look at gender:

In [34]:
```
df.groupby(['Gender'])['loan_status'].value_counts(normalize=True)
```

Out[34]:
```
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 [35]:

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

Out[35]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	ag
0	0	0	PAIDOFF	1000	30	2016-09-08	2016-10-07	45
1	2	2	PAIDOFF	1000	30	2016-09-08	2016-10-07	33
2	3	3	PAIDOFF	1000	15	2016-09-08	2016-09-22	27
3	4	4	PAIDOFF	1000	30	2016-09-09	2016-10-08	28
4	6	6	PAIDOFF	1000	30	2016-09-09	2016-10-08	29

One Hot Encoding

How about education?

In [36]:
```
df.groupby(['education'])['loan_status'].value_counts(normalize=True)
```

Out[36]:
```
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 [37]:
```
df[['Principal','terms','age','Gender','education']].head()
```

Out[37]:

	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 [39]:

```
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[39]:

	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 [40]:
```
X = Feature
X[0:5]
```

Out[40]:

	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 [178]:
```
y = df['loan_status'].values
y[0:5]
```

Out[178]:
```
array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF'],
      dtype=object)
```

In [179]:

```
# Import label encoder
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'Species'.
y= label_encoder.fit_transform(y)
y
```

Out[179]:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

Normalize Data

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

In [180]:

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

Out[180]:

```
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)

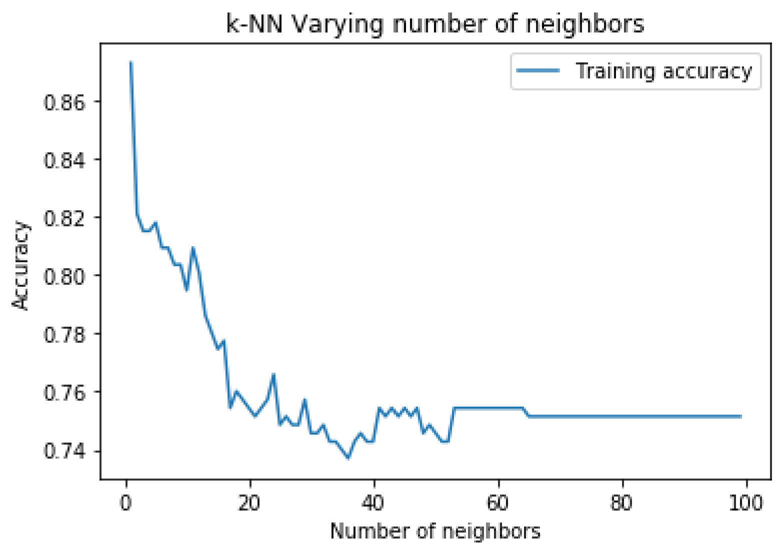
Notice: You should find the best k to build the model with the best accuracy.
warning: You should not use the **loan_test.csv** for finding the best k, however, you can split your train_loan.csv into train and test to find the best **k**.

In [181]:
```
#from sklearn.model_selection import train_test_split
#x_train,x_test,y_train,y_test = train_test_split(X,y,test_size = 0.25,random_state = 0)
```

In [182]:
```
from sklearn.neighbors import KNeighborsClassifier
neighbors = np.arange(1,100)
train_accuracy =np.empty(len(neighbors))
for i,k in enumerate(neighbors):
    #Setup a knn classifier with k neighbors
    knn = KNeighborsClassifier(n_neighbors=k)
    #Fit the model
    knn.fit(X, y)
    #Compute accuracy on the training set
    train_accuracy[i] = knn.score(X, y)
```

In [183]:

```
#KNN_Generate plot
plt.title('k-NN Varying number of neighbors')
plt.plot(neighbors, train_accuracy, label='Training accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
```



In [184]:

```
#Setup a knn classifier with k neighbors
knn = KNeighborsClassifier(n_neighbors=20)
knn.fit(X,y)
```

Out[184]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=20, p=2,
           weights='uniform')
```

Decision Tree

In [185]:

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X,y)
```

Out[185]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            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')
```

Support Vector Machine

In [186]:

```
from sklearn import svm
clf = svm.SVC(kernel='linear')
clf.fit(X, y)
```

Out[186]:

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

Logistic Regression

In [187]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X,y)
```

Out[187]:
```
LogisticRegression(C=1.0, 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)
```

Model Evaluation using Test set

In [188]:
```
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:

Load Test set for evaluation

In [189]:
```
test_df = pd.read_csv('loan_test.csv')
test_df.head()
```

Out[189]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	ag
0	1	1	PAIDOFF	1000	30	9/8/2016	10/7/2016	50
1	5	5	PAIDOFF	300	7	9/9/2016	9/15/2016	35
2	21	21	PAIDOFF	1000	30	9/10/2016	10/9/2016	43
3	24	24	PAIDOFF	1000	30	9/10/2016	10/9/2016	26
4	35	35	PAIDOFF	800	15	9/11/2016	9/25/2016	29

In [190]:
```
#Preprosseing Data:
test_df['due_date'] = pd.to_datetime(test_df['due_date'])
test_df['effective_date'] = pd.to_datetime(test_df['effective_date'])
```

In [191]:

```
test_df['dayofweek'] = test_df['effective_date'].dt.dayofweek
test_df['dayofweek']
```

Out[191]:

```
0     3
1     4
2     5
3     5
4     6
5     6
6     6
7     6
8     6
9     6
10    6
11    6
12    6
13    6
14    6
15    6
16    6
17    6
18    6
19    6
20    6
21    6
22    0
23    0
24    0
25    0
```

```
26    0
27    0
28    0
29    0
30    0
31    0
32    0
33    0
34    0
35    1
36    1
37    1
38    2
39    2
40    4
41    5
42    5
43    5
44    5
45    5
46    5
47    6
48    6
49    6
50    6
51    6
52    6
53    0
Name: dayofweek, dtype: int64
```

In [205]:
```
test_df['weekend'] = test_df['dayofweek'].apply(lambda x: 1 if (x>3)  else 0)
test_df
```

Out[205]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date
0	1	1	PAIDOFF	1000	30	2016-09-08	2016-10-07
1	5	5	PAIDOFF	300	7	2016-09-09	2016-09-15
2	21	21	PAIDOFF	1000	30	2016-09-10	2016-10-09
3	24	24	PAIDOFF	1000	30	2016-09-10	2016-10-09
4	35	35	PAIDOFF	800	15	2016-09-11	2016-09-25
5	37	37	PAIDOFF	700	15	2016-09-11	2016-09-25
6	38	38	PAIDOFF	1000	15	2016-09-11	2016-09-25

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date
7	48	48	PAIDOFF	1000	30	2016-09-11	2016-10-10
8	50	50	PAIDOFF	800	15	2016-09-11	2016-09-25
9	61	61	PAIDOFF	1000	15	2016-09-11	2016-09-25
10	64	64	PAIDOFF	800	15	2016-09-11	2016-09-25
11	68	68	PAIDOFF	300	7	2016-09-11	2016-09-17
12	76	76	PAIDOFF	1000	30	2016-09-11	2016-10-10
13	78	78	PAIDOFF	1000	30	2016-09-11	2016-10-10
14	84	84	PAIDOFF	1000	30	2016-09-11	2016-10-10
15	85	85	PAIDOFF	1000	30	2016-09-11	2016-11-09

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date
16	88	88	PAIDOFF	800	15	2016-09-11	2016-09-25
17	91	91	PAIDOFF	1000	7	2016-09-11	2016-09-17
18	96	96	PAIDOFF	1000	15	2016-09-11	2016-09-25
19	100	100	PAIDOFF	1000	7	2016-09-11	2016-09-17
20	105	105	PAIDOFF	1000	30	2016-09-11	2016-10-10
21	142	142	PAIDOFF	1000	7	2016-09-11	2016-09-17
22	147	147	PAIDOFF	300	7	2016-09-12	2016-09-18
23	150	150	PAIDOFF	1000	15	2016-09-12	2016-10-26

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date
24	156	156	PAIDOFF	1000	15	2016-09-12	2016-09-26
25	167	167	PAIDOFF	800	30	2016-09-12	2016-10-11
26	169	169	PAIDOFF	1000	30	2016-09-12	2016-10-11
27	179	179	PAIDOFF	1000	30	2016-09-12	2016-10-11
28	186	186	PAIDOFF	1000	30	2016-09-12	2016-10-11
29	196	196	PAIDOFF	1000	30	2016-09-12	2016-11-10
30	199	199	PAIDOFF	1000	30	2016-09-12	2016-10-11
31	202	202	PAIDOFF	1000	15	2016-09-12	2016-09-26
32	222	222	PAIDOFF	1000	30	2016-09-12	2016-11-10

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date
33	236	236	PAIDOFF	1000	30	2016-09-12	2016-10-11
34	239	239	PAIDOFF	1000	15	2016-09-12	2016-09-26
35	251	251	PAIDOFF	1000	30	2016-09-13	2016-10-12
36	252	252	PAIDOFF	1000	30	2016-09-13	2016-10-12
37	264	264	PAIDOFF	800	15	2016-09-13	2016-09-27
38	287	287	PAIDOFF	1000	30	2016-09-14	2016-10-13
39	295	295	PAIDOFF	1000	30	2016-09-14	2016-10-13
40	302	302	COLLECTION	1000	30	2016-09-09	2016-10-08

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date
41	305	305	COLLECTION	1000	15	2016-09-10	2016-09-24
42	309	309	COLLECTION	800	15	2016-09-10	2016-09-24
43	310	310	COLLECTION	1000	30	2016-09-10	2016-10-09
44	311	311	COLLECTION	800	15	2016-09-10	2016-09-24
45	313	313	COLLECTION	1000	30	2016-09-10	2016-10-09
46	315	315	COLLECTION	1000	15	2016-09-10	2016-10-09
47	328	328	COLLECTION	1000	30	2016-09-11	2016-10-10

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date
48	331	331	COLLECTION	800	15	2016-09-11	2016-09-25
49	348	348	COLLECTION	1000	30	2016-09-11	2016-10-10
50	349	349	COLLECTION	800	15	2016-09-11	2016-09-25
51	369	369	COLLECTION	1000	30	2016-09-11	2016-10-10
52	370	370	COLLECTION	800	15	2016-09-11	2016-09-25
53	396	396	COLLECTION	1000	30	2016-09-12	2016-10-11

In [206]:

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

In [208]:

```
x_test=Feature_test
```

In [209]:

```
x_test=preprocessing.StandardScaler().fit(x_test).transform(x_test)
x_test[1:5]
```

Out[209]:

```
array([[-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 [210]:

```
y_test=test_df['loan_status'].values
# Import label encoder
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'Species'.
y_test= label_encoder.fit_transform(y_test)
y_test
```

Out[210]:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

In [212]:

```
#prediction
y_pred_1 = knn.predict(x_test)
y_pred_2 = dt.predict(x_test)
y_pred_3 = clf.predict(x_test)
y_pred_4 = lr.predict(x_test)
```

KNN Evaluation:

In [213]:

```
print(jaccard_similarity_score(y_test,y_pred_1))
```

```
0.7407407407407407
```

In [214]:
```
print(f1_score(y_test,y_pred_1,average=None))
```

```
[0.3        0.84090909]
```

Decision Tree Evaluation:

In [215]:
```
print(jaccard_similarity_score(y_test,y_pred_2))
```

```
0.6851851851851852
```

In [216]:
```
print(f1_score(y_test,y_pred_2,average=None))
```

```
[0.4137931  0.78481013]
```

SVM Evaluation:

In [217]:
```
print(jaccard_similarity_score(y_test,y_pred_3))
```

```
0.7407407407407407
```

In [218]:

```
print(f1_score(y_test,y_pred_3,average=None))
```

```
[0.         0.85106383]
```

```
C:\Users\Majid\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

LogisticRegression Evalaution:

In [219]:

```
print(jaccard_similarity_score(y_test,y_pred_4))
```

```
0.7592592592592593
```

In [220]:

```
print(f1_score(y_test,y_pred_4,average=None))
```

```
[0.13333333 0.86021505]
```

In [221]:

```
# Import label encoder
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'Species'.
y_test= label_encoder.fit_transform(y_test)
```

In [222]:

```
print(log_loss(y_test,y_pred_4))
```

```
8.315083109267249
```

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 (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.
