IBM Developer
SKILLS NETWORK

Classification with Python

In this notebook we try to practice all the classification algorithms that we learned in this course.

We load a dataset using Pandas library, and apply the following algorithms, and find the best one for this specific dataset by accuracy evaluation methods.

Lets first load required libraries:

In [1]:
```
import itertools
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
import pandas as pd
import numpy as np
import matplotlib.ticker as ticker
from sklearn import preprocessing
%matplotlib inline
```

About dataset

This dataset is about past loans. The **Loan_train.csv** data set includes details of 346 customers whose loan are already paid off or defaulted. It includes following fields:

Field	Description
Loan_status	Whether a loan is paid off on in collection
Principal	Basic principal loan amount at the
Terms	Origination terms which can be weekly (7 days), biweekly, and monthly payoff schedule
Effective_date	When the loan got originated and took effects
Due_date	Since it’s one-time payoff schedule, each loan has one single due date
Age	Age of applicant
Education	Education of applicant
Gender	The gender of applicant

Lets download the dataset

In [2]:
```
!wget -O loan_train.csv https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/FinalModule_Coursera/data/loan_train.csv
```

```
--2021-01-31 21:21:03--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/FinalModule_Coursera/data/loan_train.csv
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud)... 198.23.119.245
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud)|198.23.119.245|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 23101 (23K) [text/csv]
Saving to: 'loan_train.csv'

loan_train.csv      100%[===================>]  22.56K  --.-KB/s    in 0s

2021-01-31 21:21:03 (176 MB/s) - 'loan_train.csv' saved [23101/23101]
```

Load Data From CSV File

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

Out[3]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	age	education
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 [4]:
```
df.shape
```

Out[4]: (346, 10)

Convert to date time object

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

Out[5]:

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

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

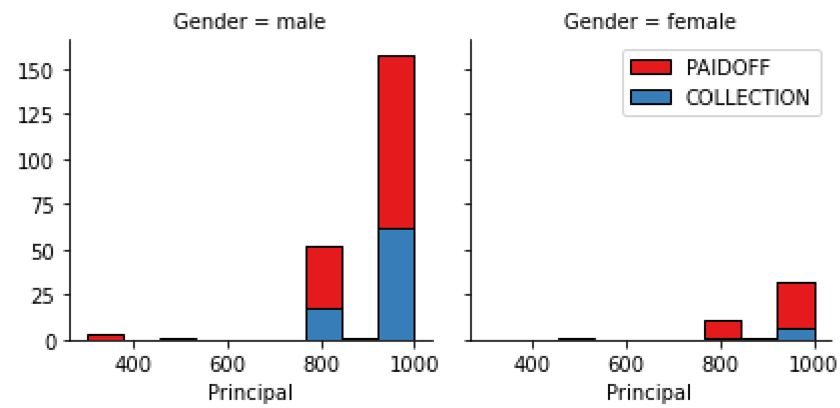
260 people have paid off the loan on time while 86 have gone into collection

Lets plot some columns to underestand data better:

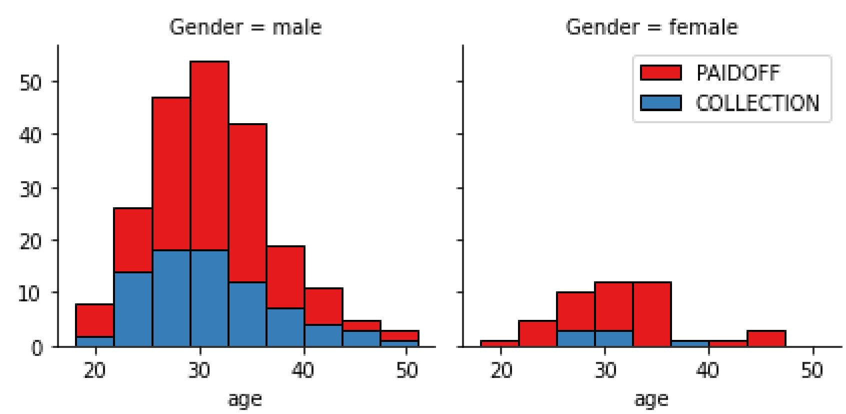
In [7]:
```
# notice: installing seaborn might takes a few minutes
!conda install -c anaconda seaborn -y
```

```
Collecting package metadata (current_repodata.json): done
Solving environment: \ ^C
failed with initial frozen solve. Retrying with flexible solve.

CondaError: KeyboardInterrupt
```

In [8]:
```python
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()
```



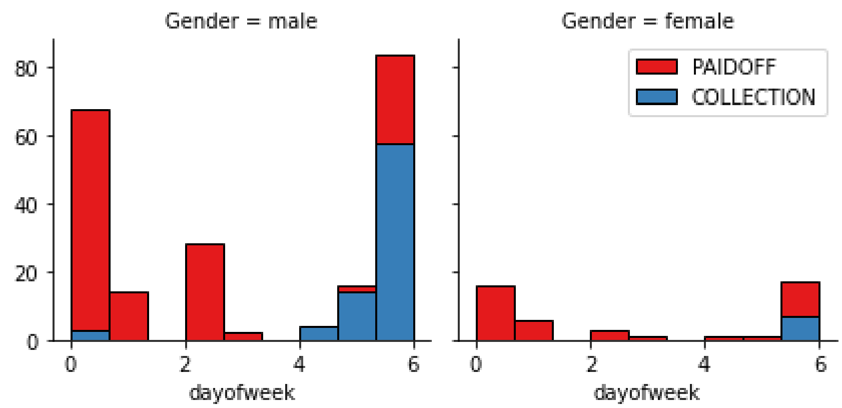
In [9]:
```python
bins = np.linspace(df.age.min(), df.age.max(), 10)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", c
ol_wrap=2)
g.map(plt.hist, 'age', bins=bins, ec="k")

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]
```

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 [30]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
```

In [23]:

```
# 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 [24]:
```
# checking the best value of K
for k in range(1, 10):
    knnModel = KNeighborsClassifier(n_neighbors = k).fit(x_train, y_train)
    knn_yhat = knnModel.predict(x_test)
    print("For K = {} accuracy = {}".format(k,accuracy_score(y_test,knn_yhat)))
```

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

In [32]:
```
best_knn = KNeighborsClassifier(n_neighbors=7).fit(x_train, y_train)
print("Train set Accuracy (Jaccard)  = ", accuracy_score(y_train, best_knn.predict(x_train)))
print("Test set Accuracy (Jaccard) = ", accuracy_score(y_test, best_knn.predict(x_test)))

print("Train set Accuracy (F1 Score) = ", f1_score(y_train, best_knn.predict(x_train), average='weighted'))
print("Test set Accuracy (F1 Score) = ", f1_score(y_test, best_knn.predict(x_test), average='weighted'))
```

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

Decision Tree

In [33]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
```

In [37]:
```
for d in range(1, 10):
    dt = DecisionTreeClassifier(criterion = 'entropy', max_depth = d).fi
t(x_train, y_train)
    dt_yHat = dt.predict(x_test)
    print("For depth = {}  the accuracy score is {} ".format(d, accuracy
_score(y_test, dt_yHat)))

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

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


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

In [40]:
```
best_dt_model = DecisionTreeClassifier(criterion='entropy', max_depth=2)
.fit(x_train, y_train)

print("Train set Accuracy (Jaccard)  = ", accuracy_score(y_train, best_d
t_model.predict(x_train)))
print("Test set Accuracy (Jaccard) = ", accuracy_score(y_test, best_dt_m
odel.predict(x_test)))

print("Train set Accuracy (F1 Score) = ", f1_score(y_train, best_dt_mode
l.predict(x_train), average='weighted'))
print("Test set Accuracy (F1 Score) = ", f1_score(y_test, best_dt_model.
predict(x_test), average='weighted'))
```

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

Support Vector Machine

In [41]:
```
from sklearn import svm
from sklearn.metrics import f1_score
```

In [42]:
```
for k in ('linear', 'poly', 'rbf', 'sigmoid'):
    svm_model = svm.SVC(kernel = k).fit(x_train, y_train)
    svm_yHat = svm_model.predict(x_test)
    print("For kernel: {}, the f1 score is: {}".format(k,f1_score(y_test
,svm_yHat, average='weighted')))


print("\n\nWe can see the rbf has the best f1 score ")
```

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


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

In [43]:
```
best_svm = svm.SVC(kernel='rbf').fit(x_train,y_train)
print("Train set Accuracy (Jaccard): ", accuracy_score(y_train, best_svm
.predict(x_train)))
print("Test set Accuracy (Jaccard): ", accuracy_score(y_test, best_svm.p
redict(x_test)))

print("Train set Accuracy (F1): ", f1_score(y_train, best_svm.predict(x_
train), average='weighted'))
print("Test set Accuracy (F1): ", f1_score(y_test, best_svm.predict(x_te
st), average='weighted'))
```

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

Logistic Regression

In [44]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss
```

In [49]:
```
for idx in ('lbfgs', 'saga', 'liblinear', 'newton-cg', 'sag'):
    lr_model = LogisticRegression(C = 0.01, solver=idx).fit(x_train, y_t
rain)
    lr_yhat = lr_model.predict(x_test)
    y_prob = lr_model.predict_proba(x_test)
    print('When Solver is {}, logloss is : {}'.format(idx, log_loss(y_te
st, y_prob)))

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

```
When Solver is lbfgs, logloss is : 0.4920179847937498
When Solver is saga, logloss is : 0.49201836388409936
When Solver is liblinear, logloss is : 0.5772287609479654
When Solver is newton-cg, logloss is : 0.4920178014679269
When Solver is sag, logloss is : 0.49202623151928626

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

In [56]:
```
best_lr_model = LogisticRegression(C = 0.01, solver = 'liblinear').fit(x
_train, y_train)
print("Train set Accuracy (Jaccard): ", accuracy_score(y_train, best_lr_
model.predict(x_train)))
print("Test set Accuracy (Jaccard): ", accuracy_score(y_test, best_lr_mo
del.predict(x_test)))

print("Train set Accuracy (F1): ", f1_score(y_train, best_lr_model.predi
ct(x_train), average='weighted'))
print("Test set Accuracy (F1): ", f1_score(y_test, best_lr_model.predict
(x_test), average='weighted'))
```

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

Model Evaluation using Test set

In [61]:
```
from sklearn.metrics import jaccard_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
```

First, download and load the test set:

In [52]:
```
!wget -O loan_test.csv https://s3-api.us-geo.objectstorage.softlayer.ne
t/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_test.csv
```

```
--2021-01-31 21:47:06--  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)... 67.228.254.196
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'

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

2021-01-31 21:47:06 (94.4 MB/s) - 'loan_test.csv' saved [3642/3642]
```

Load Test set for evaluation

In [62]:
```
test_df = pd.read_csv('loan_test.csv')
test_df.head()
```

Out[62]:

	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 [63]:

```python
# process data
test_df['due_date'] = pd.to_datetime(test_df['due_date'])
test_df['effective_date'] = pd.to_datetime(test_df['effective_date'])
test_df['dayofweek'] = test_df['effective_date'].dt.dayofweek

test_df['weekend'] = test_df['dayofweek'].apply(lambda x:1 if (x>3) else 0)
test_df['Gender'].replace(to_replace=['male', 'female'], value=[0,1], inplace=True)

Feat1 = test_df[['Principal', 'terms', 'age', 'Gender', 'weekend']]
Feat1 = pd.concat([Feat1, pd.get_dummies(test_df['education'])], axis=1)
Feat1.drop(['Master or Above'], axis=1, inplace=True)

x_loan_test = Feat1
x_loan_test = preprocessing.StandardScaler().fit(x_loan_test).transform(x_loan_test)
y_loan_test = test_df['loan_status'].values
```

In [71]:

```python
# Jaccard Index Values
knn_jacc = round(accuracy_score(y_loan_test, best_knn.predict(x_loan_test)),2)      #KNN
dt_jacc = round(accuracy_score(y_loan_test, best_dt_model.predict(x_loan_test)),2)  # Decision Tree
svm_jacc = round(accuracy_score(y_loan_test, best_svm.predict(x_loan_test)),2)      # Support Vector Machine
lr_jacc = round(accuracy_score(y_loan_test, best_lr_model.predict(x_loan_test)),2) # Logistic Regression

jss = [knn_jacc, dt_jacc, svm_jacc, lr_jacc]
print("Jaccard Index Values:", jss)



# F1 Scores
knn_f1 = round(f1_score(y_loan_test, best_knn.predict(x_loan_test), average='weighted'),2)     # KNN
dt_f1 = round(f1_score(y_loan_test, best_dt_model.predict(x_loan_test), average='weighted'),2) # Decision Tree
svm_f1 = round(f1_score(y_loan_test, best_svm.predict(x_loan_test), average='weighted'),2)     # Support Vector Machine
lr_f1 = round(f1_score(y_loan_test, best_lr_model.predict(x_loan_test), average='weighted'),2) # Logistic Regression

f1_vals = [knn_f1, dt_f1, svm_f1, lr_f1]
print("F1-Score Values:",f1_vals)

# Log Loss (Logistic Regression)
lr_prob = round(log_loss(y_loan_test, best_lr_model.predict_proba(x_loan_test)),2)
logLoss_vals = ['NA', 'NA', 'NA', lr_prob]
print("LogLoss Values:", logLoss_vals)
```

```
Jaccard Index Values: [0.67, 0.74, 0.8, 0.74]
F1-Score Values: [0.63, 0.63, 0.76, 0.66]
LogLoss Values: ['NA', 'NA', 'NA', 0.57]
```

In [72]:
```
columns = ['KNN', 'Decision Tree', 'SVM', 'Logistic Regression']
index = ['Jaccard', 'F1-Score', 'LogLoss']

accuracy_df = pd.DataFrame([jss, f1_vals, logLoss_vals], index=index, co
lumns=columns)
final_df = accuracy_df.transpose()
final_df.columns.name = 'Algorithm'
final_df
```

Out[72]:

Algorithm	Jaccard	F1-Score	LogLoss
KNN	0.67	0.63	NA
Decision Tree	0.74	0.63	NA
SVM	0.8	0.76	NA
Logistic Regression	0.74	0.66	0.57

Report

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

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

Want to learn more?

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

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-10-27	2.1	Lakshmi Holla	Made changes in import statement due to updates in version of sklearn library
2020-08-27	2.0	Malika Singla	Added lab to GitLab
