(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 [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://s3-api.us-geo.objectstorage.softlayer.net/cf-co
urses-data/CognitiveClass/ML0101ENv3/labs/loan_train.csv
```

```
--2020-10-02 08:02:25--  https://s3-api.us-geo.objectstorage.softl
ayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_trai
n.csv
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-ge
o.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... connecte
d.
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

2020-10-02 08:02:26 (304 KB/s) - ‘loan_train.csv’ saved [23101/231
01]
```

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	a
0	0	0	PAIDOFF	1000	30	9/8/2016	10/7/2016	
1	2	2	PAIDOFF	1000	30	9/8/2016	10/7/2016	
2	3	3	PAIDOFF	1000	15	9/8/2016	9/22/2016	
3	4	4	PAIDOFF	1000	30	9/9/2016	10/8/2016	
4	6	6	PAIDOFF	1000	30	9/9/2016	10/8/2016	

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	a
0	0	0	PAIDOFF	1000	30	2016-09-08	2016-10-07	
1	2	2	PAIDOFF	1000	30	2016-09-08	2016-10-07	
2	3	3	PAIDOFF	1000	15	2016-09-08	2016-09-22	
3	4	4	PAIDOFF	1000	30	2016-09-09	2016-10-08	
4	6	6	PAIDOFF	1000	30	2016-09-09	2016-10-08	

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

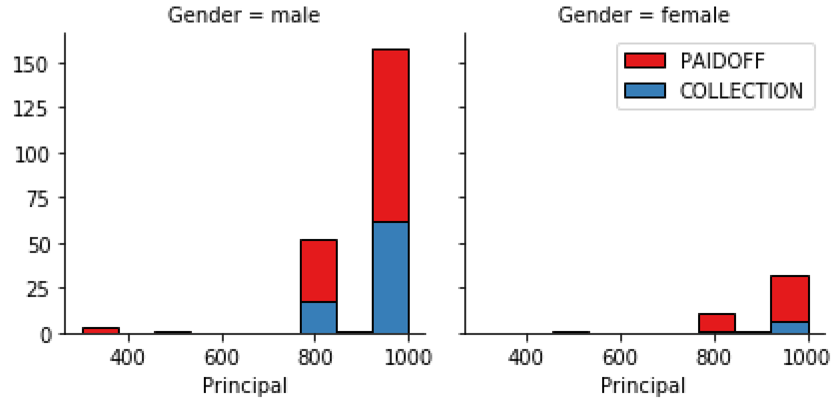
Lets plot some columns to underestand data better:

In [7]:

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

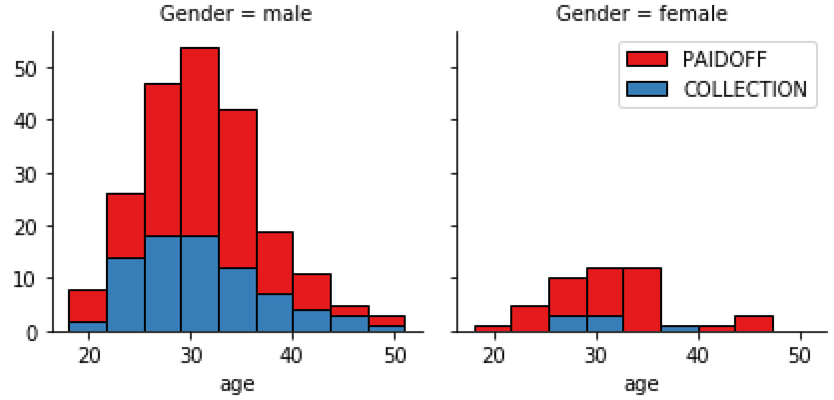
In [8]:

```
import seaborn as sns

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

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



In [9]:

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

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

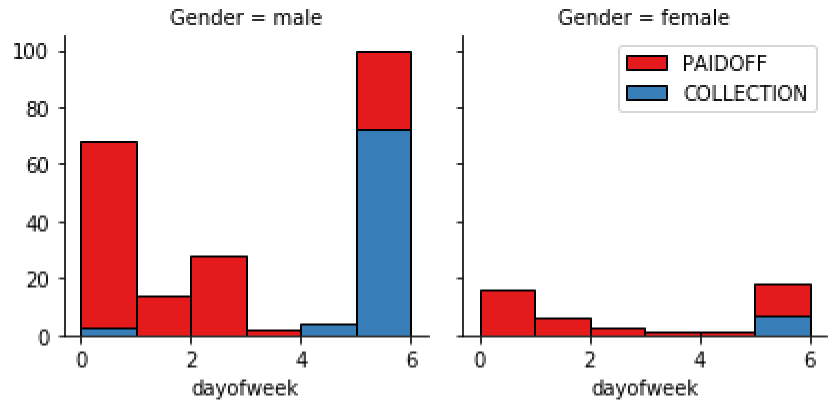


Pre-processing: Feature selection/extraction

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

In [10]:

```
df['dayofweek'] = df['effective_date'].dt.dayofweek
bins = np.linspace(df.dayofweek.min(), df.dayofweek.max(), 7)
g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wrap
=2)
g.map(plt.hist, 'dayofweek', bins=bins, ec="k")
g.axes[-1].legend()
plt.show()
```



We see that people who get the loan at the end of the week dont pay it off, so lets use Feature binarization to set a threshold values less then day 4

In [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	a
0	0	0	PAIDOFF	1000	30	2016-09-08	2016-10-07	
1	2	2	PAIDOFF	1000	30	2016-09-08	2016-10-07	
2	3	3	PAIDOFF	1000	15	2016-09-08	2016-09-22	
3	4	4	PAIDOFF	1000	30	2016-09-09	2016-10-08	
4	6	6	PAIDOFF	1000	30	2016-09-09	2016-10-08	

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	a
0	0	0	PAIDOFF	1000	30	2016-09-08	2016-10-07	
1	2	2	PAIDOFF	1000	30	2016-09-08	2016-10-07	
2	3	3	PAIDOFF	1000	15	2016-09-08	2016-09-22	
3	4	4	PAIDOFF	1000	30	2016-09-09	2016-10-08	
4	6	6	PAIDOFF	1000	30	2016-09-09	2016-10-08	

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 convert 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)
print(Feature.head())
Feature.drop(['Master or Above'], axis = 1,inplace=True)
Feature.head()
```

```
   Principal  terms  age  Gender  weekend  Bechalor  High School o
r Below  \
0       1000     30   45       0        0         0
1
1       1000     30   33       1        0         1
0
2       1000     15   27       0        0         0
0
3       1000     30   28       1        1         0
0
4       1000     30   29       0        1         0
0

   Master or Above  college
0                0        0
1                0        0
2                0        1
3                0        1
4                0        1
```

Out[16]:

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

Feature selection

Lets defind feature sets, X:

In [17]:

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

Out[17]:

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

What are our lables?

In [18]:

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

Out[18]:

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

Normalize Data

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

In [19]:

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

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/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[19]:

```
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**.

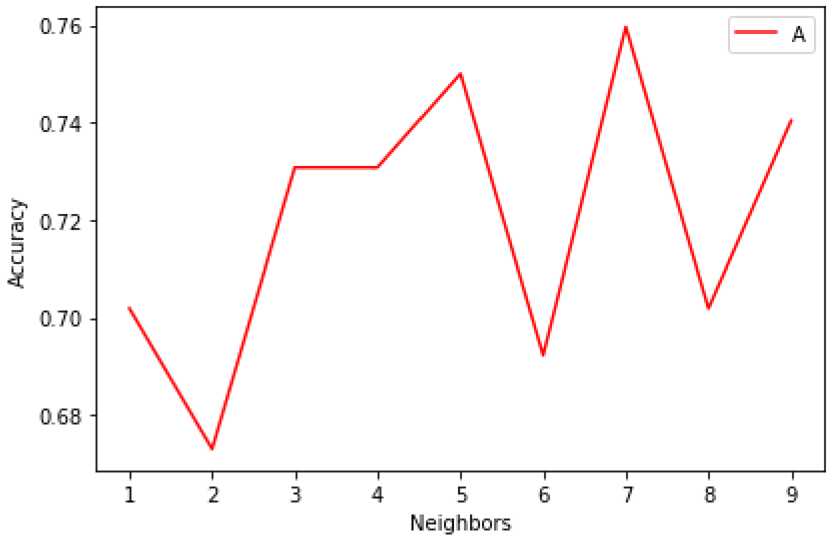
In [20]:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
X_train, X_test, y_train, y_test = train_test_split(Feature, df['loan_status'],
test_size = 0.3, random_state = 4)
X_train = preprocessing.StandardScaler().fit(X_train).transform(X_train)
X_test = preprocessing.StandardScaler().fit(X_test).transform(X_test)
```

```
/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:4: DataConversionWarning: Data with input dtype uint8, in
t64 were all converted to float64 by StandardScaler.
/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:5: DataConversionWarning: Data with input dtype uint8, in
t64 were all converted to float64 by StandardScaler.
```

In [21]:

```
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])
plt.plot(range(1,Ks), mean_acc, 'r')
plt.legend("Accuracy Score")
plt.xlabel("Neighbors")
plt.ylabel("Accuracy")
plt.tight_layout()
plt.show()
```



In [22]:

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

The best accuracy was 0.7596153846153846 with k = 7

In [23]:

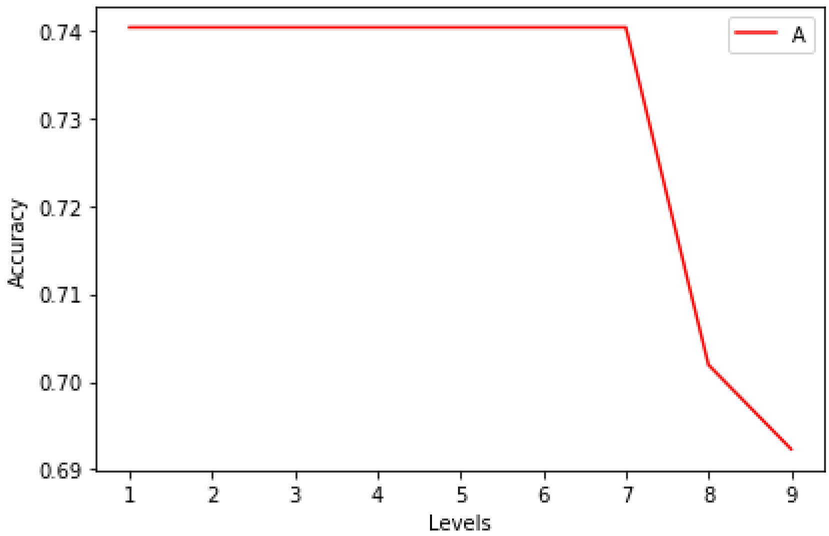
```
knn_mod = KNeighborsClassifier(n_neighbors = 7).fit(X, y)
```

Decision Tree

In [24]:

```
from sklearn.tree import DecisionTreeClassifier
levels = 10
acc_score = np.zeros((levels-1))
for n in range(1, levels):
    dtree = DecisionTreeClassifier(criterion = "entropy", max_depth = n).fit(X_
train, y_train)
    yhat = dtree.predict(X_test)
    acc_score[n-1] = metrics.accuracy_score(yhat, y_test)
print(acc_score)
plt.plot(range(1,levels), acc_score, 'r')
plt.legend("Accuracy Score")
plt.xlabel("Levels")
plt.ylabel("Accuracy")
plt.tight_layout()
plt.show()
```

```
[0.74038462 0.74038462 0.74038462 0.74038462 0.74038462 0.74038462
 0.74038462 0.70192308 0.69230769]
```



In [25]:

```
print("The best accuracy was", acc_score.max(), "with level =", acc_score.argma
x() + 1)
```

```
The best accuracy was 0.7403846153846154 with level = 1
```

In [26]:

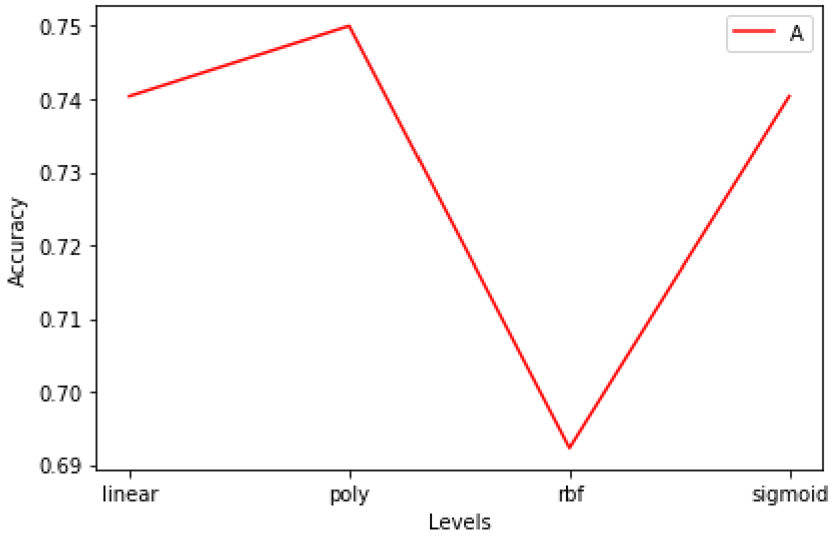
```
dtree_mod = DecisionTreeClassifier(criterion = "entropy", max_depth = 7).fit(X,
y)
```

Support Vector Machine

In [27]:

```
from sklearn import svm
k_functions = ['linear', 'poly', 'rbf', 'sigmoid']
acc_score = np.zeros((4))
for n in range(0,len(k_functions)):
    clf = svm.SVC(kernel = k_functions[n]).fit(X_train, y_train)
    yhat = clf.predict(X_test)
    acc_score[n-1] = metrics.accuracy_score(yhat, y_test)
print(acc_score)
plt.plot(k_functions, acc_score, 'r')
plt.legend("Accuracy Score")
plt.xlabel("Levels")
plt.ylabel("Accuracy")
plt.tight_layout()
plt.show()
```

```
[0.74038462 0.75       0.69230769 0.74038462]
```



In [28]:

```
print("The best accuracy was", acc_score.max(), "with level =", k_functions[acc
_score.argmax()])
```

```
The best accuracy was 0.75 with level = poly
```

In [29]:

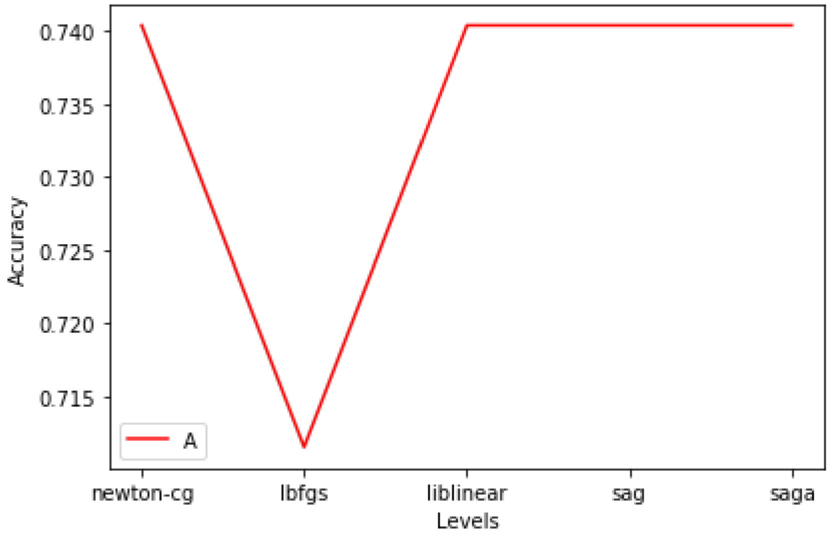
```
svm_mod = svm.SVC(kernel = "poly").fit(X, y)
```

Logistic Regression

In [30]:

```
from sklearn.linear_model import LogisticRegression
k_functions = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
acc_score = np.zeros((5))
for n in range(0,len(k_functions)):
    LR = LogisticRegression(C = 0.01, solver = k_functions[n]).fit(X_train, y_t
rain)
    yhat = LR.predict(X_test)
    acc_score[n-1] = metrics.accuracy_score(yhat, y_test)
print(acc_score)
plt.plot(k_functions, acc_score, 'r')
plt.legend("Accuracy Score")
plt.xlabel("Levels")
plt.ylabel("Accuracy")
plt.tight_layout()
plt.show()
```

```
[0.74038462 0.71153846 0.74038462 0.74038462 0.74038462]
```



In [31]:

```
print("The best accuracy was", acc_score.max(), "with level =", k_functions[acc
_score.argmax()])
```

```
The best accuracy was 0.7403846153846154 with level = newton-cg
```

In [32]:

```
lr_mod = LogisticRegression(C = 0.01, solver = "newton-cg").fit(X, y)
```

Model Evaluation using Test set

In [33]:

```
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 [34]:

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

```
--2020-10-02 08:02:30--  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-ge
o.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... connecte
d.
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

2020-10-02 08:02:30 (348 MB/s) - ‘loan_test.csv’ saved [3642/3642]
```

Load Test set for evaluation

In [35]:

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

Out[35]:

	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	
1	5	5	PAIDOFF	300	7	9/9/2016	9/15/2016	
2	21	21	PAIDOFF	1000	30	9/10/2016	10/9/2016	
3	24	24	PAIDOFF	1000	30	9/10/2016	10/9/2016	
4	35	35	PAIDOFF	800	15	9/11/2016	9/25/2016	

In [36]:

```
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.groupby(['Gender'])['loan_status'].value_counts(normalize=True)
test_df['Gender'].replace(to_replace=['male','female'], value=[0,1],inplace=Tru
e)
test_df.groupby(['education'])['loan_status'].value_counts(normalize=True)
test_df[['Principal','terms','age','Gender','education']].head()
Feature_test = test_df[['Principal','terms','age','Gender','weekend']]
Feature_test = pd.concat([Feature_test,pd.get_dummies(test_df['education'])], a
xis=1)
print(Feature_test.head())
Feature_test.drop(['Master or Above'], axis = 1,inplace=True)
test_y = test_df['loan_status'].values
test_df = Feature_test
test_df = preprocessing.StandardScaler().fit(test_df).transform(test_df)
```

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

   Master or Above  college
0                0        0
1                1        0
2                0        0
3                0        1
4                0        0

/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:15: DataConversionWarning: Data with input dtype uint8, i
nt64 were all converted to float64 by StandardScaler.
```

In [37]:

```
# KNN
yhat_knn = knn_mod.predict(test_df)
print("Jaccard Similarity Score:", jaccard_similarity_score(yhat_knn, test_y))
print("F1 Score:", f1_score(yhat_knn, test_y, average = "weighted"))
```

```
Jaccard Similarity Score: 0.7222222222222222
F1 Score: 0.7442455242966752
```

In [38]:

```
# Decision Trees
yhat_dtree = dtree_mod.predict(test_df)
print("Jaccard Similarity Score:", jaccard_similarity_score(yhat_dtree, test_y
))
print("F1 Score:", f1_score(yhat_dtree, test_y, average = "weighted"))
```

```
Jaccard Similarity Score: 0.7592592592592593
F1 Score: 0.7783461210571184
```

In [39]:

```
# Support Vector Machines
yhat_svm = svm_mod.predict(test_df)
print("Jaccard Similarity Score:", jaccard_similarity_score(yhat_svm, test_y))
print("F1 Score:", f1_score(yhat_svm, test_y, average = "weighted"))
```

```
Jaccard Similarity Score: 0.7407407407407407
F1 Score: 0.7983539094650205
```

In [40]:

```
# Logistic Regression
yhat_lr = lr_mod.predict(test_df)
print("Jaccard Similarity Score:", jaccard_similarity_score(yhat_lr, test_y))
print("F1 Score:", f1_score(yhat_lr, test_y, average = "weighted"))
print("Log Loss:", log_loss(test_y, LR.predict_proba(test_df)))
```

```
Jaccard Similarity Score: 0.7407407407407407
F1 Score: 0.851063829787234
Log Loss: 0.5252783993638966
```

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/metri
cs/classification.py:1145: UndefinedMetricWarning: F-score is ill-
defined and being set to 0.0 in labels with no true samples.
  'recall', 'true', average, warn_for)
```

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.
