Machine learning: toutes les classsifictions etudiees

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:

[2]:
```
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
```


```
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

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

```
--2020-03-29 11:03:14--  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)... 67.228.254.196
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.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 0.002s

2020-03-29 11:03:14 (13.3 MB/s) - 'loan_train.csv' saved [23101/23101]
```


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

Load Data From CSV File

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

[4]:

	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

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

Convert to date time object

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

[6]:

	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

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

Data visualization and pre-processing

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

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

[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

df['loan_status'].value_counts()

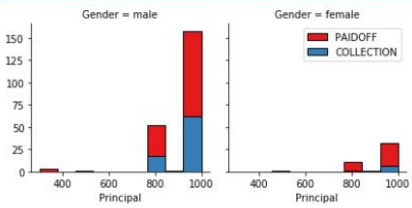
Lets plot some columns to underestand data better:

```
[8]: # 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
    ---------------------------|-----------------
    ca-certificates-2020.1.1   |                0         132 KB  anaconda
    certifi-2019.11.28         |           py36_1         157 KB  anaconda
    openssl-1.1.1              |       h7b6447c_0         5.0 MB  anaconda
    seaborn-0.10.0             |             py_0         161 KB  anaconda
    ------------------------------------------------------------
```

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

```
The following packages will be UPDATED:

    ca-certificates: 2020.1.1-0         --> 2020.1.1-0         anaconda
    certifi:         2019.11.28-py36_0  --> 2019.11.28-py36_1  anaconda
    openssl:         1.1.1e-h7b6447c_0  --> 1.1.1-h7b6447c_0   anaconda
    seaborn:         0.9.0-pyh91ea838_1 --> 0.10.0-py_0        anaconda


Downloading and Extracting Packages
ca-certificates-2020 | 132 KB    | ##################################### | 100%
certifi-2019.11.28   | 157 KB    | ##################################### | 100%
openssl-1.1.1        | 5.0 MB    | ##################################### | 100%
seaborn-0.10.0       | 161 KB    | ##################################### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

```
[9]: 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()
```

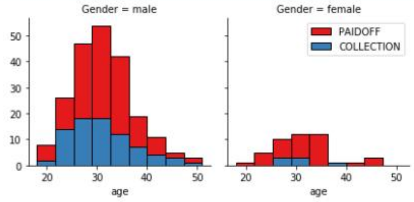


import seaborn as sns

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()

```
[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()
```



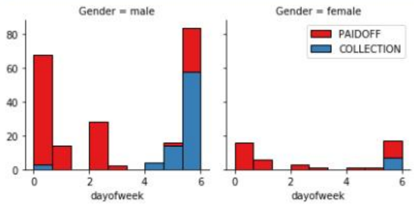
bins = np.linspace(df.age.min(), df.age.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wrap=2)
g.map(plt.hist, 'age', bins=bins, ec="k")

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

Pre-processing: Feature selection/extraction

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

```
[11]: 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()
```



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

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

[12]:

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

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

Convert Categorical features to numerical values

Lets look at gender:

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

```
[13]: 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

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

Lets convert male to 0 and female to 1:

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

[14]:

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

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

One Hot Encoding

How about education? ¶

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

```
[15]: 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
```

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

Feature befor One Hot Encoding

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

[16]:

	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

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

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

```
[17]: 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()
```

[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

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()

Feature selection

Lets defind feature sets, X:

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

[18]:

	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

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

What are our lables?

```
[20]: y ['loan_status'].values
y[0:5]
```

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

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

Normalize Data

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

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

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/preprocessing/data.py:645: DataConversionWarning: Data with input dtype uint8, int64 were all converted to flo
at64 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 St
andardScaler.
  if __name__ == '__main__':
```

```
[21]: 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]])
```

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

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**.

```
[22]: 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)
```

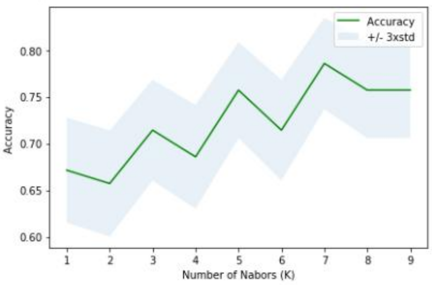
```
Train set: (276, 8) (276,)
Test set: (70, 8) (70,)
```

```
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)
```

```
[24]: from sklearn.neighbors import KNeighborsClassifier
      from sklearn import metrics
      Ks = 10
      mean_acc = np.zeros((Ks-1))
      std_acc = np.zeros((Ks-1))
      ConfustionMx = [];
      for n in range(1,Ks):

          #Train Model and Predict
          neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
          yhat=neigh.predict(X_test)
          mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)


          std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

      mean_acc

[24]: array([0.67142857, 0.65714286, 0.71428571, 0.68571429, 0.75714286,
```


```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfustionMx = [];
for n in range(1,Ks):

    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)


    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

mean_acc
```


```
[25]: #Plot model accuracy for different number of neighbors
      plt.plot(range(1,Ks),mean_acc,'g')
      plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
      plt.legend(('Accuracy ', '+/- 3xstd'))
      plt.ylabel('Accuracy ')
      plt.xlabel('Number of Nabors (K)')
      plt.tight_layout()
      plt.show()

      print( "The best accuracy was with", mean_acc.max(), "with k=", mean_acc.argmax()+1)
```




```
The best accuracy was with 0.7857142857142857 with k= 7
```


```
#Plot model accuracy for different number of neighbors
plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Nabors (K)')
plt.tight_layout()
plt.show()

print( "The best accuracy was with", mean_acc.max(), "with k=", mean_acc.argmax()+1)
```

Decision Tree

```
[27]: import numpy as np
      import pandas as pd
      from sklearn.tree import DecisionTreeClassifier
      from sklearn.model_selection import train_test_split
      X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y, test_size=0.3, random_state=3)
```

```
[28]: # Modeling
      loan_statu_Tree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
      loan_statu_Tree # it shows the default parameters
      loan_statu_Tree.fit(X_trainset,y_trainset)
```

```
[28]: 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')
```

27

```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y, test_size=0.3, random_state=3)
```

28

```
# Modeling
loan_statu_Tree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
loan_statu_Tree # it shows the default parameters
loan_statu_Tree.fit(X_trainset,y_trainset)
```

```
[29]: # Prediction
      predTree = loan_statu_Tree.predict(X_testset)
      print (predTree [0:5])
      print (y_testset [0:5])

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

```
# Prediction
predTree = loan_statu_Tree.predict(X_testset)
print (predTree [0:5])
print (y_testset [0:5])
```

```
[30]: # Evaluation
      from sklearn import metrics
      import matplotlib.pyplot as plt
      print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_testset, predTree))

      DecisionTrees's Accuracy:  0.6538461538461539
```

```
# Evaluation
from sklearn import metrics
import matplotlib.pyplot as plt
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_testset, predTree))
```

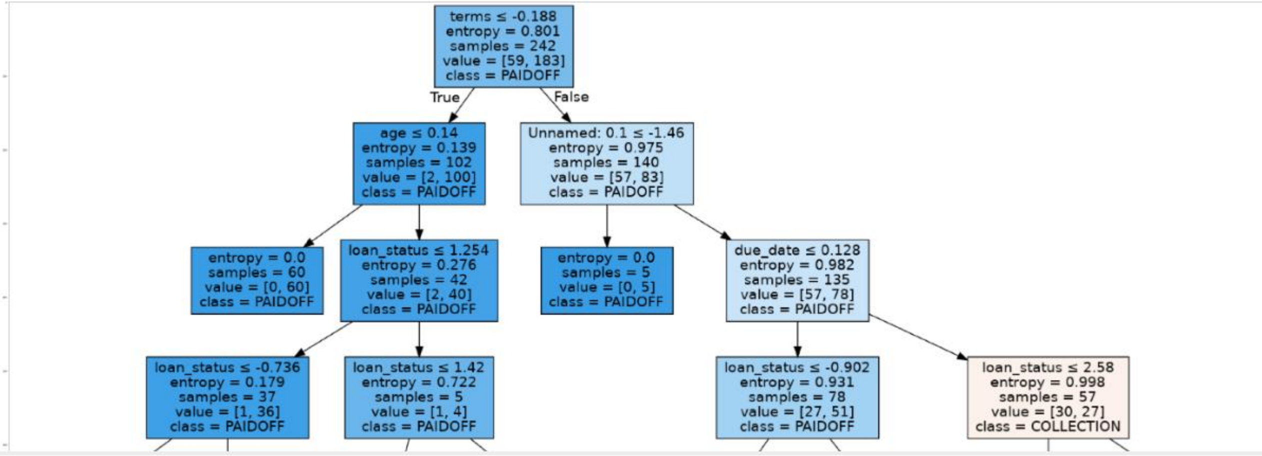
```
[35]: # Visualization
      !conda install -c conda-forge pydotplus -y
      !conda install -c conda-forge python-graphviz -y

      from sklearn.externals.six import StringIO
      import pydotplus
      import matplotlib.image as mpimg
      from sklearn import tree
      %matplotlib inline

      Solving environment: done
```

```
# Visualization
!conda install -c conda-forge pydotplus -y
!conda install -c conda-forge python-graphviz -y

from sklearn.externals.six import StringIO
import pydotplus
import matplotlib.image as mpimg
from sklearn import tree
```

%matplotlib inline

```
# All requested packages already installed.

Solving environment: done

# All requested packages already installed.
```

```
4]: dot_data = StringIO()
filename = "loan_statu_Tree.png"
featureNames = df.columns[0:8]
targetNames =df['loan_status'].unique().tolist()
out=tree.export_graphviz(loan_statu_Tree,feature_names=featureNames, out_file=dot_data, class_names= np.unique(y_trainset), filled=True,  special_characters=True,rotate=False)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png(filename)
img = mpimg.imread(filename)
plt.figure(figsize=(100, 200))
plt.imshow(img,interpolation='nearest')
```



dot_data = StringIO()
filename = "loan_statu_Tree.png"
featureNames = df.columns[0:8]
targetNames =df['loan_status'].unique().tolist()
out=tree.export_graphviz(loan_statu_Tree,feature_names=featureNames, out_file=dot_data, class_names= np.unique(y_trainset), filled=True, special_characters=True,rotate=False)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png(filename)
img = mpimg.imread(filename)
plt.figure(figsize=(100, 200))
plt.imshow(img,interpolation='nearest')

Support Vector Machine

```
[37]: import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
%matplotlib inline
import matplotlib.pyplot as plt

# Change df['loan_status'] values to 0 or 1 int
df['loan_status'].replace(to_replace=['PAIDOFF','COLLECTION'], value=[1,0],inplace=True)
df.head()
```

[37]:

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

import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
%matplotlib inline

import matplotlib.pyplot as plt

Change df['loan_status'] values to 0 or 1 int
df['loan_status'].replace(to_replace=['PAIDOFF','COLLECTION'], value=[1,0],inplace=True)
df.head()

[40]:

```
# Concatenate feature with new values of df['loan_status']
New_Feature = pd.concat([Feature,df['loan_status']], axis=1)


New_Feature.head()
```

[40]:

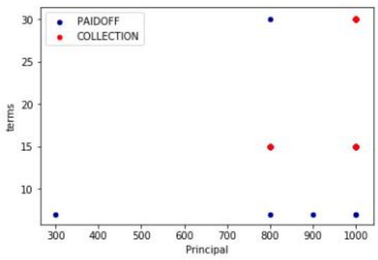
	Principal	terms	age	Gender	weekend	Bechalor	High School or Below	college	loan_status
0	1000	30	45	0	0	0	1	0	1
1	1000	30	33	1	0	1	0	0	1
2	1000	15	27	0	0	0	0	1	1
3	1000	30	28	1	1	0	0	1	1
4	1000	30	29	0	1	0	0	1	1

# Concatenate feature with new values of df['loan_status']
New_Feature = pd.concat([Feature,df['loan_status']], axis=1)

New_Feature.head()

[42]:

```
# Let's look at the dstribution
ax = New_Feature[New_Feature['loan_status'] == 1][0:50].plot(kind='scatter', x='Principal', y='terms', color='DarkBlue', label='PAIDOFF');
New_Feature[New_Feature['loan_status'] == 0][0:50].plot(kind='scatter', x='Principal', y='terms', color='Red', label='COLLECTION', ax=ax);
plt.show()
```



# Let's look at the dstribution
ax = New_Feature[New_Feature['loan_status'] == 1][0:50].plot(kind='scatter', x='Principal', y='terms', color='DarkBlue', label='PAIDOFF');
New_Feature[New_Feature['loan_status'] == 0][0:50].plot(kind='scatter', x='Principal', y='terms', color='Red', label='COLLECTION', ax=ax);
plt.show()

[43]:

```
New_Feature.dtypes
```

[43]:

```
Principal               int64
terms                   int64
age                     int64
Gender                  int64
weekend                 int64
Bechalor                uint8
High School or Below    uint8
college                 uint8
loan_status             int64
dtype: object
```

New_Feature.dtypes

```
[44]: # Drop the Non numerical rows in Bechalor, College and High school columns
New_Feature = New_Feature[pd.to_numeric(New_Feature['Bechalor'], errors='coerce').notnull()]
New_Feature['Bechalor'] = New_Feature['Bechalor'].astype('int')
New_Feature = New_Feature[pd.to_numeric(New_Feature['High School or Below'], errors='coerce').notnull()]
New_Feature['High School or Below'] = New_Feature['High School or Below'].astype('int')
New_Feature = New_Feature[pd.to_numeric(New_Feature['college'], errors='coerce').notnull()]
New_Feature['college'] = New_Feature['college'].astype('int')

New_Feature.dtypes

[44]: Principal              int64
terms                  int64
age                    int64
Gender                 int64
weekend                int64
Bechalor               int64
High School or Below   int64
college                int64
loan_status            int64
dtype: object
```

# Drop the Non numerical rows in Bechalor, College and High school columns
New_Feature = New_Feature[pd.to_numeric(New_Feature['Bechalor'], errors='coerce').notnull()]
New_Feature['Bechalor'] = New_Feature['Bechalor'].astype('int')
New_Feature = New_Feature[pd.to_numeric(New_Feature['High School or Below'], errors='coerce').notnull()]
New_Feature['High School or Below'] = New_Feature['High School or Below'].astype('int')
New_Feature = New_Feature[pd.to_numeric(New_Feature['college'], errors='coerce').notnull()]
New_Feature['college'] = New_Feature['college'].astype('int')

New_Feature.dtypes

```
[45]: # X Data selection
feature2_df = New_Feature[['Principal', 'terms', 'age', 'Gender', 'weekend', 'Bechalor', 'High School or Below', 'college']]
X1 = np.asarray(feature2_df)
X1[0:5]

[45]: array([[1000,   30,   45,    0,    0,    0,    1,    0],
       [1000,   30,   33,    1,    0,    1,    0,    0],
       [1000,   15,   27,    0,    0,    0,    0,    1],
       [1000,   30,   28,    1,    1,    0,    0,    1],
       [1000,   30,   29,    0,    1,    0,    0,    1]])
```

# X Data selection
feature2_df = New_Feature[['Principal', 'terms', 'age', 'Gender', 'weekend', 'Bechalor', 'High School or Below', 'college']]
X1 = np.asarray(feature2_df)
X1[0:5]

```
[50]: # y Data selection
New_Feature['loan_status'] = New_Feature['loan_status'].astype('int')
y1 = np.asarray(New_Feature['loan_status'])
y1 [0:5]

[50]: array([1, 1, 1, 1, 1])
```

y Data selection
New_Feature['loan_status'] = New_Feature['loan_status'].astype('int')
y1 = np.asarray(New_Feature['loan_status'])
y1 [0:5]

```
[51]: # Train and test data
X1_train, X1_test, y1_train, y1_test = train_test_split( X1, y1, test_size=0.2, random_state=4)
print ('Train set:', X1_train.shape,  y1_train.shape)
print ('Test set:', X1_test.shape,  y1_test.shape)

Train set: (276, 8) (276,)
Test set: (70, 8) (70,)
```

Train and test data
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.2, random_state=4)
print ('Train set:', X1_train.shape, y1_train.shape)
print ('Test set:', X1_test.shape, y1_test.shape)

```
[52]: # Modeling
      from sklearn import svm
      clf = svm.SVC(kernel='rbf')
      clf.fit(X1_train, y1_train)
      yhat = clf.predict(X1_test)
      yhat [0:5]
```

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

```
# Modeling
from sklearn import svm
clf = svm.SVC(kernel='rbf')
clf.fit(X1_train, y1_train)
yhat = clf.predict(X1_test)
yhat [0:5]
```

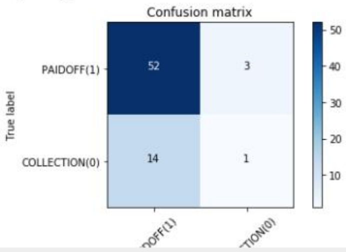
```
[53]: # Evaluation
      from sklearn.metrics import classification_report, confusion_matrix
      import itertools

      def plot_confusion_matrix(cm, classes,
                                normalize=False,
                                title='Confusion matrix',
                                cmap=plt.cm.Blues):
          """
          This function prints and plots the confusion matrix.
          Normalization can be applied by setting `normalize=True`.
          """
          if normalize:
              cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
              print("Normalized confusion matrix")
          else:
              print('Confusion matrix, without normalization')

          print(cm)

          plt.imshow(cm, interpolation='nearest', cmap=cmap)
          plt.title(title)
          plt.colorbar()
          plt.xticks(tick_marks, classes, rotation=45)
          plt.yticks(tick_marks, classes)

          fmt = '.2f' if normalize else 'd'
          thresh = cm.max() / 2.
          for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
              plt.text(j, i, format(cm[i, j], fmt),
                       horizontalalignment="center",
                       color="white" if cm[i, j] > thresh else "black")

          plt.tight_layout()
          plt.ylabel('True label')
          plt.xlabel('Predicted label')
```

```
# Evaluation
from sklearn.metrics import classification_report, confusion_matrix
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
```

```
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```
[55]: # Compute the confusion matrix
      cnf_matrix = confusion_matrix(y1_test, yhat, labels=[1,0])
      np.set_printoptions(precision=2)

      print (classification_report(y1_test, yhat))

      # Plot non-normalized confusion matrix
      plt.figure()
      plot_confusion_matrix(cnf_matrix, classes=['PAIDOFF(1)','COLLECTION(0)'],normalize= False,  title='Confusion matrix')
```

```
              precision    recall  f1-score   support

           0       0.25      0.07      0.11        15
           1       0.79      0.95      0.86        55

   micro avg       0.76      0.76      0.76        70
   macro avg       0.52      0.51      0.48        70
weighted avg       0.67      0.76      0.70        70

Confusion matrix, without normalization
[[52  3]
 [14  1]]
```




```
# Compute the confusion matrix
cnf_matrix = confusion_matrix(y1_test, yhat, labels=[1,0])
np.set_printoptions(precision=2)

print (classification_report(y1_test, yhat))

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['PAIDOFF(1)','COLLECTION(0)'],normalize= False,  title='Confusion matrix')
```

Logistic Regression

```
[57]: # Normalize the dataset
      from sklearn import preprocessing
      X1_norm = preprocessing.StandardScaler().fit(X1).transform(X1)
      X1_norm[0:5]
```

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with input dtype int64 was converted to float64 by StandardS
caler.
  warnings.warn(msg, DataConversionWarning)
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with input dtype int64 was converted to float64 by StandardS
caler.
  warnings.warn(msg, DataConversionWarning)
```

```
# Normalize the dataset
from sklearn import preprocessing
X1_norm = preprocessing.StandardScaler().fit(X1).transform(X1)
X1_norm[0:5]
```

```
[58]: # Train and test dataset
from sklearn.model_selection import train_test_split
X1_train, X1_test, y1_train, y1_test = train_test_split( X1, y1, test_size=0.2, random_state=4)
print ('Train set:', X1_train.shape,  y1_train.shape)
print ('Test set:', X1_test.shape,  y1_test.shape)

Train set: (276, 8) (276,)
Test set: (70, 8) (70,)
```

# Train and test dataset
from sklearn.model_selection import train_test_split
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.2, random_state=4)
print ('Train set:', X1_train.shape, y1_train.shape)
print ('Test set:', X1_test.shape, y1_test.shape)

```
[59]: # Modeling logistic regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X1_train,y1_train)
LR

[59]: 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)
```

# Modeling logistic regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X1_train,y1_train)
LR

```
[60]: # Predicting using our test set
y1hat = LR.predict(X1_test)
y1hat

[60]: 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])
```

# Predicting using our test set
y1hat = LR.predict(X1_test)
y1hat

```
[61]: # Predict proba estimate all classes ordered by label
      y1hat_prob = LR.predict_proba(X1_test)
      y1hat_prob
```

```
[61]: array([[0.32, 0.68],
       [0.25, 0.75],
       [0.18, 0.82],
       [0.32, 0.68],
       [0.24, 0.76],
       [0.21, 0.79],
       [0.27, 0.73],
       [0.24, 0.76],
       [0.32, 0.68],
       [0.3 , 0.7 ],
       [0.3 , 0.7 ],
       [0.32, 0.68],
       [0.29, 0.71],
       [0.3 , 0.7 ],
       [0.14, 0.86],
       [0.16, 0.84],
       [0.4 , 0.6 ],
       [0.18, 0.82],
       [0.34, 0.66],
       [0.22, 0.78],
       [0.28, 0.72],
       [0.31, 0.69],
       [0.36, 0.64],
       [0.37, 0.63],
       [0.21, 0.79],
       [0.34, 0.66],
       [0.35, 0.65],
       [0.15, 0.85],
       [0.35, 0.65],
       [0.14, 0.86],
       [0.22, 0.78],
       [0.34, 0.66],
       [0.28, 0.72],
       [0.28, 0.72],
       [0.21, 0.79],
       [0.2 , 0.8 ],
       [0.34, 0.66],
       [0.14, 0.86],
       [0.31, 0.69],
       [0.21, 0.79],
       [0.35, 0.65],
       [0.23, 0.77],
       [0.17, 0.83],
       [0.34, 0.66],
       [0.19, 0.81],
       [0.34, 0.66],
       [0.23, 0.77],
       [0.32, 0.68],
       [0.34, 0.66],
       [0.19, 0.81],
       [0.34, 0.66],
       [0.23, 0.77],
       [0.32, 0.68],
       [0.19, 0.81],
       [0.19, 0.81],
       [0.24, 0.76],
       [0.2 , 0.8 ],
       [0.24, 0.76],
       [0.29, 0.71],
       [0.26, 0.74],
       [0.1 , 0.9 ],
       [0.21, 0.79],
       [0.25, 0.75],
       [0.22, 0.78],
       [0.27, 0.73],
       [0.12, 0.88],
       [0.28, 0.72],
       [0.2 , 0.8 ],
       [0.41, 0.59],
       [0.24, 0.76],
       [0.31, 0.69],
       [0.28, 0.72],
       [0.2 , 0.8 ],
       [0.16, 0.84],
       [0.24, 0.76]])
```


# Predict proba estimate all classes ordered by label
y1hat_prob = LR.predict_proba(X1_test)
y1hat_prob

Model Evaluation using Test set

```
[62]: 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:

from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss

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

```
--2020-03-29 16:33:58--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_test.csv
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.net)... 67.228.254.196
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.objectstorage.softlayer.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

2020-03-29 16:33:58 (335 MB/s) - 'loan_test.csv' saved [3642/3642]
```

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

Load Test set for evaluation

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

[64]:

	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

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

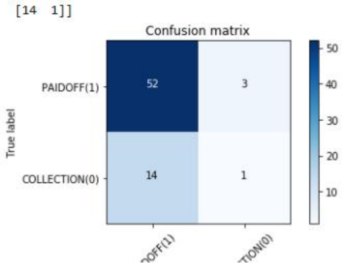
```
[65]: # Evaluate the model using jaccard
from sklearn.metrics import jaccard_similarity_score
jaccard_similarity_score(y1_test, y1hat)
```

```
[65]: 0.7857142857142857
```

# Evaluate the model using jaccard
from sklearn.metrics import jaccard_similarity_score
jaccard_similarity_score(y1_test, y1hat)

```
[66]: # Evaluating the confusing matrix
from sklearn.metrics import classification_report, confusion_matrix
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

    cnf_matrix = confusion_matrix(y1_test, yhat, labels=[1,0])
np.set_printoptions(precision=2)

print (classification_report(y1_test, yhat))

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['PAIDOFF(1)','COLLECTION(0)'],normalize= False,  title='Confusion matrix')
```

```
              precision    recall  f1-score   support

           0       0.25      0.07      0.11        15
           1       0.79      0.95      0.86        55

   micro avg       0.76      0.76      0.76        70
   macro avg       0.52      0.51      0.48        70
weighted avg       0.67      0.76      0.70        70

Confusion matrix, without normalization
[[52  3]
 [14  1]]
```



# Evaluating the confusing matrix
from sklearn.metrics import classification_report, confusion_matrix
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)

```
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

    cnf_matrix = confusion_matrix(y1_test, yhat, labels=[1,0])
np.set_printoptions(precision=2)

print (classification_report(y1_test, yhat))

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['PAIDOFF(1)','COLLECTION(0)'],normalize= False,  title='Confusion matrix')
```

```
[69]: # Evaluating the model using  log_loss
      log_loss(y1_test, y1hat_prob)

[69]: 0.5435595463662589
```

```
# Evaluating the model using  log_loss
log_loss(y1_test, y1hat_prob)
```

```
[71]: # 1. Using jaccard to evaluate KNN
      from sklearn.metrics import jaccard_similarity_score
      jaccard_similarity_score(y_test, yhat)

[71]: 0.7571428571428571
```

```
# 1. Using jaccard to evaluate KNN
from sklearn.metrics import jaccard_similarity_score
jaccard_similarity_score(y_test, yhat)
```

```
[75]: # Using jaccard to evaluate Decision tree
      jaccard_similarity_score(y_testset, predTree)

[75]: 0.6538461538461539
```

```
# Using jaccard to evaluate Decision tree
jaccard_similarity_score(y_testset, predTree)
```

```
[76]: # Using jaccard to evaluate SVM
      jaccard_similarity_score(y1_test, y1hat)

[76]: 0.7857142857142857
```

```
# Using jaccard to evaluate SVM
jaccard_similarity_score(y1_test, y1hat)
```

```
[77]: # Using jaccard to evaluate Logistic regression
      jaccard_similarity_score(y1_test, y1hat)

[77]: 0.7857142857142857
```

```
# Using jaccard to evaluate Logistic regression
jaccard_similarity_score(y1_test, y1hat)
```

```
[81]: # Using jaccard to evaluate different models
      print('jaccard to accurate KNN =',jaccard_similarity_score(y_test, yhat))
      print('jaccard to accurate Decision Tree =',jaccard_similarity_score(y_testset, predTree))
      print('jaccard to accurate SVM =',jaccard_similarity_score(y1_test, y1hat))
      print('jaccard to accurate Logistic Regression =',jaccard_similarity_score(y1_test, y1hat))

      jaccard to accurate KNN = 0.7571428571428571
      jaccard to accurate Decision Tree = 0.6538461538461539
      jaccard to accurate SVM = 0.7857142857142857
      jaccard to accurate Logistic Regression = 0.7857142857142857
```

```
# Using jaccard to evaluate different models
print('jaccard to accurate KNN =',jaccard_similarity_score(y_test, yhat))
print('jaccard to accurate Decision Tree =',jaccard_similarity_score(y_testset, predTree))
print('jaccard to accurate SVM =',jaccard_similarity_score(y1_test, y1hat))
print('jaccard to accurate Logistic Regression =',jaccard_similarity_score(y1_test, y1hat))
```

```
[88]: # Using f1 score to evaluate different models
print('F1 score to accurate KNN')
print (classification_report(y_test, yhat))
print('F1 score to accurate Decision Tree')
print (classification_report(y_testset, predTree))
print('F1 score to accurate SVM')
print (classification_report(y1_test, y1hat))
print('F1 score to accurate Logistic Regression')
print (classification_report(y1_test, y1hat))
```

```
F1 score to accurate KNN
              precision    recall  f1-score   support

           0       0.25      0.07      0.11        15
           1       0.79      0.95      0.86        55

   micro avg       0.76      0.76      0.76        70
   macro avg       0.52      0.51      0.48        70
weighted avg       0.67      0.76      0.70        70

F1 score to accurate Decision Tree
              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
weighted avg       0.69      0.65      0.67       104

F1 score to accurate SVM
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        15
           1       0.79      1.00      0.88        55

   micro avg       0.79      0.79      0.79        70
   macro avg       0.39      0.50      0.44        70
weighted avg       0.62      0.79      0.69        70

F1 score to accurate Logistic Regression
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        15
           1       0.79      1.00      0.88        55

   micro avg       0.79      0.79      0.79        70
   macro avg       0.39      0.50      0.44        70
weighted avg       0.62      0.79      0.69        70

/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are 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:1143: UndefinedMetricWarning: Precision and F-score are 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:1143: UndefinedMetricWarning: Precision and F-score are 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:1143: UndefinedMetricWarning: Precision and F-score are 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:1143: UndefinedMetricWarning: Precision and F-score are 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:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```



```
# Using f1 score to evaluate different models
print('F1 score to accurate KNN')
print (classification_report(y_test, yhat))
print('F1 score to accurate Decision Tree')
print (classification_report(y_testset, predTree))
print('F1 score to accurate SVM')
print (classification_report(y1_test, y1hat))
print('F1 score to accurate Logistic Regression')
print (classification_report(y1_test, y1hat))
```

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	?	?	?

[105]:
```
# Create a report table
import pandas as pd
df = pd.DataFrame({'Jaccard': [0.7571428571428571, 0.6538461538461539, 0.7857142857142857,0.7857142857142857], \
                   'F1-score': [0.70, 0.67 , 0.69, 0.69], \
                   'LogLoss': ['NA', 'NA', 'NA',0.5435595463662589]},
                      index = ['KNN','Decision Tree','SVM','LogisticRegression'])
df.head()
```

[105]:

	Jaccard	F1-score	LogLoss
KNN	0.757143	0.70	NA
Decision Tree	0.653846	0.67	NA
SVM	0.785714	0.69	NA
LogisticRegression	0.785714	0.69	0.54356


```
# Create a report table
import pandas as pd
df = pd.DataFrame({'Jaccard': [0.7571428571428571, 0.6538461538461539,
0.7857142857142857,0.7857142857142857], \
             'F1-score': [0.70, 0.67 , 0.69, 0.69], \
             'LogLoss': ['NA', 'NA', 'NA',0.5435595463662589]},
               index = ['KNN','Decision Tree','SVM','LogisticRegression'])
df.head()
```


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.
