Deep Learning Lab 3

Spring 2018

Author:

Moe Almansoori

Objective:

Utilizing some of the most popular python scientific modules such as nltk, pandas, and sklearn to practice several scientific analyses of big data problems such as most of the regression types. Also implementing both of the machine learning types; the supervised and the unsupervised.

The objective for each task:

1- In this task a sample data of three classes is used to implement a supervised learning by splitting this data set to 40% for training and 60% of it for testing. Finally a graphical illustration is provided to show the grouping of these three classes.
   Shedding the light briefly on the difference between the Linear Regressio-LR and Linear Discriminant Analysis -LDA. The goal of LR is to find the best fitting and most parsimonious model to describe the relationship between the outcome (dependent or response variable) and a set of independent (predictor or explanatory) variables. The method is relatively robust, flexible and easily used, and it lends itself to a meaningful interpretation. In LR, unlike in the case of LDA, no assumptions are made regarding the distribution of the explanatory variables. On the other hand, LDA can be used to determine which variable discriminates between two or more classes, and to derive a classification model for predicting the group membership of new observations. LDA assumes the explanatory variables to be normally distributed with equal covariance matrices.
2- Here the sklearn python module is used to implement the Support Vector Machine classification-SVM with both the Linear Kernel and RBF Kernel on a dataset that been loaded from sklearn which is load_digits.
   It worth to mentioned that the accuracy is much higher in the case of the SVM with Linear Kernel than the case in of RBF Kernel.
3- In this task several of the popular applications of the natural language machine learning analysis are implemented on a text file that contains a short story ‘The Tale of Peter Rabbit’.
4- In the fourth and last task the K Nearest Neighbor-KNN algorithm is implemented over a wide variety of K value to find the impact of different values of K. As K lies btween 1 to 70, this model

gives a high accuracy results. However, the accuracy decays dramatically when K goes higher than 70.

Features:

The accomplished programs have been written in a simple programming structure and considering the logic sequence in results processing and proper transitioning between the steps. It is considered that when someone else reads the program to understand its commands line by line and or blocks functionality, by clearly separating the tasks and giving descriptive names for the variables and providing descriptive comments for the main parts of the code.

Configuration:

Similar working configuration had been used for all the Lab1 requirements, they are:

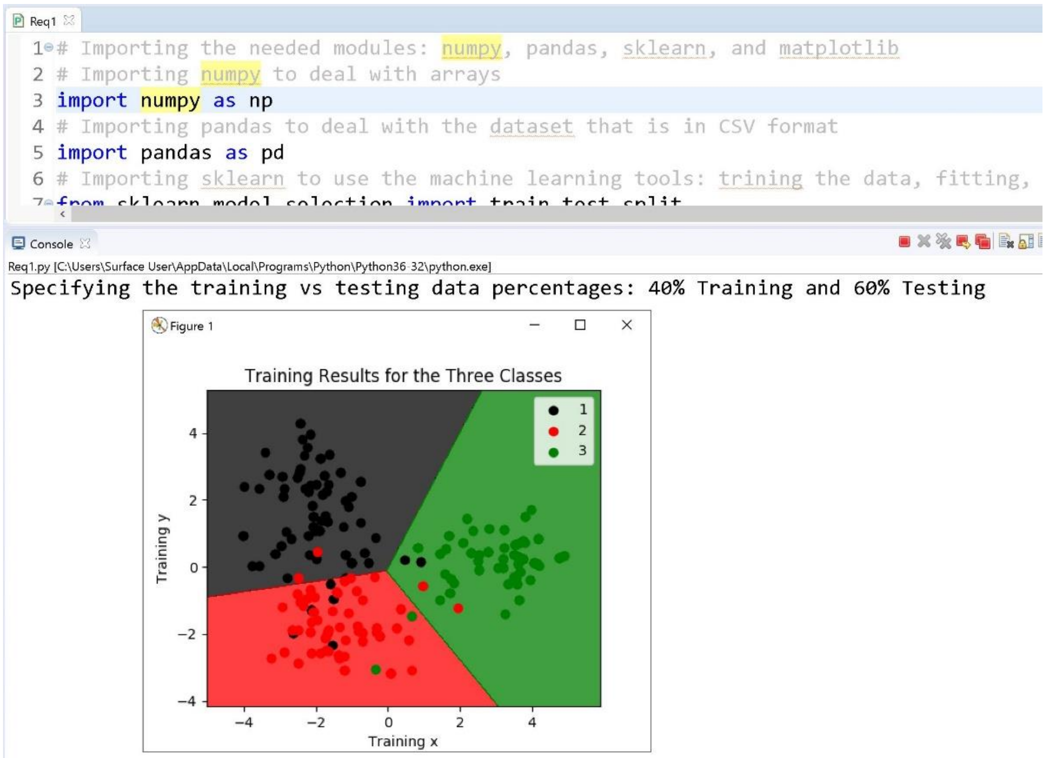
Python 3.6

OS: Widows 10

IDE: Eclipse Oxygen 2

Input/output (screenshots):

Requirement 1:



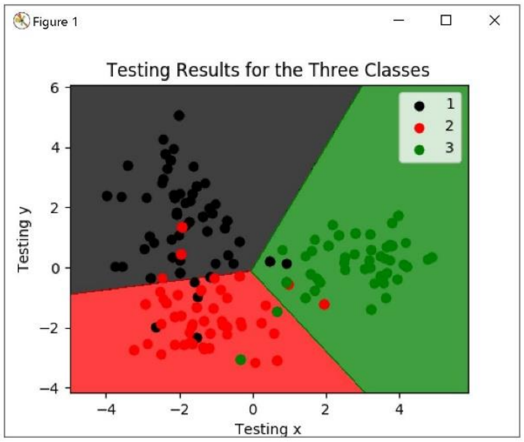
Req1

```python
# Importing the needed modules: numpy, pandas, sklearn, and matplotlib
# Importing numpy to deal with arrays
import numpy as np
# Importing pandas to deal with the dataset that is in CSV format
import pandas as pd
# Importing sklearn to use the machine learning tools: trining the data, fitting, i
from sklearn.model_selection import train_test_split
```

Console

Req1.py [C:\Users\Surface User\AppData\Local\Programs\Python\Python36-32\python.exe]

```
Specifying the training vs testing data percentages: 40% Training and 60% Testing
```



Equirement 2:

Req1

```python
# Importing the needed modules: numpy, pandas, sklearn, and matplotlib
# Importing numpy to deal with arrays
import numpy as np
# Importing pandas to deal with the dataset that is in CSV format
import pandas as pd
# Importing sklearn to use the machine learning tools: trining the data, fitting, imp
from sklearn.model_selection import train_test_split
```

Console

```
<terminated> Req2.py [C:\Users\Surface User\AppData\Local\Programs\Python\Python36-32\python.exe]
C:\Users\Surface User\AppData\Local\Programs\Python\Python36-32\lib\site-packages\sklearn
  "This module will be removed in 0.20.", DeprecationWarning)
Loading Digits dataset from sklearn module ...

Specifying the training vs testing data percentages: 40% Training and 60% Testing ...

Applying SVC with Linear Kernel ...
The accuracy score of the SVC with Linear Kernel =  0.97635605007

Applying the SVC with RBF kernel ...
The accuracy score of the SVC with RBF =  0.0890125173853
```

Requirement 3:

Req3

```python
# Importing the needed dependencies: re, collections, and nltk
# Importing sklearn to use the machine learning components
import re, collections
from nltk.tokenize import wordpunct_tokenize, sent_tokenize, word_tokenize
from nltk.tag import pos_tag
from nltk.stem import WordNetLemmatizer
```

Console

```
<terminated> Req3.py [C:\Users\Surface User\AppData\Local\Programs\Python\Python36-32\python.exe]

Applying Lemmatization on the words ...


Applying the bigram and calculating the words frequency ...
Words frequencies is :  Counter({('the', 'DT'): 47, ('and', 'CC'): 43, ('he', 'PRP'): 35, ('be', 'VB


Finding the top five bi-grams that has been repeated most ...
The top five frequent words are :  [(('the', 'DT'), 47), (('and', 'CC'), 43), (('he', 'PRP'), 35), (
['the', 'and', 'he', 'be', 'to']


Going through the original text and finding all the sentences with those most repeated bi-grams ...


Extracting the sentences and concatenating them ...
The concatenated sentences :  ['the tale of peter rabbit\nby beatrix potter\nthe most beloved story
```

Requirement 4:

Req4

```python
# Importing the needed dependencies: sklearn
# Importing sklearn to use the machine learning components
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets, metrics
from sklearn.cross_validation import train_test_split

# Loading the data
```

Console

<terminated> Req4.py [C:\Users\Surface User\AppData\Local\Programs\Python\Python36-32\python.exe]

```
Specifying the training vs testing data percentages: 20% Training and 80% Testing ...

For K = 1, the accuracy is : 0.933333333333
For K = 2, the accuracy is : 0.966666666667
For K = 5, the accuracy is : 0.966666666667
For K = 10, the accuracy is : 0.966666666667
For K = 40, the accuracy is : 1.0
For K = 50, the accuracy is : 0.966666666667
For K = 60, the accuracy is : 0.966666666667
For K = 70, the accuracy is : 0.9
For K = 80, the accuracy is : 0.666666666667
For K = 90, the accuracy is : 0.666666666667
For K = 100, the accuracy is : 0.666666666667
For K = 110, the accuracy is : 0.633333333333
```

Conclusion: It is very clear that the accurace maintain a high value as K is from 1 to around 70.
After K = 70, the accuracy is dropping dramatically

The implementation including code snippet

The code for all the four tasks is described in detail bellow

Requirement 1:

```
# Importing the needed modules: numpy, pandas,
sklearn, and matplotlib
# Importing numpy to deal with arrays
import numpy as np
# Importing pandas to deal with the dataset that is
in CSV format
import pandas as pd
# Importing sklearn to use the machine learning
tools: trining the data, fitting, implementing LDA
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis as LDA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
# Importing matplotlib to plot the regression
analysis results
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
# Loading the data
LoadedData = pd.read_csv('dataset.csv')
# The data
x = LoadedData.iloc[:, 0:9].values
# The three groups members 1's, 2's, and 3's which is
the last column in the data
y = LoadedData.iloc[:, 9].values
# Specifying the training vs testing data percentages
```

```
print('Specifying the training vs testing data
percentages: 40% Training and 60% Testing')
TrainingX, TestingX, TrainingY, TestingY =
train_test_split(x, y, test_size = 0.4, random_state
= 0)
# Fitting the training data
sc = StandardScaler()
TrainingX = sc.fit_transform(TrainingX)
TestingX = sc.transform(TestingX)
# Applying LDA
LDAClass = LDA(n_components = 2)
TrainingX = LDAClass.fit_transform(TrainingX,
TrainingY)
TestingX = LDAClass.transform(TestingX)
# Testing the trained results
RegClass = LogisticRegression(random_state = 0)
RegClass.fit(TrainingX, TrainingY)
TestingResult = RegClass.predict(TestingX)
cm = confusion_matrix(TestingY, TestingResult)

# Plotting the Training results
xCoordinates, yCoordinates = TrainingX, TrainingY
HorizStart, HorizEnd = np.meshgrid(np.arange(start =
xCoordinates[:, 0].min() - 1, stop = xCoordinates[:,
0].max() + 1, step = 0.005),
                     np.arange(start =
xCoordinates[:, 1].min() - 1, stop = xCoordinates[:,
1].max() + 1, step = 0.005))
plt.contourf(HorizStart, HorizEnd,
RegClass.predict(np.array([HorizStart.ravel(),
HorizEnd.ravel()]).T).reshape(HorizStart.shape),
             alpha = 0.75, cmap =
ListedColormap(('black', 'red', 'green')))
plt.xlim(HorizStart.min(), HorizStart.max())
```

```
plt.ylim(HorizEnd.min(), HorizEnd.max())
for a, b in enumerate(np.unique(yCoordinates)):
    plt.scatter(xCoordinates[yCoordinates == b, 0],
xCoordinates[yCoordinates == b, 1],
                c = ListedColormap(('black', 'red',
'green'))(a), label = b)
plt.title('Training Results for the Three Classes')
plt.xlabel('Training x')
plt.ylabel('Training y')
plt.legend()
plt.show()
#Plotting the Testing results
xCoordinates, yCoordinates = TestingX, TestingY
HorizStart, HorizEnd = np.meshgrid(np.arange(start =
xCoordinates[:, 0].min() - 1, stop = xCoordinates[:,
0].max() + 1, step = 0.005),
                     np.arange(start =
xCoordinates[:, 1].min() - 1, stop = xCoordinates[:,
1].max() + 1, step = 0.005))
plt.contourf(HorizStart, HorizEnd,
RegClass.predict(np.array([HorizStart.ravel(),
HorizEnd.ravel()]).T).reshape(HorizStart.shape),
             alpha = 0.75, cmap =
ListedColormap(('black', 'red', 'green')))
plt.xlim(HorizStart.min(), HorizStart.max())
plt.ylim(HorizEnd.min(), HorizEnd.max())
for a, b in enumerate(np.unique(yCoordinates)):
    plt.scatter(xCoordinates[yCoordinates == b, 0],
xCoordinates[yCoordinates == b, 1],
                c = ListedColormap(('black', 'red',
'green'))(a), label = b)
plt.title('Testing Results for the Three Classes')
plt.xlabel('Testing x')
plt.ylabel('Testing y')
```

```
plt.legend()
plt.show()
```

Requirement 2:

```
# Importing the needed dependencies: sklearn module
# Importing sklearn to use the machine learning
components
from sklearn.cross_validation import train_test_split
from sklearn import svm, datasets, metrics
# Loading digits dataset from sklearn module
print('Loading Digits dataset from sklearn module
...')
DigitsLoaded = datasets.load_digits()
Entry = DigitsLoaded.data
Response = DigitsLoaded.target
# Specifying the training vs testing data percentages
print('\nSpecifying the training vs testing data
percentages: 40% Training and 60% Testing ...')
TraningX, TestingX, TrainingY, TestingY =
train_test_split(Entry, Response, test_size = 0.4)
# Applying CVS with Linear Kernel
print('\nApplying SVC with Linear Kernel ...')
SVCLinearKernel = svm.SVC(kernel = 'linear', C = 1,
gamma = 1)
# Fitting the training data
SVCLinearKernel.fit(TraningX,TrainingY)
# Testing the results
TestingResults = SVCLinearKernel.predict(TestingX)
```

```
print ('The accuracy score of the SVC with Linear
Kernel =
',metrics.accuracy_score(TestingY,TestingResults))
# Applying the SVC with RBF kernel
print('\nApplying the SVC with RBF kernel ...')
SVCRBF =svm.SVC(kernel = 'rbf', C = 1, gamma = 1)
# Fitting the training data
SVCRBF.fit(TraningX,TrainingY)
# Testing the results
TestingResults = SVCRBF.predict(TestingX)
#checking the accuracy of the model with digits
datasets by rbf kernel
print ('The accuracy score of the SVC with RBF =
',metrics.accuracy_score(TestingY,TestingResults))
```

Requirement 3:

```
# Importing the needed dependencies: re, collections,
and nltk
# Importing sklearn to use the machine learning
components
import re, collections
from nltk.tokenize import wordpunct_tokenize,
sent_tokenize, word_tokenize
from nltk.tag import pos_tag
from nltk.stem import WordNetLemmatizer


def tokens(text):
    return re.findall('[a-z]+', text.lower())


WordsList=[]
```

```
# Reading the text file: TheTaleofPeterRabbit.txt
print('\n\nReading the text file:
TheTaleofPeterRabbit.txt ...')
TextFile = open('TheTaleofPeterRabbit.txt').read()
print('Converting the text into words (Tokenizing it)
...')
TextFileAsWords = tokens(TextFile)

# Applying Lemmatization on the words:
print('\n\nApplying Lemmatization on the words ...')
LemmatizedWords = WordNetLemmatizer()
for i in TextFileAsWords:
    WordsList.append(LemmatizedWords.lemmatize(i, pos
= 'v'))

# Applying the bigram and calculating the words
frequency
print('\n\nApplying the bigram and calculating the
words frequency ...')
WordsClass = []
ClassifiedWordsList = pos_tag(WordsList)
frequency = 0
TempIteration = 0
FrequentWordHolder = []
while frequency<len(ClassifiedWordsList):
    if ClassifiedWordsList[frequency][1] != 'VB' or
ClassifiedWordsList[frequency][1] != 'VBD' or
ClassifiedWordsList[frequency][1] != 'VBG' or
ClassifiedWordsList[frequency][1] != 'VBN' or
ClassifiedWordsList[frequency][1] != 'VBP' or
ClassifiedWordsList[frequency][1] != 'VBZ':

WordsClass.append(ClassifiedWordsList[frequency])
```

```
    frequency +=1

Frequencies = collections.Counter(WordsClass)
print('Words frequencies is : ', Frequencies)

# Choosing the top five bi-grams that has been
repeated most:
print('\n\nFinding the top five bi-grams that has
been repeated most ...')
TopFiveFrequent = Frequencies.most_common(5)
print('The top five frequent words are : ',
TopFiveFrequent)

while TempIteration<len(TopFiveFrequent):

FrequentWordHolder.append(TopFiveFrequent[TempIterati
on][0][0])
    TempIteration +=1
print(FrequentWordHolder)

# Going through the original text and finding all the
sentences with those most repeated bi-grams:
print('\n\nGoing through the original text and
finding all the sentences with those most repeated
bi-grams ...')
TextFile2 =
open('TheTaleofPeterRabbit.txt').read().lower()
TextFile2Sentences = sent_tokenize(TextFile2)
SentencesList = []
# Extracting the sentences and concatenating them
print('\n\nExtracting the sentences and concatenating
them ...')
for i in TextFile2Sentences:
    SentencesAsWords = word_tokenize(i)
```

```
    for j in SentencesAsWords:
        if j in FrequentWordHolder:
            SentencesList.append(i)
        break
print ('The concatenated sentences : ',
SentencesList)
```

Requirement 4:

```
# Importing the needed dependencies: sklearn
# Importing sklearn to use the machine learning
components
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets, metrics
from sklearn.cross_validation import train_test_split

# Loading the data
print('\n\nLoading the iris dataset ...')
irisdataset = datasets.load_iris()
Entries = irisdataset.data
Responses = irisdataset.target
# Specifying the training vs testing data percentages
print('\n\nSpecifying the training vs testing data
percentages: 20% Training and 80% Testing ...')
TrainingX, TestingX, TrainingY, TestingY =
train_test_split(Entries, Responses, test_size = 0.2)

# Fitting the training data
K = 1
KNNTemp = KNeighborsClassifier(n_neighbors = K)
KNNTemp.fit(TrainingX,TrainingY)
# Testing the results
Testingresult = KNNTemp.predict(TestingX)
```

```
print('\nFor K = ' + str(K) + ', the accuracy is : '
+
str(metrics.accuracy_score(TestingY,Testingresult)))



K = 2
KNNTemp = KNeighborsClassifier(n_neighbors = K)
KNNTemp.fit(TrainingX,TrainingY)
# Testing the results
Testingresult = KNNTemp.predict(TestingX)
print('For K = ' + str(K) + ', the accuracy is : ' +
str(metrics.accuracy_score(TestingY,Testingresult)))


K = 5
KNNTemp = KNeighborsClassifier(n_neighbors = K)
KNNTemp.fit(TrainingX,TrainingY)
# Testing the results
Testingresult = KNNTemp.predict(TestingX)
print('For K = ' + str(K) + ', the accuracy is : ' +
str(metrics.accuracy_score(TestingY,Testingresult)))



K = 10
KNNTemp = KNeighborsClassifier(n_neighbors = K)
KNNTemp.fit(TrainingX,TrainingY)
# Testing the results
Testingresult = KNNTemp.predict(TestingX)
print('For K = ' + str(K) + ', the accuracy is : ' +
str(metrics.accuracy_score(TestingY,Testingresult)))
```

```python
K = 40
KNNTemp = KNeighborsClassifier(n_neighbors = K)
KNNTemp.fit(TrainingX,TrainingY)
# Testing the results
Testingresult = KNNTemp.predict(TestingX)
print('For K = ' + str(K) + ', the accuracy is : ' +
str(metrics.accuracy_score(TestingY,Testingresult)))
```

```python
K = 50
KNNTemp = KNeighborsClassifier(n_neighbors = K)
KNNTemp.fit(TrainingX,TrainingY)
# Testing the results
Testingresult = KNNTemp.predict(TestingX)
print('For K = ' + str(K) + ', the accuracy is : ' +
str(metrics.accuracy_score(TestingY,Testingresult)))
```

```python
K = 60
KNNTemp = KNeighborsClassifier(n_neighbors = K)
KNNTemp.fit(TrainingX,TrainingY)
# Testing the results
Testingresult = KNNTemp.predict(TestingX)
print('For K = ' + str(K) + ', the accuracy is : ' +
str(metrics.accuracy_score(TestingY,Testingresult)))
```

```python
K = 70
```

```
KNNTemp = KNeighborsClassifier(n_neighbors = K)
KNNTemp.fit(TrainingX,TrainingY)
# Testing the results
Testingresult = KNNTemp.predict(TestingX)
print('For K = ' + str(K) + ', the accuracy is : ' +
str(metrics.accuracy_score(TestingY,Testingresult)))




K = 80
KNNTemp = KNeighborsClassifier(n_neighbors = K)
KNNTemp.fit(TrainingX,TrainingY)
# Testing the results
Testingresult = KNNTemp.predict(TestingX)
print('For K = ' + str(K) + ', the accuracy is : ' +
str(metrics.accuracy_score(TestingY,Testingresult)))



K = 90
KNNTemp = KNeighborsClassifier(n_neighbors = K)
KNNTemp.fit(TrainingX,TrainingY)
# Testing the results
Testingresult = KNNTemp.predict(TestingX)
print('For K = ' + str(K) + ', the accuracy is : ' +
str(metrics.accuracy_score(TestingY,Testingresult)))


K = 100
KNNTemp = KNeighborsClassifier(n_neighbors = K)
KNNTemp.fit(TrainingX,TrainingY)
# Testing the results
Testingresult = KNNTemp.predict(TestingX)
```

```python
print('For K = ' + str(K) + ', the accuracy is : ' +
str(metrics.accuracy_score(TestingY,Testingresult)))



K = 110
KNNTemp = KNeighborsClassifier(n_neighbors = K)
KNNTemp.fit(TrainingX,TrainingY)
# Testing the results
Testingresult = KNNTemp.predict(TestingX)
print('For K = ' + str(K) + ', the accuracy is : ' +
str(metrics.accuracy_score(TestingY,Testingresult)))


print('\n\n Conclusion: It is very clear that the
accurace maintain a high value as K is from 1 to
around 70.')
print('After K = 70, the accuracy is dropping
dramatically')
```

Explain about the deployment

It started with understanding the problem and figuring out what a good way to provide the required results. Then writing down a pseudo code and what is the input type and format. I was trying to avoid using any unnecessary programming structures and seek the simplicity. The startup program can be containing unnecessary loops or variables, but with several testing and polishing the code, the programs because more effective and easy to follow and understand.

Limitation

The programs that been developed for this lab purpose still need more improvements to serve in real life environment. For example, the contacts program can be improved more by including changing the name and adding new contacts. Other thing is these programs does not save the results permanently such as in a file.

References

- https://www.stat-d.si/mz/mz1.1/pohar.pdf
- https://support.spatialkey.com/spatialkey-sample-csv-data/
- https://stackoverflow.com/questions/40529848/python-beautifulsoup-how-to-write-the-output-to-html-file