NEURAL NETWORKS & DEEP LEARNING: ICP1

Name: Reshma Maddala

ID: 700740808

1. Implement Naïve Bayes method using scikit-learn library Use dataset available with name glass Use train_test_split to create training and testing part Evaluate the model on test part using score and classification_report(y_true,y_pred)

```
import pandas as pd
import numpy as np
#1.Implement Naïve Bayes method using scikit-learn library
# Use dataset available with name glass
# Use train_test_split to create training and testing part
# Evaluate the model on test part using score and
# classification_report(y_true, y_pred)
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, accuracy_score
df = pd.read_csv('C:/Users/reshm/Downloads/NNDL/glass.csv')
x_train = df.drop("Type", axis=1)
y_train = df['Type']
x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.2, random_state=0)
# Train the model using the training sets
gnb = GaussianNB()
gnb.fit(x_train, y_train)
y_pred = gnb.predict(x_test)
# Classification report
print("Classification Report: \n", classification_report(y_test, y_pred))
print("Naive Bayes accuracy is: ",  (accuracy_score(y_test, y_pred))*100)
```

In [1]:

```
import pandas as pd
import numpy as np
#1.Implement Naïve Bayes method using scikit-learn library
# Use dataset available with name glass
# Use train_test_split to create training and testing part
# Evaluate the model on test part using score and
# classification_report(y_true, y_pred)


from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, accuracy_score


df = pd.read_csv('C:/Users/reshm/Downloads/NNDL/glass.csv')

x_train = df.drop("Type", axis=1)
y_train = df['Type']


x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.2, random_state=0)

# Train the model using the training sets
gnb = GaussianNB()
gnb.fit(x_train, y_train)


y_pred = gnb.predict(x_test)
# Classification report
print("Classification Report: \n", classification_report(y_test, y_pred))
#Accuracy
print("Naive Bayes accuracy is: ",  (accuracy_score(y_test, y_pred))*100)
```

```
Classification Report:
              precision    recall  f1-score   support

           1       0.19      0.44      0.27         9
           2       0.33      0.16      0.21        19
           3       0.33      0.20      0.25         5
           5       0.00      0.00      0.00         2
           6       0.67      1.00      0.80         2
           7       1.00      1.00      1.00         6

    accuracy                           0.37        43
   macro avg       0.42      0.47      0.42        43
weighted avg       0.40      0.37      0.36        43

Naive Bayes accuracy is:  37.2093023255814
```

In the first problem, I have implemented naive bayes method using scikit learn library using the glass dataset provided. Used train_test_split to create training and testing part. By using Naïve bayes classifier , and by using 0.2 as test_size. Then I evaluated the part On test part using score. I got 37.2% accuracy.

2. Implement linear SVM method using scikit-learn Use the same dataset above Use train_test_split to create training and testing part Evaluate the model on test part using score and classification_report(y_true,y_pred). Which algorithm you got better accuracy? Can you justify why?

#Implement linear SVM method using scikit-learn

Use the same dataset above

Use train_test_split to create training and testing part

Evaluate the model on test part using score and

classification_report(y_true, y_pred)

import pandas as pd

```
from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import classification_report, accuracy_score

df2 = pd.read_csv('C:/Users/reshm/Downloads/NNDL/glass.csv')

x_train = df2.drop("Type", axis=1)

y_train = df2['Type']

# splitting train and test data using train_test_split

x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.2, random_state=0)

# Train the model using the training sets

svc = SVC()

svc.fit(x_train, y_train)

y_pred = svc.predict(x_test)

# Classification report

print("Classification Report: \n", classification_report(y_test, y_pred,zero_division = 0))

#Accuracy

print("SVM accuracy is: ", accuracy_score(y_test, y_pred)*100)
```

In the Second problem, I have implemented SVM method using scikit learn library using the glass dataset provided. Used train_test_split to create training and testing part. By using SVM classifier , and by using 0.2 as test_size. Then I evaluated the part On test part using score. I got 20.9% accuracy.

In [2]:

```
#2.Implement linear SVM method using scikit-learn
# Use the same dataset above
# Use train_test_split to create training and testing part
# Evaluate the model on test part using score and
# classification_report(y_true, y_pred)

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score


df2 = pd.read_csv('C:/Users/reshm/Downloads/NNDL/glass.csv')

x_train = df2.drop("Type", axis=1)
y_train = df2['Type']
# splitting train and test data using train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.2, random_state=0)

# Train the model using the training sets
svc = SVC()
svc.fit(x_train, y_train)
y_pred = svc.predict(x_test)
# Classification report
print("Classification Report: \n", classification_report(y_test, y_pred,zero_division = 0))
#Accuracy
print("SVM accuracy is: ", accuracy_score(y_test, y_pred)*100)
```

```
Classification Report:
               precision    recall  f1-score   support

           1       0.21      1.00      0.35         9
           2       0.00      0.00      0.00        19
           3       0.00      0.00      0.00         5
           5       0.00      0.00      0.00         2
           6       0.00      0.00      0.00         2
           7       0.00      0.00      0.00         6

    accuracy                           0.21        43
   macro avg       0.03      0.17      0.06        43
weighted avg       0.04      0.21      0.07        43

SVM accuracy is:  20.930232558139537
```

Here I got better accuracy with NB than SVM, as we tested the model with score parameter and NB is very fast and SVM works better with non-linear data and multi-dimensional. But we can't say which classifier works better than which classifier comparatively. As it depends on the data we take and the parameter we have taken for the testing part, but in here we used same data and same parameter still NB got higher accuracy than SVM. It might be that the data is linear.

3. Implement Linear Regression using scikit-learn a) Import the given "Salary_Data.csv" b) Split the data in train_test partitions, such that 1/3 of the data is reserved as test subset. c) Train and predict the model. d) Calculate the mean_squared error. e) Visualize both train and test data using scatter plot.

Implement Linear Regression using scikit-learn

Import the given "Salary_Data.csv"

df3 = pd.read_csv('C:/Users/reshm/Downloads/NNDL/Salary_Data.csv')

df3.info()

df3.head()

d1 = df3.iloc[:, :-1].values #excluding last column years of experience column

```
d2 = df3.iloc[:, 1].values     #only salary column
#  Split the data in train_test partitions, such that 1/3 of the data is reserved as test subset.
from sklearn.model_selection import train_test_split
d1_train, d1_test, d2_train, d2_test = train_test_split(d1, d2, test_size=1/3, random_state=0)
#  Train and predict the model.
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(d1_train, d2_train)
d2_Pred = reg.predict(d1_test)
d2_Pred
#  Calculate the mean_squared error
S_error = (d2_Pred - d2_test) ** 2
Sum_Serror = np.sum(S_error)
mean_squared_error = Sum_Serror / d2_test.size
mean_squared_error
#  Visualize both train and test data using scatter plot.
import matplotlib.pyplot as plt
# Training Data set
plt.scatter(d1_train, d2_train)
plt.plot(d1_train, reg.predict(d1_train))
plt.title('Training Set')
plt.show()

# Testing Data set
plt.scatter(d1_test, d2_test)
plt.plot(d1_test, reg.predict(d1_test))
plt.title('Testing Set')
```

```
plt.show()
```

In the third problem, I have implemeted Linear Regression using scikit-learn. Then I have Split the data in train_test partitions, such that 1/3 of the data is reserved as test subset. Later the model is trained and predicted. Then I have calculated the mean square error and then have shown the scatter plot through visualization.

In [3]:
```
# 3.Implement Linear Regression using scikit-learn
# Import the given "Salary_Data.csv"
df3 = pd.read_csv('C:/Users/reshm/Downloads/NNDL/Salary_Data.csv')
df3.info()
df3.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     float64
dtypes: float64(2)
memory usage: 608.0 bytes
```

Out[3]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

In [4]:
```
d1 = df3.iloc[:, :-1].values   #excluding last column  years of experience column
d2 = df3.iloc[:, 1].values     #only salary column
```

In [5]:
```
#  Split the data in train_test partitions, such that 1/3 of the data is reserved as test subset.
from sklearn.model_selection import train_test_split
d1_train, d1_test, d2_train, d2_test = train_test_split(d1, d2, test_size=1/3, random_state=0)
```

In [6]:
```
#  Train and predict the model.
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(d1_train, d2_train)
d2_Pred = reg.predict(d1_test)
d2_Pred
```

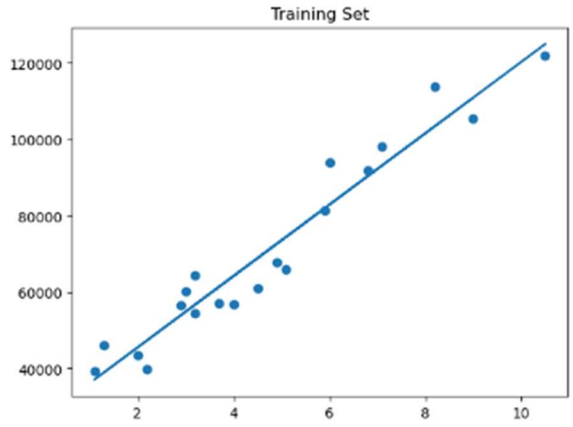
Out[6]:
```
array([ 40835.10590871, 123079.39940819,  65134.55626083,  63265.36777221,
       115602.64545369, 108125.8914992 , 116537.23969801,  64199.96201652,
        76349.68719258, 100649.1375447 ])
```

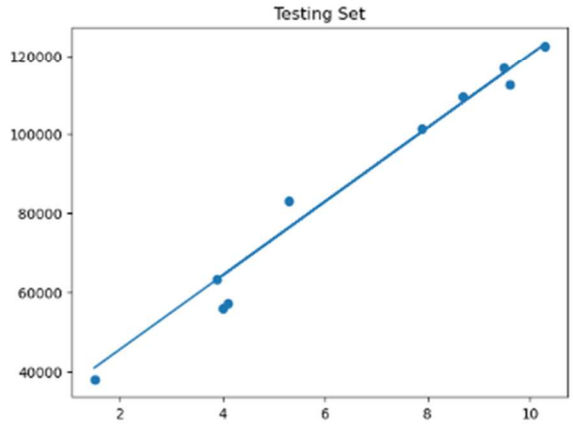
In [7]:
```
#  Calculate the mean_squared error
S_error = (d2_Pred - d2_test) ** 2
Sum_Serror = np.sum(S_error)
mean_squared_error = Sum_Serror / d2_test.size
mean_squared_error
```

Out[7]:
```
21026037.329511296
```

```python
#  Visualize both train and test data using scatter plot.
import matplotlib.pyplot as plt
# Training Data set
plt.scatter(d1_train, d2_train)
plt.plot(d1_train, reg.predict(d1_train))
plt.title('Training Set')
plt.show()

# Testing Data set
plt.scatter(d1_test, d2_test)
plt.plot(d1_test, reg.predict(d1_test))
plt.title('Testing Set')
plt.show()
```





GITHUB REPO LINK: https://github.com/maddalareshma/NNDL-ICP1