(https://www.bigdatauniversity.com)

Classification with Python

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

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

Lets first load required libraries:

In [178]:
```
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 [179]:
```
!wget -O loan_train.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_train.csv
```

```
--2019-08-19 23:06:25--  https://s3-api.us-geo.objectstorage.softl
ayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_train
.csv
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo
.objectstorage.softlayer.net)... 67.228.254.193
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us
-geo.objectstorage.softlayer.net)|67.228.254.193|:443... connected
.
HTTP request sent, awaiting response... 200 OK
Length: 23101 (23K) [text/csv]
Saving to: ‘loan_train.csv’

100%[======================================>] 23,101      --.-K/s
in 0.07s

2019-08-19 23:06:25 (303 KB/s) - ‘loan_train.csv’ saved [23101/231
01]
```

Load Data From CSV File

In [180]:
```
train_df = pd.read_csv('loan_train.csv')
train_df.head()
```

Out[180]:

	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 [181]:
```
train_df.shape
```

Out[181]: (346, 10)

Convert to date time object

In [182]:
```
train_df['due_date'] = pd.to_datetime(train_df['due_date'])
train_df['effective_date'] = pd.to_datetime(train_df['effective_dat
e'])
train_df.head()
```

Out[182]:

	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 [183]:
```
train_df['loan_status'].value_counts()
```

Out[183]:
```
PAIDOFF       260
COLLECTION     86
Name: loan_status, dtype: int64
```

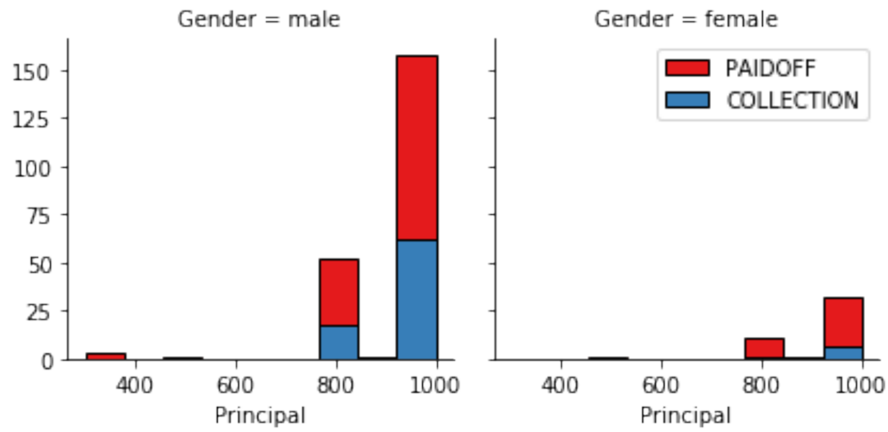
260 people have paid off the loan on time while 86 have gone into collection

Lets plot some columns to underestand data better:

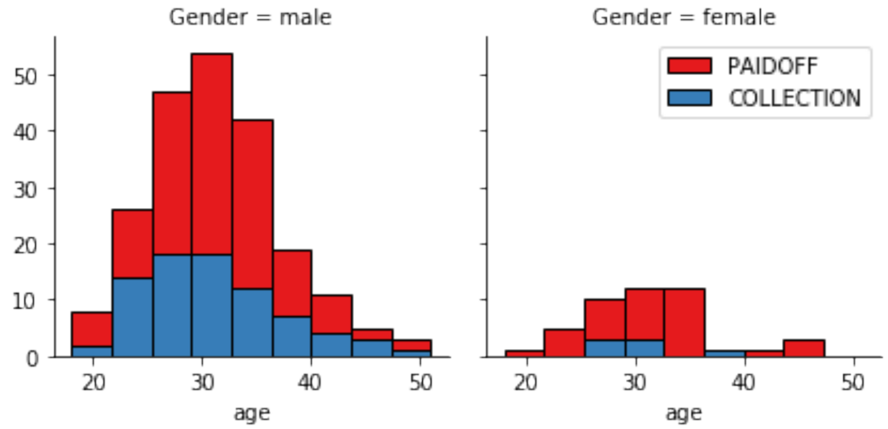
In [184]:
```
# notice: installing seaborn might takes a few minutes
!conda install -c anaconda seaborn -y
```

```
Solving environment: done

# All requested packages already installed.
```

In [185]:
```
import seaborn as sns

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

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



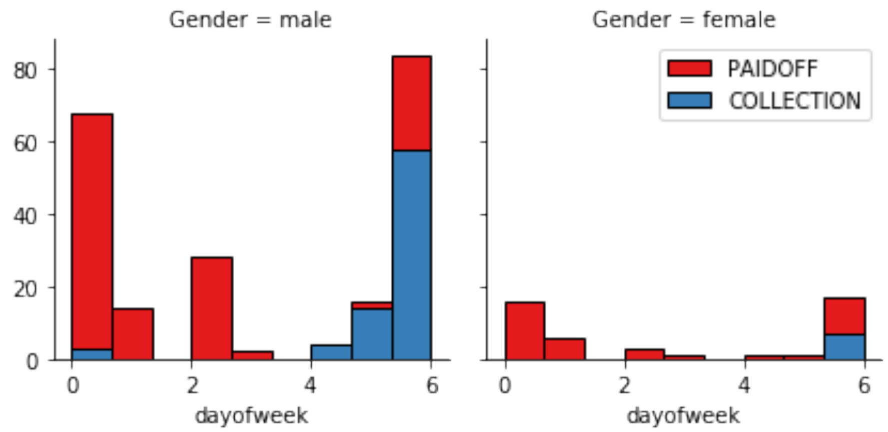
In [186]:
```
bins = np.linspace(train_df.age.min(), train_df.age.max(), 10)
g = sns.FacetGrid(train_df, col="Gender", hue="loan_status", palett
e="Set1", col_wrap=2)
g.map(plt.hist, 'age', bins=bins, ec="k")

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



Pre-processing: Feature selection/extraction

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

In [187]:
```
train_df['dayofweek'] = train_df['effective_date'].dt.dayofweek
bins = np.linspace(train_df.dayofweek.min(), train_df.dayofweek.max
(), 10)
g = sns.FacetGrid(train_df, col="Gender", hue="loan_status", palett
e="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 [188]:
```
train_df['weekend'] = train_df['dayofweek'].apply(lambda x: 1 if (x
>3)  else 0)
train_df.head()
```

Out[188]:

	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 [189]:
```
train_df.groupby(['Gender'])['loan_status'].value_counts(normalize=
True)
```

Out[189]:
```
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 [190]:
```
train_df['Gender'].replace(to_replace=['male','female'], value=[0,1
],inplace=True)
train_df.head()
```

Out[190]:

	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 [191]:
```
train_df.groupby(['education'])['loan_status'].value_counts(normali
ze=True)
```

Out[191]:
```
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 [192]:
```
train_df[['Principal','terms','age','Gender','education']].head()
```

Out[192]:

	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 [193]:
```
Feature_train = train_df[['Principal','terms','age','Gender','weeke
nd']]
Feature_train = pd.concat([Feature_train,pd.get_dummies(train_df['e
ducation'])], axis=1)
Feature_train.drop(['Master or Above'], axis = 1,inplace=True)

Feature_train.head()
```

Out[193]:

	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 [194]:
```
X_train = Feature_train
X_train[0:5]
```

Out[194]:

	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 [195]:
```
y_train = train_df['loan_status'].values
y_train[0:5]
```

Out[195]:
```
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 [196]:
```
X_train = preprocessing.StandardScaler().fit(X_train).transform(X_t
rain)
X_train[0:5]
```

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/prepr
ocessing/data.py:645: DataConversionWarning: Data with input dtype
uint8, int64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/opt/conda/envs/Python36/lib/python3.6/site-packages/ipykernel/__m
ain__.py:1: DataConversionWarning: Data with input dtype uint8, in
t64 were all converted to float64 by StandardScaler.
  if __name__ == '__main__':
```

Out[196]:
```
array([[ 0.51578458,  0.92071769,  2.33152555, -0.42056004, -1.205
77805,
        -0.38170062,  1.13639374, -0.86968108],
       [ 0.51578458,  0.92071769,  0.34170148,  2.37778177, -1.205
77805,
         2.61985426, -0.87997669, -0.86968108],
       [ 0.51578458, -0.95911111, -0.65321055, -0.42056004, -1.205
77805,
        -0.38170062, -0.87997669,  1.14984679],
       [ 0.51578458,  0.92071769, -0.48739188,  2.37778177,  0.829
34003,
        -0.38170062, -0.87997669,  1.14984679],
       [ 0.51578458,  0.92071769, -0.3215732 , -0.42056004,  0.829
34003,
        -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 [197]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [198]:
```
k_1 = 7

neigh_1 = KNeighborsClassifier(n_neighbors = k_1).fit(X_train,y_tra
in)
neigh_1
```

Out[198]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minko
wski',
                     metric_params=None, n_jobs=None, n_neighbors=7, p=2,
                     weights='uniform')
```

when k = 7, the model has the best accuracy (0.78571429)

Decision Tree

In [199]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

In [200]:
```
drugTree = DecisionTreeClassifier(criterion="entropy", max_depth =
4)
drugTree
```

Out[200]:
```
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_st
ate=None,
                       splitter='best')
```

In [201]:
```
drugTree.fit(X_train,y_train)
```

Out[201]:
```
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_st
ate=None,
            splitter='best')
```

Support Vector Machine

In [202]:
```
from sklearn import svm
SVM = svm.SVC(kernel='rbf')
SVM.fit(X_train, y_train)
```

Out[202]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated'
,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

Logistic Regression

In [203]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_t
rain)
LR
```

Out[203]:
```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_inte
rcept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='li
blinear',
          tol=0.0001, verbose=0, warm_start=False)
```

In [204]:
```
yhat = LR.predict(X_test)
```

Model Evaluation using Test set

In [205]:
```
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
```

First, download and load the test set:

In [206]:
```
!wget -O loan_test.csv https://s3-api.us-geo.objectstorage.softlaye
r.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_test.csv
```

```
--2019-08-19 23:11:22--  https://s3-api.us-geo.objectstorage.softl
ayer.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)... 67.228.254.193
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us
-geo.objectstorage.softlayer.net)|67.228.254.193|:443... connected
.
HTTP request sent, awaiting response... 200 OK
Length: 3642 (3.6K) [text/csv]
Saving to: ‘loan_test.csv’

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

2019-08-19 23:11:22 (682 MB/s) - ‘loan_test.csv’ saved [3642/3642]
```

Load Test set for evaluation

In [207]:
```
test_df = pd.read_csv('loan_test.csv')
test_df.head()
```

Out[207]:

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

In [208]:
```
test_df['due_date'] = pd.to_datetime(test_df['due_date'])
test_df['effective_date'] = pd.to_datetime(test_df['effective_date'
])
test_df.head()
```

Out[208]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	ag
0	1	1	PAIDOFF	1000	30	2016-09-08	2016-10-07	50
1	5	5	PAIDOFF	300	7	2016-09-09	2016-09-15	35
2	21	21	PAIDOFF	1000	30	2016-09-10	2016-10-09	43
3	24	24	PAIDOFF	1000	30	2016-09-10	2016-10-09	26
4	35	35	PAIDOFF	800	15	2016-09-11	2016-09-25	29

In [209]:
```
test_df['dayofweek'] = test_df['effective_date'].dt.dayofweek
```

In [210]:
```
test_df['weekend'] = test_df['dayofweek'].apply(lambda x: 1 if (x>3
)  else 0)
test_df.head()
```

Out[210]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	ag
0	1	1	PAIDOFF	1000	30	2016-09-08	2016-10-07	50
1	5	5	PAIDOFF	300	7	2016-09-09	2016-09-15	35
2	21	21	PAIDOFF	1000	30	2016-09-10	2016-10-09	43
3	24	24	PAIDOFF	1000	30	2016-09-10	2016-10-09	26
4	35	35	PAIDOFF	800	15	2016-09-11	2016-09-25	29

In [211]:
```
test_df.groupby(['Gender'])['loan_status'].value_counts(normalize=True)
```

Out[211]:
```
Gender  loan_status
female  PAIDOFF        0.727273
        COLLECTION     0.272727
male    PAIDOFF        0.744186
        COLLECTION     0.255814
Name: loan_status, dtype: float64
```

In [212]:
```
test_df['Gender'].replace(to_replace=['male','female'], value=[0,1],inplace=True)
test_df.head()
```

Out[212]:

	Unnamed: 0	Unnamed: 0.1	loan_status	Principal	terms	effective_date	due_date	ag
0	1	1	PAIDOFF	1000	30	2016-09-08	2016-10-07	50
1	5	5	PAIDOFF	300	7	2016-09-09	2016-09-15	35
2	21	21	PAIDOFF	1000	30	2016-09-10	2016-10-09	43
3	24	24	PAIDOFF	1000	30	2016-09-10	2016-10-09	26
4	35	35	PAIDOFF	800	15	2016-09-11	2016-09-25	29

In [213]:
```
test_df.groupby(['education'])['loan_status'].value_counts(normalize=True)
```

Out[213]:
```
education             loan_status
Bechalor              PAIDOFF        1.000000
High School or Below  PAIDOFF        0.523810
                      COLLECTION     0.476190
Master or Above       PAIDOFF        1.000000
college               PAIDOFF        0.826087
                      COLLECTION     0.173913
Name: loan_status, dtype: float64
```

In [214]:
```
test_df[['Principal','terms','age','Gender','education']].head()
```

Out[214]:

	Principal	terms	age	Gender	education
0	1000	30	50	1	Bechalor
1	300	7	35	0	Master or Above
2	1000	30	43	1	High School or Below
3	1000	30	26	0	college
4	800	15	29	0	Bechalor

In [215]:
```
Feature_test = test_df[['Principal','terms','age','Gender','weekend
']]
Feature_test = pd.concat([Feature_test,pd.get_dummies(test_df['educ
ation'])], axis=1)
Feature_test.drop(['Master or Above'], axis = 1,inplace=True)

Feature_test.head()
```

Out[215]:

	Principal	terms	age	Gender	weekend	Bechalor	High School or Below	college
0	1000	30	50	1	0	1	0	0
1	300	7	35	0	1	0	0	0
2	1000	30	43	1	1	0	1	0
3	1000	30	26	0	1	0	0	1
4	800	15	29	0	1	1	0	0

In [216]:
```
X_test = Feature_test
X_test[0:5]
```

Out[216]:

	Principal	terms	age	Gender	weekend	Bechalor	High School or Below	college
0	1000	30	50	1	0	1	0	0
1	300	7	35	0	1	0	0	0
2	1000	30	43	1	1	0	1	0
3	1000	30	26	0	1	0	0	1
4	800	15	29	0	1	1	0	0

In [217]:
```
y_test = test_df['loan_status'].values
y_test[0:5]
```

Out[217]:
```
array(['PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF', 'PAIDOFF'],
      dtype=object)
```

In [218]:
```
X_test= preprocessing.StandardScaler().fit(X_test).transform(X_test
)
X_test[0:5]
```

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/prepr
ocessing/data.py:645: DataConversionWarning: Data with input dtype
uint8, int64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/opt/conda/envs/Python36/lib/python3.6/site-packages/ipykernel/__m
ain__.py:1: DataConversionWarning: Data with input dtype uint8, in
t64 were all converted to float64 by StandardScaler.
  if __name__ == '__main__':
```

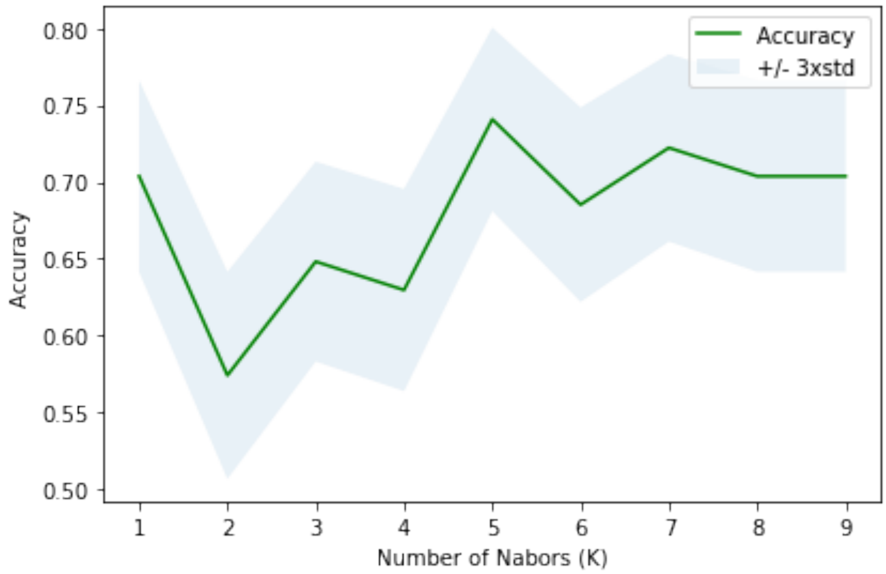
Out[218]:
```
array([[ 0.49362588,  0.92844966,  3.05981865,  1.97714211, -1.303
84048,
         2.39791576, -0.79772404, -0.86135677],
       [-3.56269116, -1.70427745,  0.53336288, -0.50578054,  0.766
96499,
        -0.41702883, -0.79772404, -0.86135677],
       [ 0.49362588,  0.92844966,  1.88080596,  1.97714211,  0.766
96499,
        -0.41702883,  1.25356634, -0.86135677],
       [ 0.49362588,  0.92844966, -0.98251057, -0.50578054,  0.766
96499,
        -0.41702883, -0.79772404,  1.16095912],
       [-0.66532184, -0.78854628, -0.47721942, -0.50578054,  0.766
96499,
         2.39791576, -0.79772404, -0.86135677]])
```

K Nearest Neighbor(KNN)

In [219]:
```
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):


    KNN = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train
)
    yhat_knn=KNN.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat_knn)


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

mean_acc
```

Out[219]:
```
array([0.7037037 , 0.57407407, 0.64814815, 0.62962963, 0.74074074,
       0.68518519, 0.72222222, 0.7037037 , 0.7037037 ])
```

In [220]:
```
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()
```



when k = 7, the model has the best accuracy (0.78571429)

In [221]:
```
k_1 = 5

KNN_5 = KNeighborsClassifier(n_neighbors = k_1).fit(X_train,y_train
)
KNN_5
```

Out[221]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minko
wski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform')
```

In [222]:
```
yhat_knn_5 = KNN_5.predict(X_test)
```

In [223]:
```
from sklearn.metrics import jaccard_similarity_score
jaccard_similarity_score(y_test, yhat_knn_5)
```

Out[223]:
```
0.7407407407407407
```

In [224]:
```
f1_score(y_test, yhat_knn_5, average='weighted')
```

Out[224]:
```
0.7253086419753088
```

Decision Tree

In [225]:
```
drugTree = DecisionTreeClassifier(criterion="entropy", max_depth =
4).fit(X_train,y_train)
drugTree
```

Out[225]:
```
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_st
ate=None,
            splitter='best')
```

In [226]:
```
predTree = drugTree.predict(X_test)
```

In [227]:
```
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_test,
predTree))
```

```
DecisionTrees's Accuracy:  0.7777777777777778
```

In [228]:
```
jaccard_similarity_score(y_test, predTree)
```

Out[228]:
```
0.7777777777777778
```

In [229]:
```
f1_score(y_test, predTree, average='weighted')
```

Out[229]:
```
0.7283950617283951
```

Support Vector Machine

In [230]:
```
from sklearn import svm
SVM = svm.SVC(kernel='rbf')
SVM.fit(X_train, y_train)
```

Out[230]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated'
,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

In [231]:
```
yhat_svm = SVM.predict(X_test)
```

In [232]:
```
jaccard_similarity_score(y_test, yhat_svm)
```

Out[232]:
```
0.7222222222222222
```

In [233]:
```
f1_score(y_test, yhat_svm, average='weighted')
```

Out[233]:
```
0.6212664277180406
```

In [ ]:
```

```

Logistic regression

In [234]:
```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_t
rain)
LR
```

Out[234]:
```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_inte
rcept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='li
blinear',
          tol=0.0001, verbose=0, warm_start=False)
```

In [235]:
```
yhat_logistic = LR.predict(X_test)
```

In [236]:
```
jaccard_similarity_score(y_test, yhat_logistic)
```

Out[236]: 0.7407407407407407

In [237]:
```
f1_score(y_test, yhat_logistic, average='weighted')
```

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

Out[237]: 0.6304176516942475

In [238]:
```
yhat_prob = LR.predict_proba(X_test)
from sklearn.metrics import log_loss
log_loss(y_test, yhat_prob)
```

Out[238]: 0.5566084946309207

In [ ]:

In [ ]:

In [ ]:

Report

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

Algorithm	Jaccard	F1-score	LogLoss
KNN	0.7407	0.7253	NA
Decision Tree	0.7778	0.7284	NA
SVM	0.7222	0.6213	NA
LogisticRegression	0.7407	0.6304	0.5566

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.
