

NEURAL NETWORK ASSIGNMENT 1

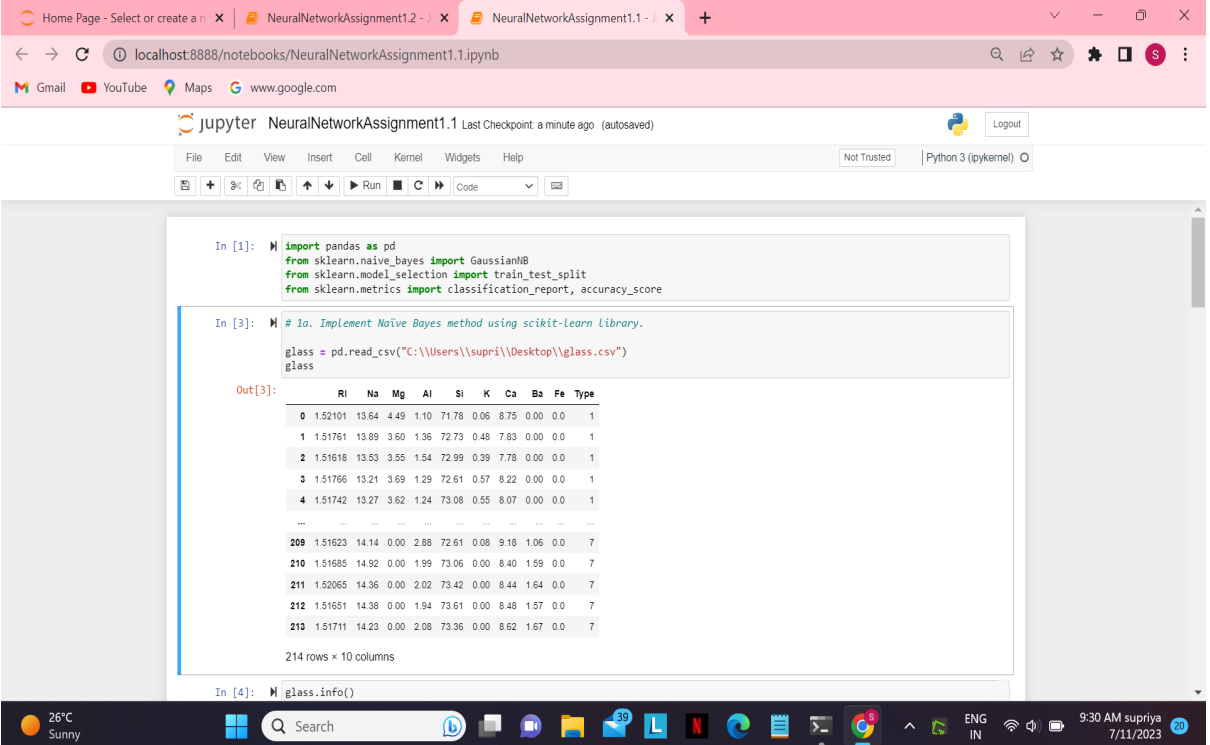
NAME: SUPRIYA SAMA
700744510

Video link:

https://drive.google.com/file/d/1rlyf51r3MSyEBM9fULhR1R2QffTP_4bP/view?usp=sharing

1. Implement Naive Bayes method using scikit-learn library.
Use the dataset available with name glass.

Using the read_csv method from the pandas module I imported a glass data set.



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [1]: import pandas as pd
        from sklearn.naive_bayes import GaussianNB
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import classification_report, accuracy_score

In [3]: # 1a. Implement Naive Bayes method using scikit-learn library.
        glass = pd.read_csv("C:\\Users\\supri\\Desktop\\glass.csv")
        glass
```

Out[3]:

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.00	0.0	1
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.00	0.0	1
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.00	0.0	1
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.00	0.0	1
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.00	0.0	1
...
209	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	1.06	0.0	7
210	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	1.59	0.0	7
211	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	1.64	0.0	7
212	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	1.57	0.0	7
213	1.51711	14.23	0.00	2.08	73.36	0.00	8.62	1.67	0.0	7

214 rows x 10 columns

```
In [4]: glass.info()
```

Use train_test_split to create the training and testing part.

sklearn module contains train_test_split method to split our data set into training and testing data sets. In this data set Type column can be used for labels. In this method, test_size defines how much proportion of data to be in the test data set. When we test_size value whole analysis results will change.

The screenshot shows a Jupyter Notebook with the following code cells:

```
In [4]: glass.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 214 entries, 0 to 213
Data columns (total 10 columns):
#   Column  Non-Null Count  Dtype  
---  -
0    RI      214 non-null    float64
1    Na      214 non-null    float64
2    Mg      214 non-null    float64
3    Al      214 non-null    float64
4    Si      214 non-null    float64
5    K       214 non-null    float64
6    Ca      214 non-null    float64
7    Ba      214 non-null    float64
8    Fe      214 non-null    float64
9    Type    214 non-null    int64  
dtypes: float64(9), int64(1)
memory usage: 16.8 KB

In [4]: #Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)

In [5]: #Train the Naive Bayes classifier
model = GaussianNB()
model.fit(x_train, y_train)

Out[5]: GaussianNB()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [6]: #Predict on the test data
y_pred = model.predict(x_test)
```

Evaluate the model on testing part using score and `classification_report(y_true, y_pred)`

In the given data, there are no missing values present in it. So, we can directly apply the machine learning algorithms on the data. sklearn module is imported to analyze the data using different algorithms. `Classification_report` and `confusion_matrix` methods to result the summary of the predictions made using the specific algorithm. These summaries can be used to compare with another algorithms to define which algorithm is better.

The screenshot shows the continuation of the Jupyter Notebook with the following code cells:

```
In [5]: #Train the Naive Bayes classifier
model = GaussianNB()
model.fit(x_train, y_train)

Out[5]: GaussianNB()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [6]: #Predict on the test data
y_pred = model.predict(x_test)

In [7]: #Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

Accuracy: 0.37209302325581395
Classification Report:
              precision    recall  f1-score   support

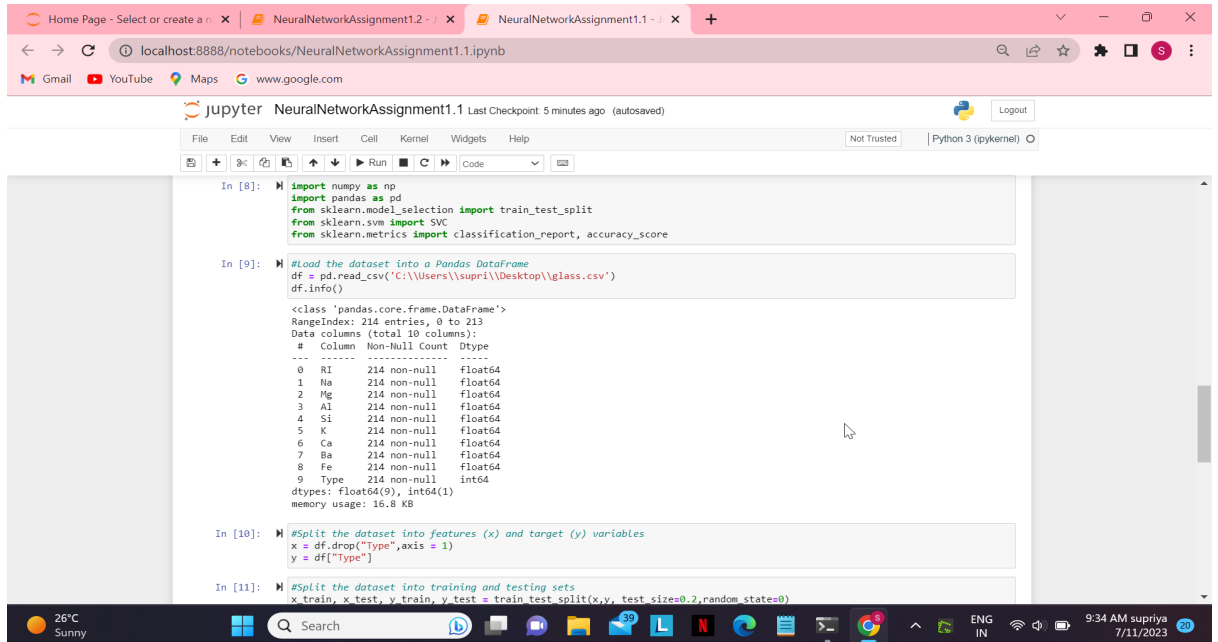
     1      0.19      0.44      0.27         9
     2      0.33      0.16      0.21        19
     3      0.33      0.20      0.25         5
     5      0.00      0.00      0.00         2
     6      0.67      1.00      0.80         2
     7      1.00      1.00      1.00         6

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

In [8]: import numpy as np
```

2. Implement linear SVM method using scikit library.

Using read_csv method from pandas module I imported glass data set.



The screenshot shows a Jupyter Notebook titled 'NeuralNetworkAssignment1.1'. The code in the first three cells is as follows:

```
In [8]: import numpy as np
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

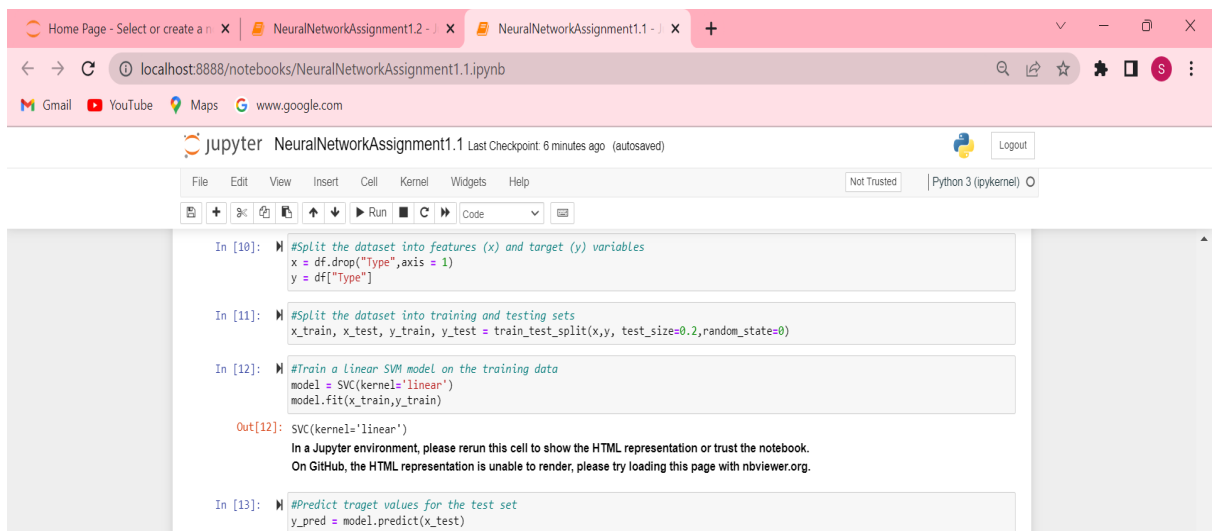
In [9]: #Load the dataset into a Pandas DataFrame
df = pd.read_csv('C:\\Users\\supri\\Desktop\\glass.csv')
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 214 entries, 0 to 213
Data columns (total 10 columns):
 # Column Non-Null Count  Dtype
---  --
0 RI      214 non-null    float64
1 Na      214 non-null    float64
2 Mg      214 non-null    float64
3 Al      214 non-null    float64
4 Si      214 non-null    float64
5 K       214 non-null    float64
6 Ca      214 non-null    float64
7 Ba      214 non-null    float64
8 Fe      214 non-null    float64
9 Type    214 non-null    int64
dtypes: float64(9), int64(1)
memory usage: 16.8 KB

In [10]: #Split the dataset into features (x) and target (y) variables
x = df.drop("Type",axis = 1)
y = df["Type"]

In [11]: #Split the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2,random_state=0)
```

Use train_test_split to create the training and testing part. sklearn module contains train_test_split method to split our data set into training and testing data sets. In this data set Type column can be used for labels. In this method, test_size defines how much proportion of data to be in the test data set. When we test_size value whole analysis results will change.



The screenshot shows the continuation of the Jupyter Notebook. The code in the next three cells is as follows:

```
In [10]: #Split the dataset into features (x) and target (y) variables
x = df.drop("Type",axis = 1)
y = df["Type"]

In [11]: #Split the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2,random_state=0)

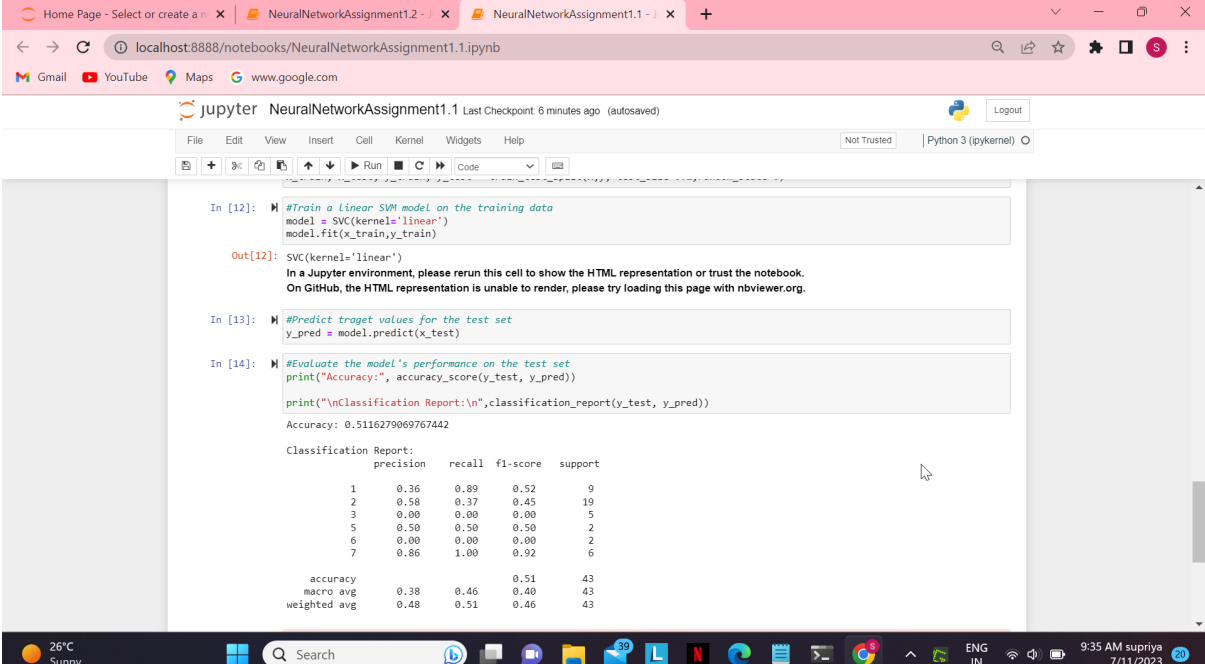
In [12]: #Train a linear SVM model on the training data
model = SVC(kernel='linear')
model.fit(x_train,y_train)

Out[12]: SVC(kernel='linear')
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [13]: #Predict target values for the test set
y_pred = model.predict(x_test)
```

Evaluate the model on testing part using score and `classification_report(y_true, y_pred)`

Support vector machine algorithm is applied to this data set using sklearn module.



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [12]: #Train a Linear SVM model on the training data
model = SVC(kernel='linear')
model.fit(x_train,y_train)

Out[12]: SVC(kernel='linear')
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.
```

```
In [13]: #Predict target values for the test set
y_pred = model.predict(x_test)

In [14]: #Evaluate the model's performance on the test set
print("Accuracy:", accuracy_score(y_test, y_pred))

print("\nClassification Report:\n",classification_report(y_test, y_pred))
```

Accuracy: 0.5116279069767442

Classification Report:

	precision	recall	f1-score	support
1	0.36	0.89	0.52	9
2	0.58	0.37	0.45	19
3	0.00	0.00	0.00	5
5	0.50	0.50	0.50	2
6	0.00	0.00	0.00	2
7	0.86	1.00	0.92	6
accuracy			0.51	43
macro avg	0.38	0.46	0.40	43
weighted avg	0.48	0.51	0.46	43

Which algorithm got better accuracy? Can you justify why?

We got better accuracy for the SVM method. We get better results using SVM than naïve bayes when we work with a glass data set. In this glass data set, SVM can handle more data than the naive bayes algorithm. Naive Bayes assumes independence between features, which can limit its ability to model complex relationships in the data.

3. Implement Linear Regression using scikit-learn.

A) Import the given “Salary_Data.csv”

I have imported the Salary_Data.csv by using read_csv.

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [4]: import numpy as np
import pandas as pd

In [28]: # Import the given "Salary_Data.csv"
dst_Sal = pd.read_csv("C:\\Users\\supri\\Desktop\\Salary_Data.csv")
dst_Sal.info()
dst_Sal.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[28]:
```

	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 [22]: A = dst_Sal.iloc[:, :-1].values #excluding last column i.e., years of experience column
B = dst_Sal.iloc[:, 1].values #only salary column

In [23]: # Split the data in train test partitions, such that 1/3 of the data is reserved as test subset.
```

B. Split the data in train_test partitions, such that $\frac{1}{3}$ of the data is reserved as test subset.

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
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 [22]: A = dst_Sal.iloc[:, :-1].values #excluding last column i.e., years of experience column
B = dst_Sal.iloc[:, 1].values #only salary column

In [23]: # 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
A_train, A_test, B_train, B_test = train_test_split(A, B, test_size=1/3, random_state=0)
```

C. Train and predict the model

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
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 [22]: A = dst_sal.iloc[:, :-1].values #excluding last column i.e., years of experience column
        B = dst_sal.iloc[:, 1].values  #only salary column

In [23]: # 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
        A_train, A_test, B_train, B_test = train_test_split(A, B, test_size=1/3, random_state=0)

In [24]: # Train and predict the model.
        from sklearn.linear_model import LinearRegression
        reg = LinearRegression()
        reg.fit(A_train, B_train)
        B_Pred = reg.predict(A_test)
        B_Pred
```

```
Out[24]: array([ 40835.10590871, 123079.39940819,  65134.55626083,  63265.36777221,
        115602.64545369, 108125.8914992 , 116537.23969801,  64199.96201652,
        76349.68719258, 100649.1375447 ])
```

D. Calculate the mean_squared error.

The screenshot shows the continuation of the Jupyter Notebook with the following code and output:

```
In [25]: # Calculate the mean_squared error
        S_error = (B_Pred - B_test) ** 2
        Sum_Serror = np.sum(S_error)
        mean_squared_error = Sum_Serror / B_test.size
        mean_squared_error
```

```
Out[25]: 21026037.329511296
```

We can see that we have got mean square error.

E. Visualise both train and test data using scatter plot.

We can see the scatter plots of both training and testing data.

