Sravani Konujula

Class ID-23

16230172

Lab Assignment-3

Objective:

The purpose of this assignment is

- Making a prediction on the dataset using LDA
- Applying Linear and RBF kernel on SVM
- Applying Lemmatization and bigram on text
- Finding the results of K nearest neighbors with K values.

Features:

- Graph has been displayed for the considered dataset using LDA implementation
- Using SVM implementation with the scikit-learn and applying SVC with linear kernel and RBF kernel
- By using Lemmatization and bigram frequency on words to display top 5 words
  And concatenate the sentences with high frequency words
- To display the results of k nearest neighbor algorithm for k=1 and k=50 values.

Configuration:

Pycharm
Python: 2.7.13

Output Screens

Question1:

Source Code:

LDA:

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import accuracy_score
from sklearn import datasets, metrics
from sklearn.linear_model import LogisticRegression
# loading iris dataset
irisDS= datasets.load_iris()
info = irisDS.data
tag = irisDS.target
#splitting data into train and test
info_train, info_test, tag_train, tag_test=train_test_split(info, tag, test_size=0.2, random_state=36)
#LDA
design = LinearDiscriminantAnalysis()
#fitting data
Q=design.fit(info_train, tag_train).transform(info)
targ_nams = irisDS.target_names
# predicting data
y_predLDA=design.predict(info_test)
# accuracy
A=metrics.accuracy_score(y_predLDA, tag_test)
print(" \n Required accuracy score is: ", A)

plt.figure()
colours = ['blue', 'purple', 'green']
for colour, k, targ_nam in zip(colours, [0, 1, 2], targ_nams):
    plt.scatter(Q[tag == k, 0], Q[tag == k, 1], alpha=.8, color=colour,
                label=targ_nam)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('LDA for the choosen IRIS dataset')
plt.show()
```

LR:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import datasets, metrics
from sklearn.linear_model import LogisticRegression
# loading iris dataset
irisDS= datasets.load_iris()
info = irisDS.data
tag = irisDS.target
#splitting data into train and test
info_train, info_test, tag_train, tag_test=train_test_split(info, tag, test_size=0.2, random_state=36)
design = LogisticRegression()
#fitting data
Q=design.fit(info_train, tag_train)
targ_nams = irisDS.target_names
# predicting data
y_predLDA=design.predict(info_test)
# accuracy
LR=metrics.accuracy_score(y_predLDA, tag_test)
print(" \n Required accuracy score is: ", LR)
```

Output Screen:

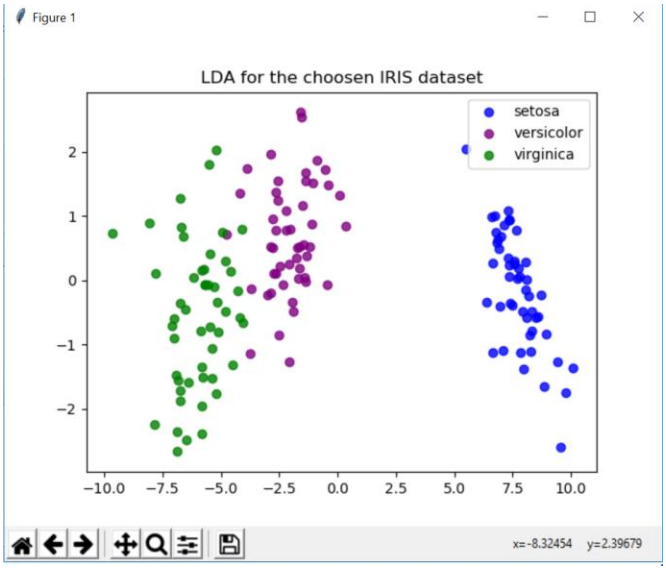
LDA output:

```
24    plt.
```

LDA | LDA | L1

```
C:\Users\Sravani\PycharmProjects\Lab3\venv\Scr
```

```
Required accuracy score is:  1.0
```



LR output:

DA | LR | L1

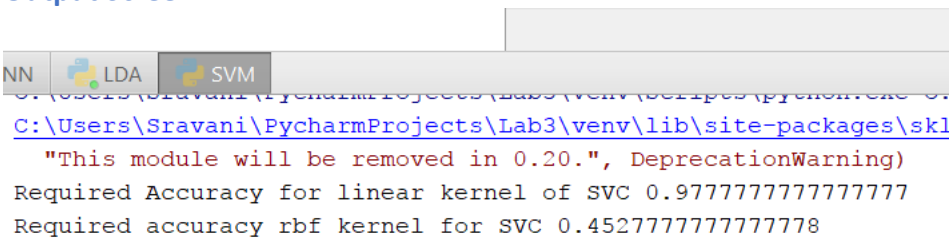
```
C:\Users\Sravani\PycharmProjects\Lab3\venv\Scripts\p
```

```
Required accuracy score is:  0.9666666666666667
```

Question2:
Source Code2:

```
from sklearn import datasets, metrics
from sklearn.cross_validation import train_test_split
from sklearn import svm
from sklearn.datasets import load_digits
C = 1.0
#loading datasets with digits information
D=load_digits()
Info=D.data
tag=D.target
info_train, info_test, tag_train, tag_test=train_test_split(Info, tag, test_size=0.2)
#linear
design = svm.SVC(kernel='linear')
design.fit(info_train, tag_train)
#prediction for linear
y_predlinear=design.predict(info_test)
#accuracy for linear
L=str(metrics.accuracy_score(tag_test, y_predlinear))
print("Required Accuracy for linear kernel of SVC " + L)
#RBF
design = svm.SVC(kernel='rbf')
design.fit(info_train, tag_train)
#predction function for rbf
y_predrbf=design.predict(info_test)
#accuracy for rbf
R=str(metrics.accuracy_score(tag_test, y_predrbf))
print("Required accuracy rbf kernel for SVC " + R)
```

Output Screen2:



Report:

Obtained accuracies for both linear and RBF(non linear) kernels is 0.977777777 and 0.45277777778 respectively. It is mostly based on the features if linear is having high then it is preferred comparative to RBF and vice versa. In my view Linear is having high so considered linear is best.

Question3:
Source Code3:

```
import nltk
nltk.download()
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.stem import WordNetLemmatizer
from nltk import FreqDist
from nltk import ngrams

#read the sample document
Doc = open('sample', 'r', encoding="utf-8")
txt = Doc.read();
lemma = nltk.WordNetLemmatizer()
wrds = word_tokenize(txt)
sents=sent_tokenize(txt)

#lemmatization
print("\n After Lemmatization the result is")
le = []
for word in wrds:
  L=lemma.lemmatize(word, pos='v')
  le.append(L)
print(le)

#BI-gram
print("\n Required Bigram are : \n")
wrds = word_tokenize(txt)
bi=[]
#give the esult with 2 words for the text
X = ngrams(wrds, 2)
for a in X:
    bi.append(a)
print(bi)
```

```
#frequent words finding
print("\n Required Freq in a Bigram are : \n")
freqD = nltk.FreqDist(bi)
rep_words = freqD.most_common()
#extracting top 5 words
top5_rep = freqD.most_common(5)
print(rep_words)
print("\n Required Top 5 Freq in a Bigram are : \n")
print(top5_rep)

#extracting the sentence with top 5 bigrams.
sentc_1 = sent_tokenize(txt)
rept_sentc1 = []
for senti1 in sentc_1:
    for word, wrds in bi:
        for ((p,q), l) in top5_rep:
            if (word, wrds == p,q):
                rept_sentc1.append(senti1)  # concatenation the sentenses.
print ("\n Sentences with top 5 Bigrams after concatenation are: ")
print(max(rept_sentc1, key=len))
```

Output Screen:

```
 After Lemmatization the result is
['``', 'Barbarous', ',', 'indeed', ',', 'be', 'my', 'master', '!', 'Here', 'I', 'have', 'serve', 'him', 'faithfully', 'for', 'years', ',', 'and', 'instead', 'of

 Required Bigram are :

[('``', 'Barbarous'), ('Barbarous', ','), (',', 'indeed'), ('indeed', ','), (',', 'is'), ('is', 'my'), ('my', 'master'), ('master', '!'), ('!', 'Here'), ('Here'

 Required Freq in a Bigram are :

[((',', 'and'), 6), (('in', 'the'), 6), (('.', 'The'), 5), (('the', 'boar'), 5), (('I', 'have'), 4), (('.', 'I'), 4), (('.', "'"), 4), (('the', 'monkey'), 4),

 Required Top 5 Freq in a Bigram are :

[((',', 'and'), 6), (('in', 'the'), 6), (('.', 'The'), 5), (('the', 'boar'), 5), (('I', 'have'), 4)]

  Required Sentences with top 5 Bigrams after concatenation are:
"Why the mother will be in a tremendous scare, and before your master and mistress know what to do, you must run after me and rescue the child and take it home
```

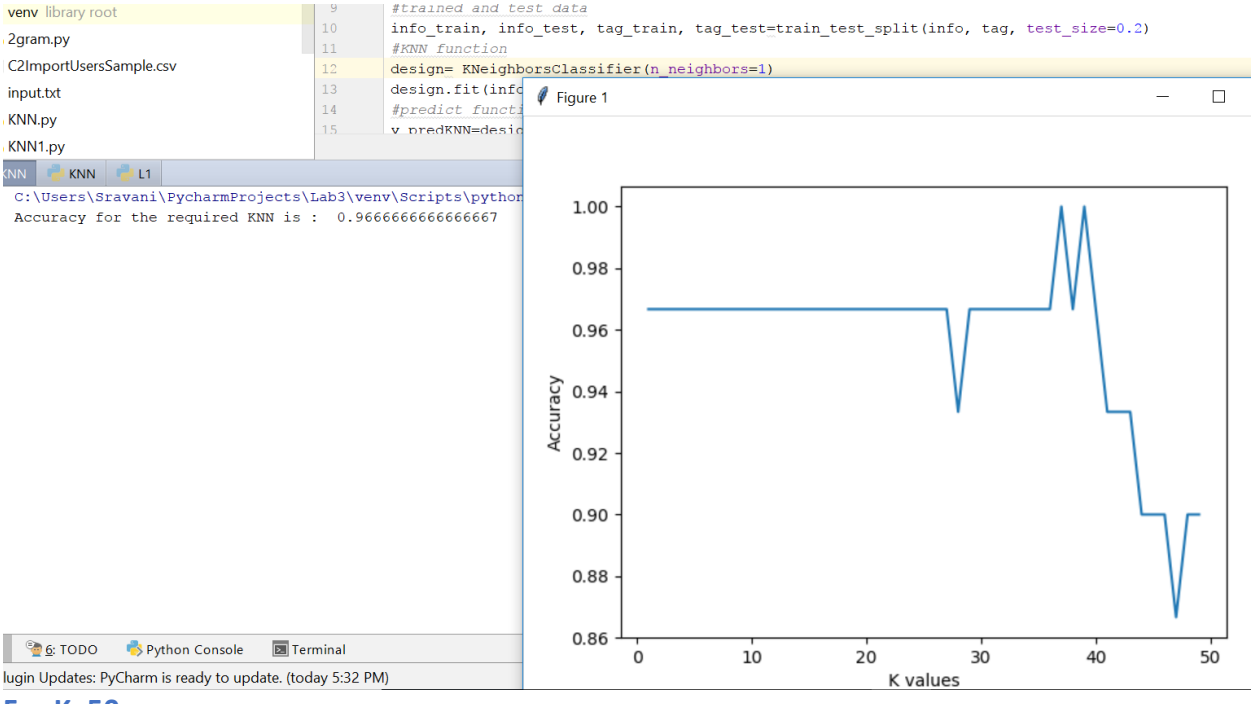
Question4:

Source Code4:

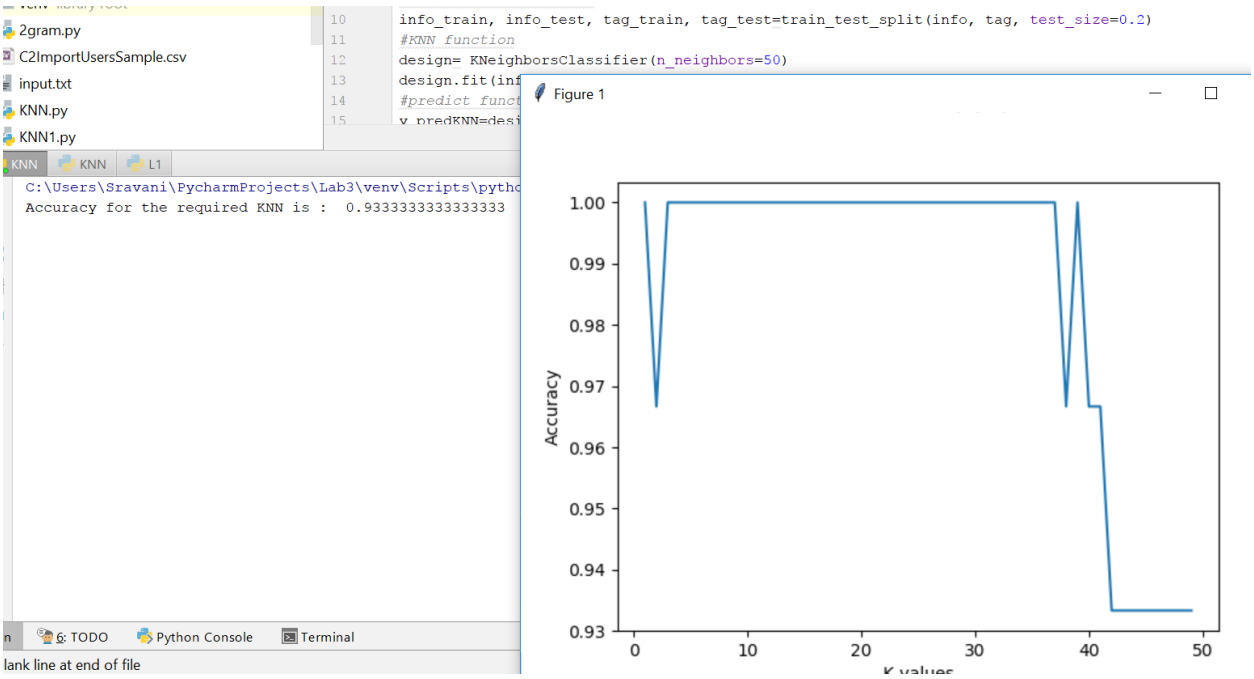
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets, metrics
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
#Load dataset
ID=datasets.load_iris()
info=ID.data
tag=ID.target
#trained and test data
info_train, info_test, tag_train, tag_test=train_test_split(info, tag, test_size=0.2)
#KNN function
design= KNeighborsClassifier(n_neighbors=5)
design.fit(info_train, tag_train)
#predict function
y_predKNN=design.predict(info_test)
#Accuracy
print("Accuracy for the required KNN is : ", metrics.accuracy_score(tag_test, y_predKNN))
k1_range=range(1, 50)
scrs=[]
for k in k1_range:
    KNN1=KNeighborsClassifier(n_neighbors=k)
    KNN1.fit(info_train, tag_train)
    y_predKNN=KNN1.predict(info_test)
    scrs.append(metrics.accuracy_score(tag_test, y_predKNN))
#ploting in graph
plt.plot(k1_range, scrs)
plt.xlabel("K values")
plt.ylabel("Accuracy")
plt.show()
```

Output Screen:

For K=1



For K=50



Code Snippet1:

Linear Discriminant Analysis and its Accuracy score with plotting it in graph.

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import accuracy_score
from sklearn import datasets, metrics
from sklearn.linear_model import LogisticRegression
# load dataset
irisDS= datasets.load_iris()
info = irisDS.data
tag = irisDS.target
info_train, info_test, tag_train, tag_test=train_test_split(info, tag, test_size=0.2,
random_state=36)
#LDA
design = LinearDiscriminantAnalysis()
#fitting data
Q=design.fit(info_train, tag_train).transform(info)
targ_nams = irisDS.target_names
# predicting data
y_predLDA=design.predict(info_test)
# accuracy
A=metrics.accuracy_score(y_predLDA, tag_test)
print(" \n Required accuracy score is: ", A)

plt.figure()
colours = ['orange', 'pink', 'violet']
for colour, k, targ_nam in zip(colours, [0, 1, 2], targ_nams):
    plt.scatter(Q[tag == k, 0], Q[tag == k, 1], alpha=.8, color=colour,
                label=targ_nam)
plt.legend(loc='best', shadow=False, scatterpoints=1)
```

```
plt.title('LDA for the choosen IRIS dataset')
plt.show()
```

For Logistic Regression:

Logistic regression and its accuracy score.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import datasets, metrics
from sklearn.linear_model import LogisticRegression
# loading dataset
irisDS= datasets.load_iris()
info = irisDS.data
tag = irisDS.target
info_train, info_test, tag_train, tag_test=train_test_split(info, tag, test_size=0.2,
random_state=36)
design = LogisticRegression()
#fitting data
Q=design.fit(info_train, tag_train)
targ_nams = irisDS.target_names
# predicting data
y_predLDA=design.predict(info_test)
# accuracy
LR=metrics.accuracy_score(y_predLDA, tag_test)
print(" \n Required accuracy score is: ", LR)
```

Report:

LDA in which more than 2 classes are taken where accuracy value is more compared to LR in which only 2 classes is taken.

Snippet2:

SVM kernal (Linear and RBF)

```
from sklearn import datasets, metrics
from sklearn.cross_validation import train_test_split
from sklearn import svm
from sklearn.datasets import load_digits
C = 1.0
#loading datasets with digits information
D=load_digits()
Info=D.data
tag=D.target
info_train, info_test, tag_train, tag_test=train_test_split(Info, tag, test_size=0.2)
#linear
design = svm.SVC(kernel='linear')
design.fit(info_train, tag_train)
#prediction for linear
tag_predlinear=design.predict(info_test)
#linear kernal with accuracy
L=str(metrics.accuracy_score(tag_test, tag_predlinear))
print("Required Accuracy for linear kernel of SVC " + L)
#RBF
design = svm.SVC(kernel='rbf')
design.fit(info_train, tag_train)
#predction function for rbf
tag_predrbf=design.predict(info_test)
#accuracy for rbf
```

```
R=str(metrics.accuracy_score(tag_test, tag_predrbf))
print("Required accuracy rbf kernel for SVC " + R)
```

Report:

Code Snippet3:

Using Natural Language toolkit worked out on the tasks lemmatization, bigram, frequency for the bigram words and top five bigram words from them and also extracting sentences in which top five words are and concatenating them.

```
import nltk
nltk.download()
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.stem import WordNetLemmatizer
from nltk import FreqDist
from nltk import ngrams

#read the sample document
Doc = open('sample', 'r', encoding="utf-8")
txt = Doc.read();
lemma = nltk.WordNetLemmatizer()
wrds = word_tokenize(txt)
sents=sent_tokenize(txt)

#lemmatization
print("\n After Lemmatization the result is")
le = []
for word in wrds:
  L=lemma.lemmatize(word, pos='v')
  le.append(L)
print(le)

#BI-gram
print("\n Required Bigram are : \n")
wrds = word_tokenize(txt)
bi=[]
#give the esult with 2 words for the text
X = ngrams(wrds, 2)
for a in X:
    bi.append(a)
print(bi)

#frequent words finding
print("\n Required Freq in a Bigram are : \n")
freqD = nltk.FreqDist(bi)
rep_words = freqD.most_common()
#extracting top 5 words
top5_rep = freqD.most_common(5)
print(rep_words)
print("\n Required Top 5 Freq in a Bigram are : \n")
print(top5_rep)

#extracting the sentence with top 5 bigrams.
sentc_1 = sent_tokenize(txt)
rept_sentc1 = []
for senti1 in sentc_1:
    for word, wrds in bi:
        for ((p,q), l) in top5_rep:
            if (word, wrds == p,q):
                rept_sentc1.append(senti1)  # concatenation the sentenses.
```

```
print ("\n Sentences with top 5 Bigrams after concatenation are: ")
print(max(rept_sentc1, key=len))
```

Code Snippet4:

KNN with accuracy score and variation between K values for 1 and 50.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets, metrics
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
#Load dataset
ID=datasets.load_iris()
info=ID.data
tag=ID.target
#trained and test data
info_train, info_test, tag_train, tag_test=train_test_split(info, tag, test_size=0.2)
#KNN function
design= KNeighborsClassifier(n_neighbors=1)
design.fit(info_train, tag_train)
#predict function
y_predKNN=design.predict(info_test)
#Accuracy
print("Accuracy for the required KNN is : ", metrics.accuracy_score(tag_test,
y_predKNN))
kn1_range1=range(1, 50)
ss1=[]
for q in kn1_range1:
    KNN1=KNeighborsClassifier(n_neighbors=q)
    KNN1.fit(info_train, tag_train)
    y_predKNN=KNN1.predict(info_test)
    ss1.append(metrics.accuracy_score(tag_test, y_predKNN))
#ploting in graph
plt.plot(kn1_range1, ss1)
plt.xlabel("K values")
plt.ylabel("Accuracy")
plt.show()
```

Summary for KNN:

- When we taken n_neighbors=1 the accuracy score value is more comparing to n_neighbors=50

- From the result my observation is accuracy score score are inversely proportional as shown in above output screens

Deployment:

Code is written in python and we used pycharm to run this and printed result in the python console.

Limitations:

No limitations for code snippets.