

Neural Networks & Deep Learning: ICP1

1. Implement Naïve Bayes method using scikit-learn library

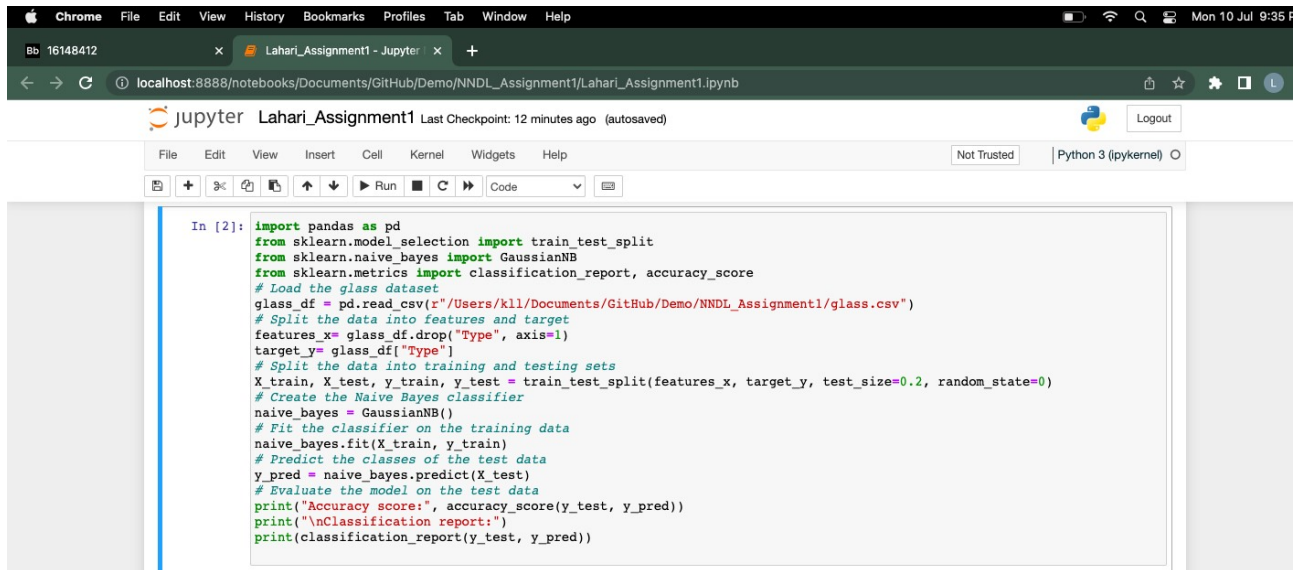
Use dataset available with name glass

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)

Ans:

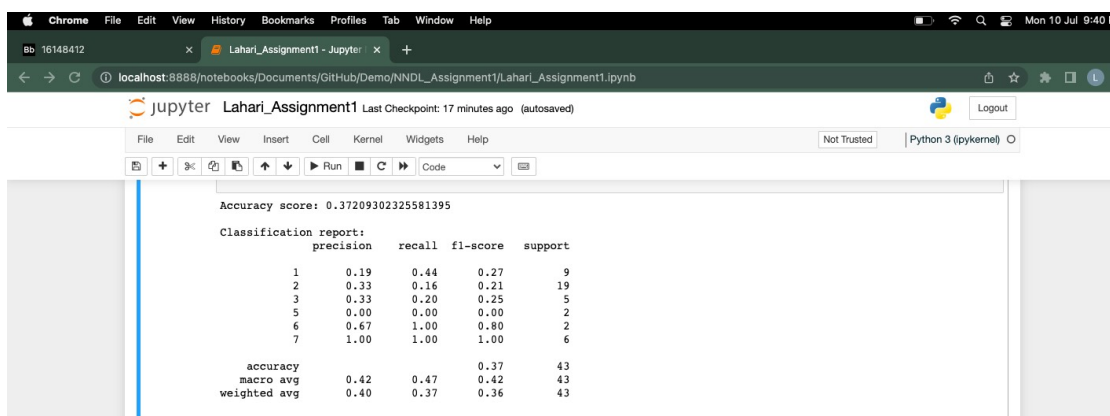


```
In [2]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, accuracy_score
# Load the glass dataset
glass_df = pd.read_csv(r"/Users/k11/Documents/GitHub/Demo/NNDL_Assignment1/glass.csv")
# Split the data into features and target
features_x = glass_df.drop("Type", axis=1)
target_y = glass_df["Type"]
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_x, target_y, test_size=0.2, random_state=0)
# Create the Naive Bayes classifier
naive_bayes = GaussianNB()
# Fit the classifier on the training data
naive_bayes.fit(X_train, y_train)
# Predict the classes of the test data
y_pred = naive_bayes.predict(X_test)
# Evaluate the model on the test data
print("Accuracy score:", accuracy_score(y_test, y_pred))
print("\nClassification report:")
print(classification_report(y_test, y_pred))
```

Steps followed:

- 1) Load the glass dataset using `pd.read_csv()` function and store in the variable.
- 2) Split the data into features and target using `drop()` function
- 3) Split the dataset into training and testing sets using `train_test_split()` function.
- 4) Created the Naive Bayes Classifier using `GaussianNB()`.
- 5) Fit the classifier on the training data using `fit()` function.
- 6) Predict the test data using `predict()` function.
- 7) Calculated the accuracy score using `accuracy_score()` function.
- 8) Generated the Classification report using `classification_report()` function.

Output:



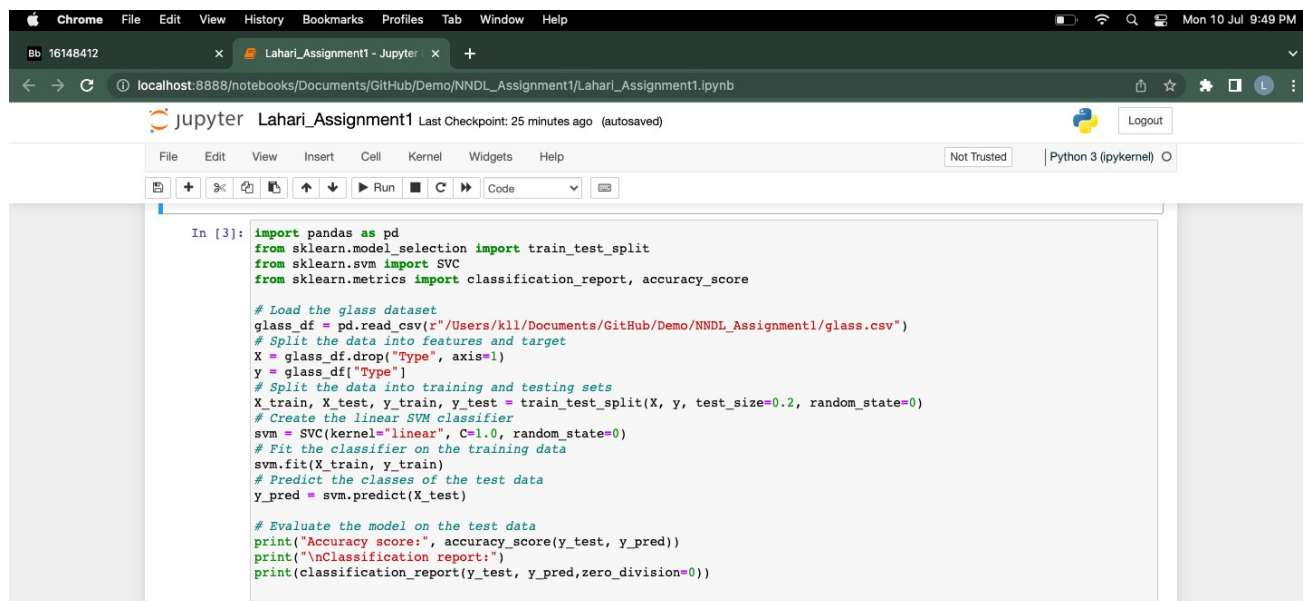
```
Accuracy score: 0.37209302325581395
Classification report:
      precision    recall  f1-score   support

     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          0.42         43
 weighted avg          0.40          0.37          0.36         43
```

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)

Ans:



```
In [3]: 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

# Load the glass dataset
glass_df = pd.read_csv(r"C:/Users/kll/Documents/GitHub/Demo/NNDL_Assignment1/glass.csv")
# Split the data into features and target
X = glass_df.drop("Type", axis=1)
y = glass_df["Type"]

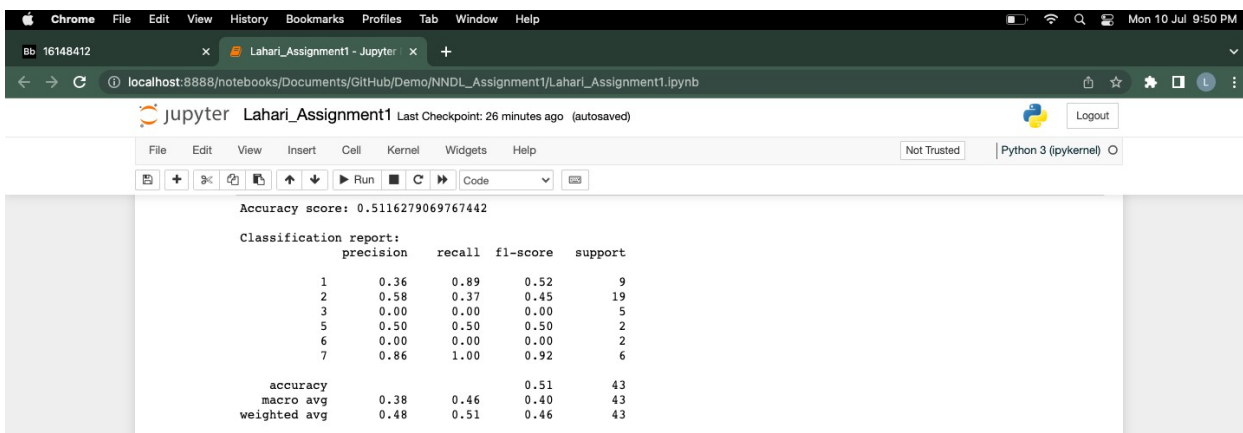
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
# Create the linear SVM classifier
svm = SVC(kernel="linear", C=1.0, random_state=0)
# Fit the classifier on the training data
svm.fit(X_train, y_train)
# Predict the classes of the test data
y_pred = svm.predict(X_test)

# Evaluate the model on the test data
print("Accuracy score:", accuracy_score(y_test, y_pred))
print("\nClassification report:")
print(classification_report(y_test, y_pred, zero_division=0))
```

Steps followed:

- 1) Load the glass dataset using `pd.read_csv()` function and store in the variable.
- 2) Split the data in to features and target using `drop()` function
- 3) Split the dataset into training and testing sets using `train_test_split()` function.
- 4) Created the Linear SVM classifier using `SVC(kernel="linear")`.
- 5) Fit the classifier on the training data using `fit()` function.
- 6) Predict the test data using `predict()` function.
- 7) Calculated the accuracy score using `accuracy_score()` function.
- 8) Generated the Classification report using `classification_report()` function.

Output:



```
Accuracy score: 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         43
 weighted avg          0.48         43
```

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

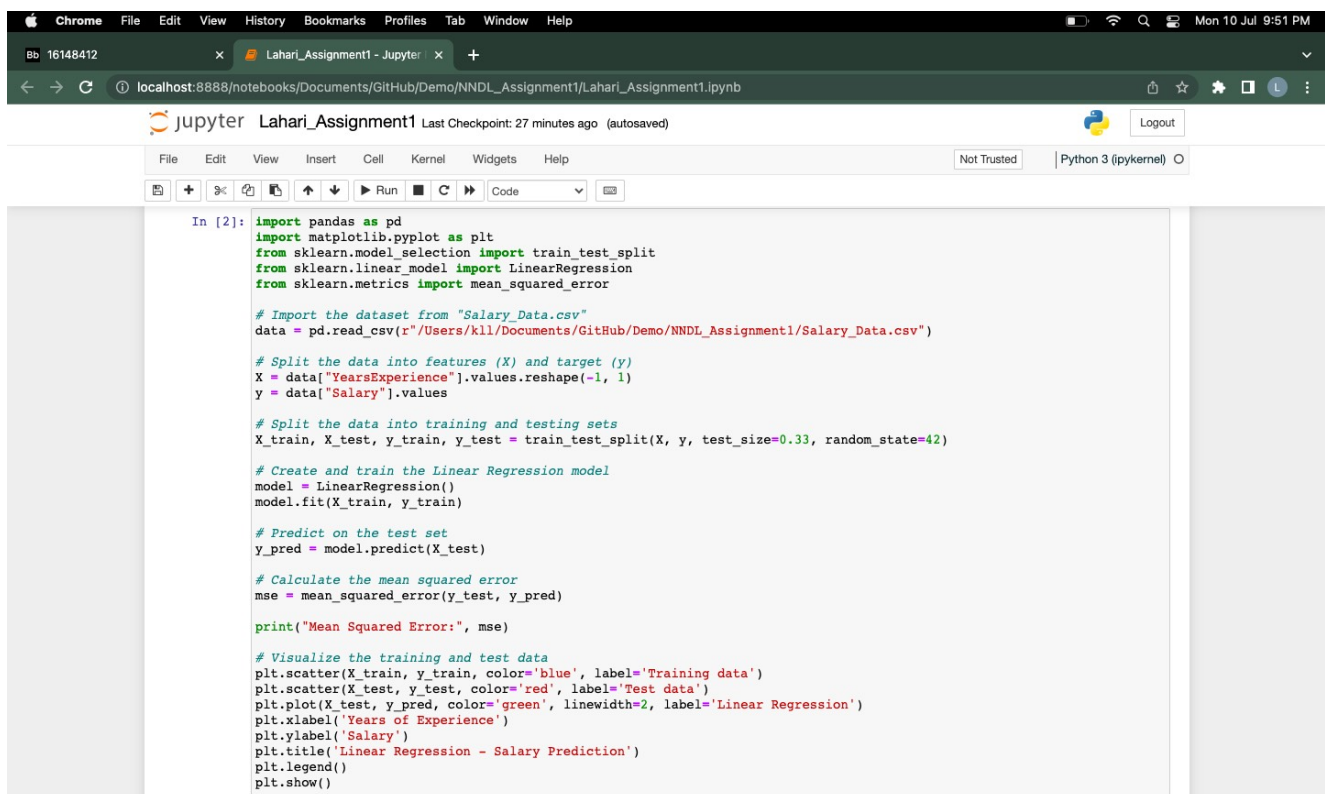
Ans

- 1) Based on the accuracy scores , the linear SVM method has a better accuracy score compared to the Naive Bayes method.
- 2)The accuracy score of 0.51 for the linear SVM method indicates that it correctly predicted the target class for 51% of the instances in your data.
- 3) On the other hand, the accuracy score of 0.37 for the Naive Bayes method indicates that it correctly predicted the target class for only 37% of the instances in your 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.

Ans:



```
In [2]: import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Import the dataset from "Salary_Data.csv"
data = pd.read_csv(r"/Users/kll/Documents/GitHub/Demo/NNDL_Assignment1/Salary_Data.csv")

# Split the data into features (X) and target (y)
X = data["YearsExperience"].values.reshape(-1, 1)
y = data["Salary"].values

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

# Create and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Calculate the mean squared error
mse = mean_squared_error(y_test, y_pred)

print("Mean Squared Error:", mse)

# Visualize the training and test data
plt.scatter(X_train, y_train, color='blue', label='Training data')
plt.scatter(X_test, y_test, color='red', label='Test data')
plt.plot(X_test, y_pred, color='green', linewidth=2, label='Linear Regression')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.title('Linear Regression - Salary Prediction')
plt.legend()
plt.show()
```

Steps followed:

- 1)Load the Salary_data dataset using pd.read_csv() function and store in the variable.
- 2)Split the data in to features and target using reshape() function
- 3)Split the dataset into training and testing sets using train_test_split() function.
- 4)Created and trained the Linear Regression Model.
- 5)Fit the classifier on the training data using fit() function.
- 6)Predict the test data using predict() function.
- 7)Calculated the mean square error using mean_squared_error() function.
- 8)Visualize the training and test data using scatter() and plot() methods.

9)By using the label() and title() methods, we put on the labels on the axis and with the show() method we finally depict the plot.

Output:

