UNIVERSITY OF MISSOURI-KANSAS CITY

Python and Deep Learning

 # Lab 2 Assignment

Team Members:

Pragathi Thammaneni

Sridevi Mallipudi

Introduction:

The key objective of this assignment is to focus on all the regression/ prediction models by using python machine learning and execute the code successfully with provided data sets**.**

Objectives:

To code for the 4 questions by implementing the below concepts**.**

- Implemented scikit-learn
- Linear Discernment Analysis
- Kernel method/Support vector machines
- Analysis of measured accuracy
- Bi-grams
- Applied lemmatization
- Used Knn algorithm
- Prediction model analysis

Approaches /Methods:

Using Python 3.6, PyCharm (Community edition)

Workflow &Datasets/Parameters and Evaluation:

The below each question will follow different approaches to solve. Coding is done to perform the evaluation of each individual snippet to execute the datasets which are provided as the input parameters.

Question 1:

1. Pick any dataset from the dataset sheet in the class sheet and make one prediction model using your imagination with **Linear Discriminant Analysis***.
a. In the report provide convincible explanations about the difference between Logistic Regression and Linear Discriminant Analysis.
b. You can also pick dataset of your own.

*Logistic Regression is a classification algorithm traditionally limited to only two-class classification problems. If you have more than two classes, then the **Linear Discriminant Analysis** algorithm is the preferred linear classification technique.

Solution:

This snippet code provides the above implementation which is linear discriminant analysis and logistic regression. Required packages are downloaded and Iris dataset has been taken in to consideration for predicting the accuracy of each model. The model is trained with the train data and the test data is used to predict the model accuracy. Split ratio of 80%&20% over train and test is taken for building the model.

Below are the differences between the logistic regression and Linear Discriminant Analysis

- LDA and LR are two widely used multivariate statistical methods for data analysis with categorical outcome variables
- LDA makes a few assumptions such as the explanatory or predictor variables must be normally distributed
- LR does not need any underlying assumption made on the distribution of the data but it takes much longer time than LDA for computing.
- From the below code execution, we can say that the Accuracy is more in LDA than compared to LR

Data Set : Iris Data set

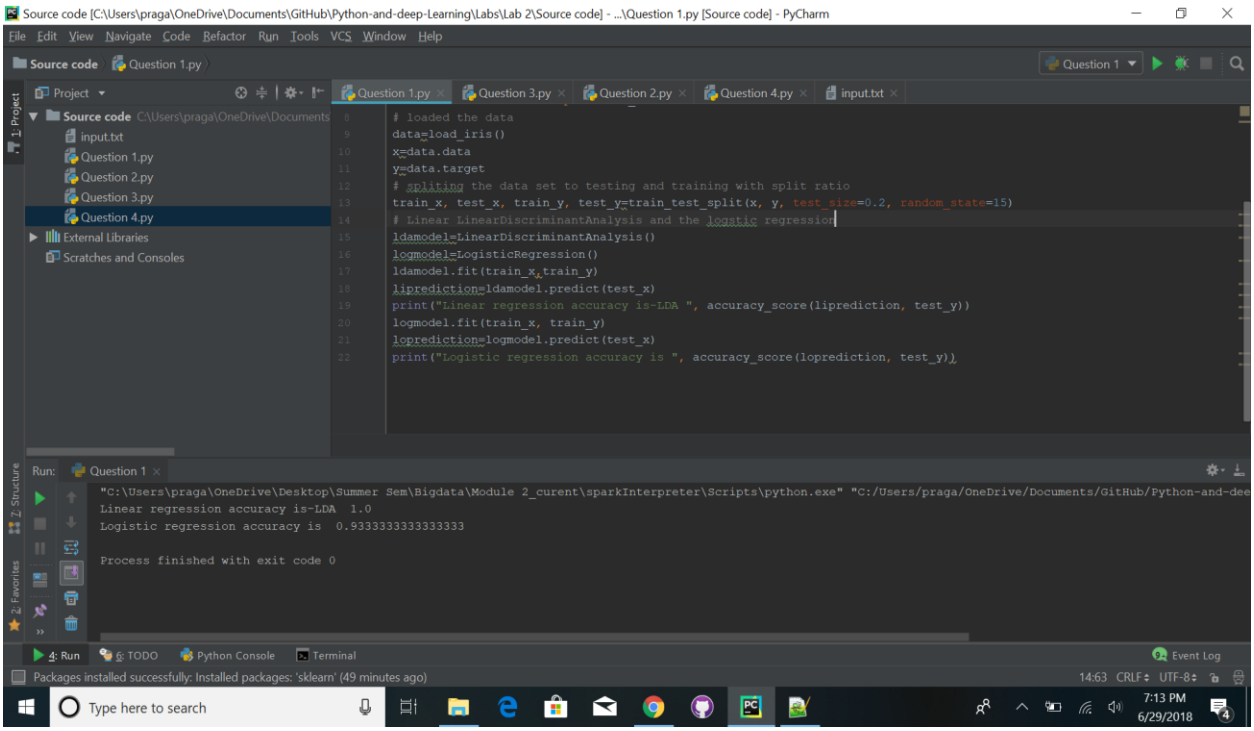
Code Snippet:

The below is the code for executing the above workflow

22 lines (22 sloc) | 1.05 KB — Raw | Blame | History

```
#e prediction model using  with Linear Discriminant Analysis*.
# imported the sklearn packages to perform the LinearDiscriminantAnalysis,LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
# loaded the data
data=load_iris()
x=data.data
y=data.target
# spliting the data set to testing and training with split ratio
train_x, test_x, train_y, test_y=train_test_split(x, y, test_size=0.2, random_state=15)
# Linear LinearDiscriminantAnalysis and the logstic regression
ldamodel=LinearDiscriminantAnalysis()
logmodel=LogisticRegression()
ldamodel.fit(train_x,train_y)
liprediction=ldamodel.predict(test_x)
print("Linear regression accuracy is-LDA ", accuracy_score(liprediction, test_y))
logmodel.fit(train_x, train_y)
loprediction=logmodel.predict(test_x)
print("Logistic regression accuracy is ", accuracy_score(loprediction, test_y))
```

Output Screenshot:



Question 2:

2. Implement Support Vector Machine classification,
a. Choose one of the datasets using the datasets features in the scikit-learn
b. Load the dataset
c. According to your dataset, split the data into 20% testing data, 80% training data (you can also use any other number)
d. Apply SVC with Linear kernel
e. Apply SVC with RBF kernel
f. Report the accuracy of the model on both models separately and report their differences if there are any
g. Report your view how can you increase the accuracy and which kernel is the best for your dataset and why

Solution:

This snippet code provides the above implementation of Support vector classification. Imported the required modules from the python packages. Iris dataset has been loaded to perform SVM. The data is splitted in to 20% testing and 80% training data. Then Applied both the Linear kernel and RBF kernel and reported the below screen shots for the predicted accuracy.

- Linear kernel is best compared to RBF kernel but it also depends on the data set and the data size. It varies mainly when the random state is changed. So it is difficult to figure out which is a better kernel, it mainly depends and varies accordingly to the data.

- **Data Set : Iris Data set**

Code Snippet:

The below is the code for executing the above workflow

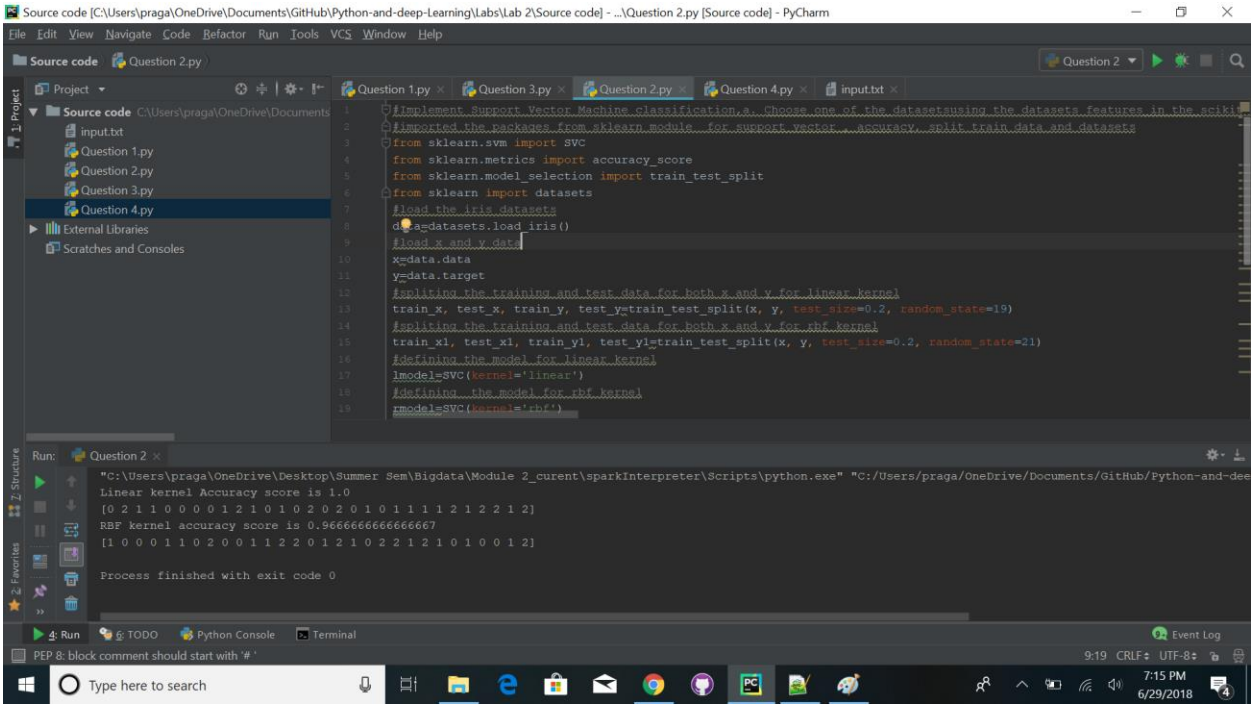
33 lines (33 sloc) | 1.89 KB

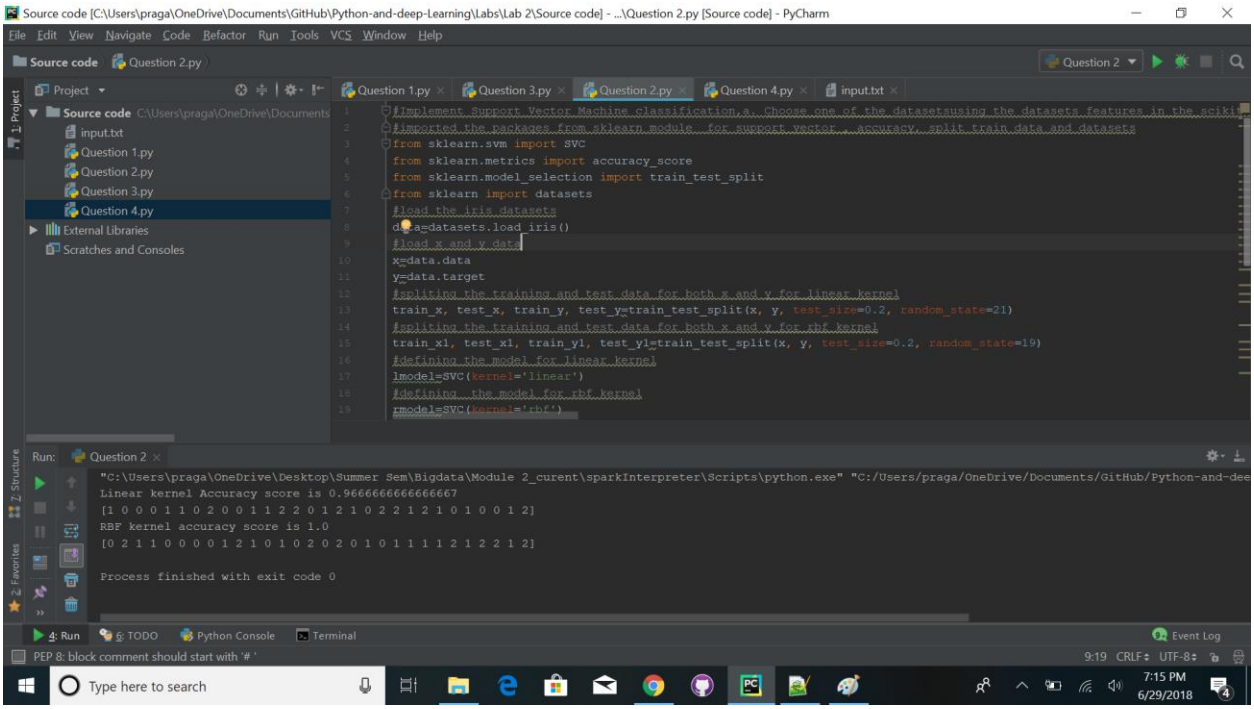
Raw Blame History

```
#Implement Support Vector Machine classification,a. Choose one of the datasetsusing the datasets features in the scikit-learnb. Load the da
#imported the packages from sklearn module  for support vector , accuracy, split train data and datasets
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn import datasets
#load the iris datasets
data=datasets.load_iris()
#load x and y data
x=data.data
y=data.target
#spliting the training and test data for both x and y for linear kernel
train_x, test_x, train_y, test_y=train_test_split(x, y, test_size=0.2, random_state=21)
#spliting the training and test data for both x and y for rbf kernel
train_x1, test_x1, train_y1, test_y1=train_test_split(x, y, test_size=0.2, random_state=19)
#defining the model for linear kernel
lmodel=SVC(kernel='linear')
#defining  the model for rbf kernel
rmodel=SVC(kernel='rbf')
#fitting the training data into linear kernel
lmodel.fit(train_x, train_y)
#predicting the test data using linear kernel
prediction=lmodel.predict(test_x)
#calculate the  accuracy score for linear kernel
print("Linear kernel Accuracy score is", accuracy_score(prediction, test_y))
print(prediction)
#fitting the  training data into rbc kernel
rmodel.fit(train_x1, train_y1)
#predicting the test data for rbc kernel
pred=lmodel.predict(test_x1)
#calculated theb  accuracy for rbc kernel
print("RBF kernel accuracy score is", accuracy_score(pred, test_y1))
print(pred)
```

Output Screenshot:

Variations in the accuracy depending up on the random state of each kernel.





Question 3:

3. Write a program
Take an Input file. Use the simple approach below to summarize a text file:
a. Read the file
b. Apply lemmatization on the words
c. Apply the bigram on the text
d. Calculate the word frequency (bi-gram frequency) of the words (bi-grams)
f. Choose top five bi-grams that have been repeated most
g. Go through the original text that you had in the file
h. Find all the sentences with those most repeated bi-grams
i. Extract those sentences and concatenate
j. Enjoy the summarization

Solution:

This snippet code provides the above implementation of lemmatization, bigram, top rated 5-word s and the concatenation of the sentences. Input file is loaded and below code is used for performing the above steps.

Data Set : A sample text file with the paragraph

Code Snippet:

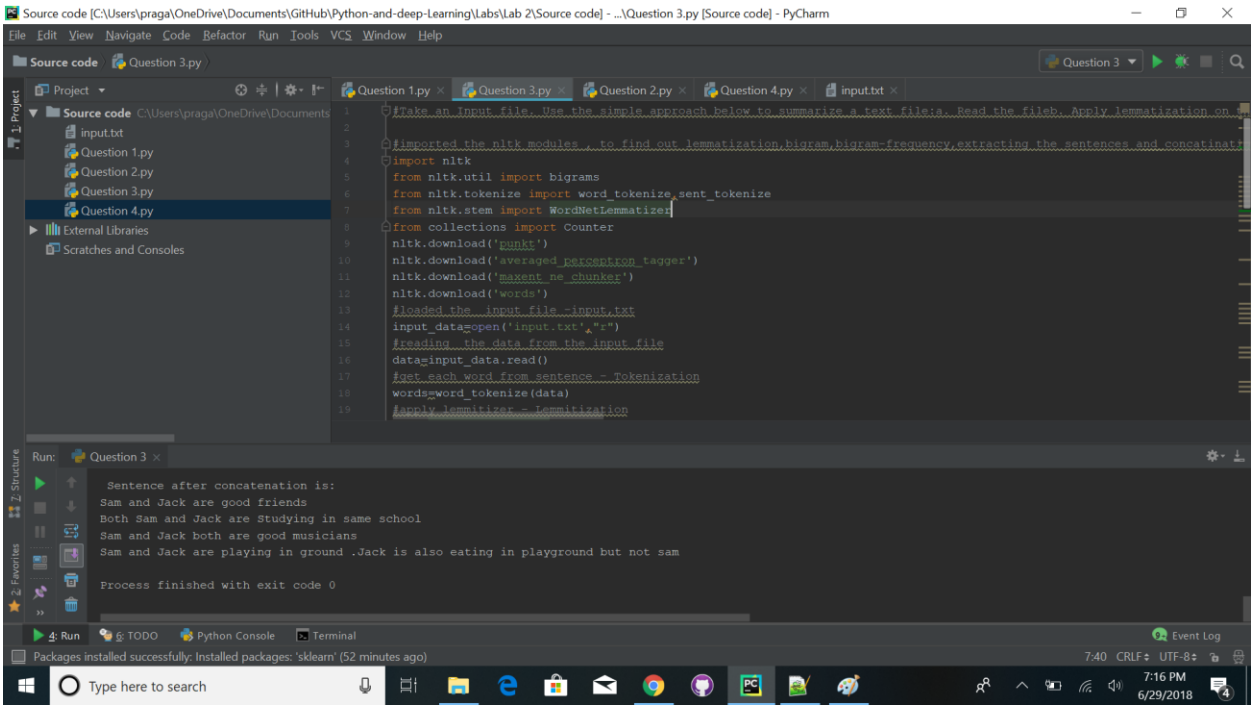
The below is the code for executing the above workflow

```
#Take an Input file. Use the simple approach below to summarize a text file:a. Read the fileb. Apply lemmatization on the wordsc. Apply the

#imported the nltk modules , to find out lemmatization,bigram,bigram-frequency,extracting the sentences and concatination
import nltk
from nltk.util import bigrams
from nltk.tokenize import word_tokenize,sent_tokenize
from nltk.stem import WordNetLemmatizer
from collections import Counter
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')
#loaded the  input file -input,txt
input_data=open('input.txt',"r")
#reading  the data from the input file
data=input_data.read()
#get each word from sentence - Tokenization
words=word_tokenize(data)
#apply lemmitizer - Lemmitization
lemm=WordNetLemmatizer()
list=[]
list1=[]
print("Lemmitizor:"+"\n")
#printing the lemmitizer  for each word
for k in words:
    print("Lemmatizor for word "+k+" is "+lemm.lemmatize(k))

print("\n")
print("Bigrams:"+"\n")
```

Output Screenshot:



Question 4:

4. Report your views on the k nearest neighbor algorithm when we change the K how it will affect the accuracy. Provide a good justification for the changes of the accuracy when we change the amount of K.
For example: compare the accuracy when K=1 and K is a big number like 50, why the accuracy will change

Solution:

This snippet code provides the above implementation KNN algorithm with different k values. When the k value increases, the model will better fit into the training data and get the accurate results while compared to the less k value. The K value has more effect on the test data So there will be variation in predicting the accuracy.

Data Set : Iris Data set

Code Snippet:

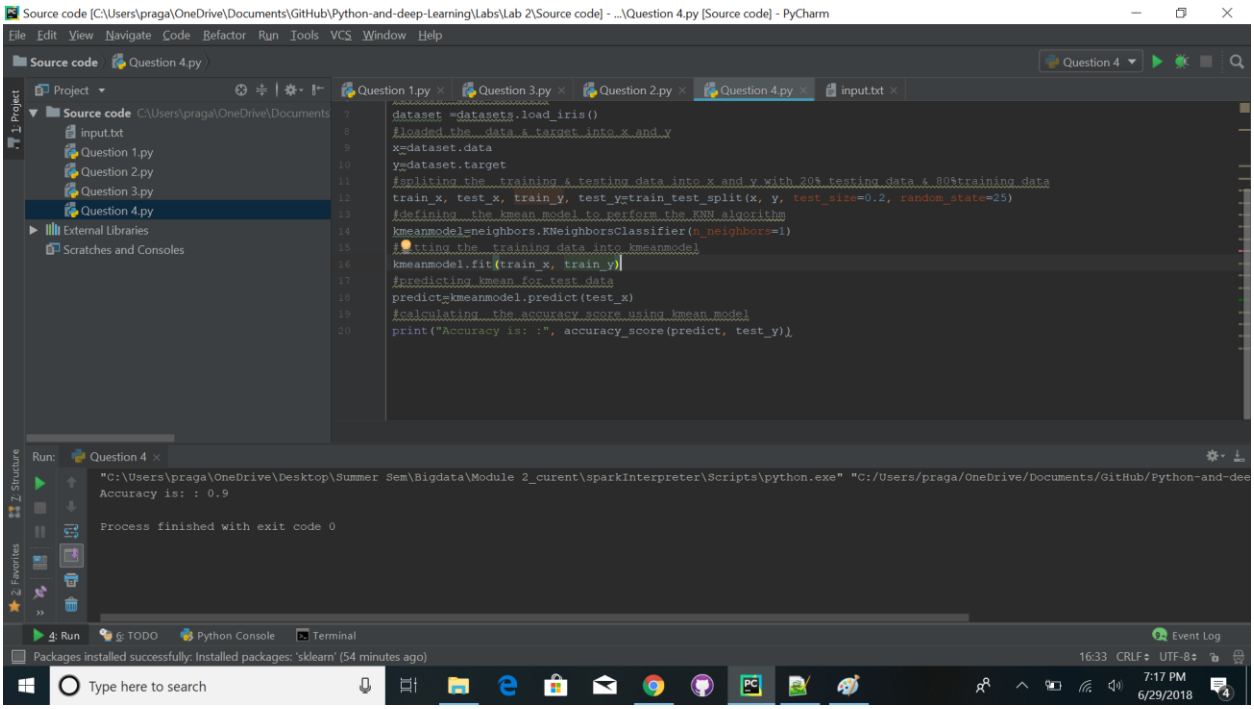
The below is the code for executing the above workflow

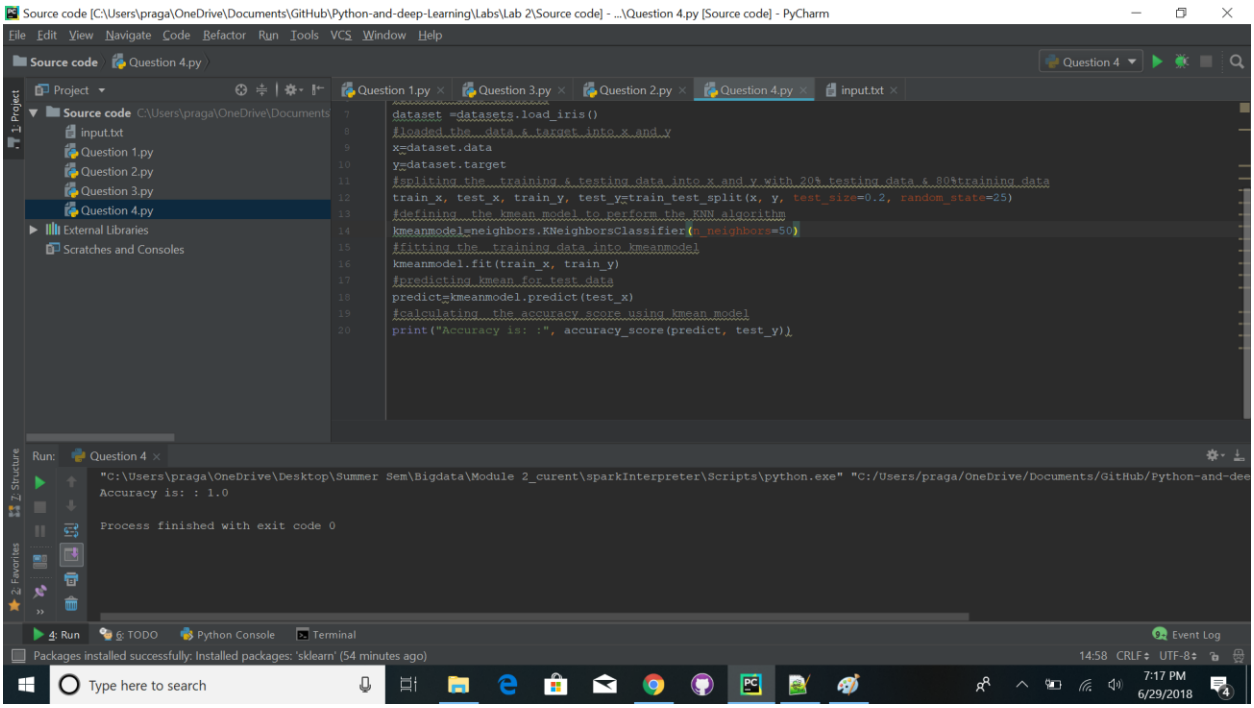
20 lines (20 sloc) | 941 Bytes

Raw | Blame | History

```python
# k nearest neighbor algorithm
#imported the  sklearn package for accuracy,metrics,datasets,split train & test data
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn import neighbors,datasets
#loaded  iris datasets
dataset =datasets.load_iris()
#loaded the  data & target into x and y
x=dataset.data
y=dataset.target
#spliting the  training & testing data into x and y with 20% testing data & 80%training data
train_x, test_x, train_y, test_y=train_test_split(x, y, test_size=0.2, random_state=25)
#defining  the kmean model to perform the KNN algorithm
kmeanmodel=neighbors.KNeighborsClassifier(n_neighbors=50)
#fitting the  training data into kmeanmodel
kmeanmodel.fit(train_x, train_y)
#predicting kmean for test data
predict=kmeanmodel.predict(test_x)
#calculating  the accuracy score using kmean model
print("Accuracy is: :", accuracy_score(predict, test_y))
```

Output Screenshot:





Conclusion:

As stated the above workflow with certain set of parameters is followed in solving the execution by implementing the core and basic concepts of the python programming.

Source code link https://github.com/PragathiThammaneni/Python-and-deep-Learning/tree/master/Lab%202/Source%20code

Video Link: https://youtu.be/OdlXO6z8gCE (made a short video as it should be less time)

Wiki Link:

https://github.com/PragathiThammaneni/Python-and-deep-Learning/wiki/Lab-2-Assignment