Neural Networks & Deep Learning: ICP1

Name: Dheeraj Kodakandla
Student Id: 700738796

Video Link:

https://drive.google.com/file/d/1RIT_t0YiIiKOsqAhnO5W3P6ODyYqBCLE/view?usp=sharing

GitHub Link: https://github.com/dheeraj3119/Assignment_1.git

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

In [27]:
```
# importing required libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import LinearSVC
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

Naïve Bayes

In [28]:
```
# reading "Glass.csv" file
df = pd.read_csv("glass.csv")
df.head()
```

Out[28]:

	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	0.0	0.0	1
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	1
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	1
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0	1

In [29]:
```
# seperating x_data and y_data
y_data = df['Type']
x_data = df.drop('Type', axis=1)
```

In [30]:
```
# x_data
x_data.head()
```

Out[30]:

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0

In [31]:
```
# splitting the data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.3, random_state=7)
```

In [32]:
```
# train data shape
print(x_train.shape, y_train.shape)
```

```
(149, 9) (149,)
```

In [33]:
```
# test data shape
print(x_test.shape, y_test.shape)
```

```
(65, 9) (65,)
```

In [34]:
```
# training Naive Bayes Model
nb_model = GaussianNB()
nb_model.fit(x_train, y_train)
```

Out[34]:
```
GaussianNB()
```

In [35]:
```
# predicting the x_test data using Naive Bayes Model
y_pred = nb_model.predict(x_test)
print(y_pred)
```

```
[3 3 3 3 6 3 2 3 3 3 3 2 3 3 3 1 1 2 3 6 3 2 3 7 3 7 7 1 1 3 7 2 3 5 2 7 3
 3 3 3 3 7 5 3 3 7 1 2 3 3 3 3 3 3 2 2 1 3 2 3 3 3 3 7 3]
```

In [36]:
```
# Naive Bayes Model score
print(nb_model.score(x_test, y_test))
```

```
0.24615384615384617
```

In [37]:
```
# classification report of Naive Bayes Model
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.33      0.10      0.15        20
           2       0.60      0.21      0.31        29
           3       0.03      0.25      0.05         4
           5       0.00      0.00      0.00         4
           6       0.00      0.00      0.00         1
           7       0.88      1.00      0.93         7

    accuracy                           0.25        65
   macro avg       0.31      0.26      0.24        65
weighted avg       0.47      0.25      0.29        65
```

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)

Linear SVM

In [38]:
```
# training Linear SVM Model
svm_model = LinearSVC(random_state=6)
svm_model.fit(x_train, y_train)
```

Out[38]:
```
LinearSVC(random_state=6)
```

In [39]:
```
# predicting the x_test data using Linear SVM Model
y_pred = svm_model.predict(x_test)
print(y_pred)
```

```
[2 1 2 2 1 1 2 2 2 1 1 1 1 2 1 1 1 6 2 6 1 2 2 7 2 7 7 1 2 2 7 2 1 2 2 7 1
 2 2 2 2 7 5 2 2 7 1 2 2 2 1 2 2 1 2 6 2 2 6 2 2 2 1 7 2]
```

In [40]:
```
# Linear SVM Model score
print(svm_model.score(x_test, y_test))
```

```
0.5384615384615384
```

In [16]:
```
# classification report of Linear SVM Model
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
1	0.50	0.45	0.47	20
2	0.56	0.66	0.60	29
3	0.00	0.00	0.00	4
5	0.00	0.00	0.00	4
6	0.00	0.00	0.00	1
7	0.88	1.00	0.93	7
accuracy			0.54	65
macro avg	0.32	0.35	0.34	65
weighted avg	0.50	0.54	0.52	65

Which algorithm you got better accuracy? Can you justify why?

Linear SVM has better accuracy than Naive Bayes Model because SVM can performwell in classifying multi-dimensional data and since Naive Bayes is based upon the frequency of occurrence it was not able to classify data.

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.

Linear Regression

In [41]:
```
# reading "Salary Data.csv" file
salary_df = pd.read_csv("Salary_Data.csv")
salary_df.head()
```

Out[41]:

	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 [42]:
```
# seperating x_data and y_data
y_data = salary_df['Salary']
x_data = salary_df.drop('Salary', axis=1)
```

In [43]:
```
# x_data
print(x_data.head())
```

```
   YearsExperience
0              1.1
1              1.3
2              1.5
3              2.0
4              2.2
```

In [44]:
```
# splitting the data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=(1/3), random_state=7)
```

In [45]:
```
# training Linear Regression Model
linear_model = LinearRegression()
linear_model.fit(x_train, y_train)
```

Out[45]: LinearRegression()

In [46]:
```
# predicting the x_test data using Linear Regression Model
y_pred = linear_model.predict(x_test)
print(y_pred)
```

```
[ 38744.28011204  75907.          36788.34748636  60259.53899455
  63193.43793307  52435.80849182  81774.79787705 109157.85463659
 117959.55145216 126761.24826773]
```

In [47]:
```
# calculating mean square error
mean_squared_error(y_test, y_pred)
```

Out[47]: 27563856.326517493

In [48]:

```python
# visualizing x_train data
plt.scatter(x_train, y_train)
plt.xlabel("Years Of Experience")
plt.ylabel("Salary");
plt.title("Experience vs Salary - Train Data");
```

In [28]:

```python
# visualizing x_test data
plt.scatter(x_test, y_test)
plt.xlabel("Years Of Experience")
plt.ylabel("Salary");
plt.title("Experience vs Salary - Test Data");
```

In [ ]:

```python

```