NEURAL NETWORKS AND DEEPLEARNING: ICP1

Akshay Chandre

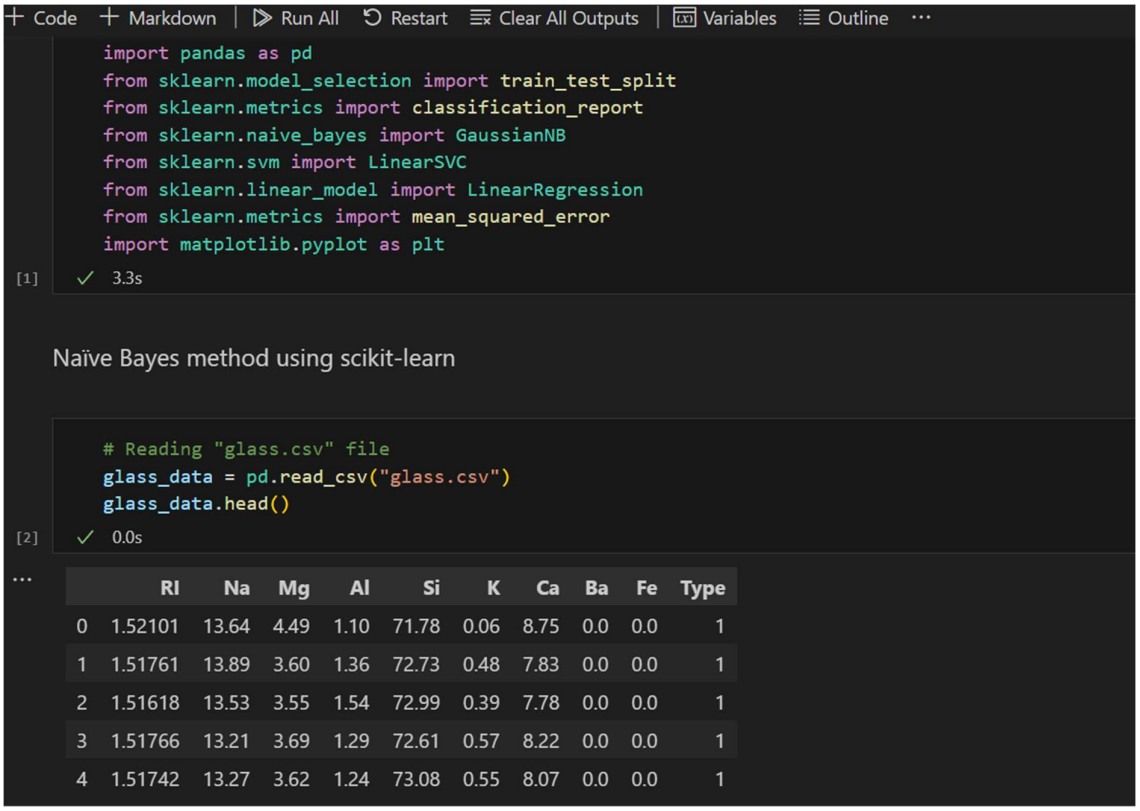
700741715

Video Link: (if the video doesn't play please download the video)
https://drive.google.com/file/d/1cWO1yKfxZBgGnhxxD3m_aop9VGYvAZGE/view?usp=sharing

code Link: https://github.com/akshaychandre47/AkshayChandre_NN-DL_ICP1.git

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)
```



	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

```
# Separating x_data and y_data
y_data = glass_data['Type']
x_data = glass_data.drop('Type', axis=1)

# 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)
```

[3] ✓ 0.0s

```
#checking for y_data
y_data.head()
```

[4] ✓ 0.0s

```
0    1
1    1
2    1
3    1
4    1
Name: Type, dtype: int64
```

```
#checking for x_data
x_data.head()
```

[5] ✓ 0.0s

	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

```
# train Naive Bayes classifier
naive_bayes = GaussianNB()
naive_bayes.fit(x_train, y_train)
```

[6] ✓ 0.0s

```
▾ GaussianNB
GaussianNB()
```

```
# predicting the test data set
y_pred = naive_bayes.predict(x_test)
print(y_pred)
```

[7] ✓ 0.0s

```
[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]
```

```
# Naive base model accuracy and classification report of Naive Bayes Model
print("Accuracy is:", naive_bayes.score(x_test, y_test))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

[8] ✓ 0.0s

```
Accuracy is: 0.24615384615384617

Classification Report:
              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)
```

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

Linear SVM

```
# Reading "glass.csv" file
glass_data = pd.read_csv("glass.csv")
glass_data.head()
```

[9] ✓ 0.0s

	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

```
# training Linear SVM Model
svm_model = LinearSVC(random_state=6, dual='auto')
svm_model.fit(x_train, y_train)
```

[25] ✓ 0.0s

```
LinearSVC(dual='auto', random_state=6)
```

```
# predicting the test data set
y_pred = svm_model.predict(x_test)
print(y_pred)
```

[13] ✓ 0.0s

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

```
# Linear SVM Model score
print(svm_model.score(x_test, y_test))
```

[14] ✓ 0.0s

```
0.676923076923077
```

```
# Linear SVM Model score and classification report of Linear SVM Model
print("Accuracy is:", svm_model.score(x_test, y_test))
print("\nClassification Report:")
print(classification_report(y_test, y_pred, zero_division=0))
```

[15] ✓ 0.0s

```
Accuracy is: 0.676923076923077

Classification Report:
              precision    recall  f1-score   support

           1       0.61      0.85      0.71        20
           2       0.70      0.66      0.68        29
           3       0.00      0.00      0.00         4
           5       0.00      0.00      0.00         4
           6       0.50      1.00      0.67         1
           7       0.88      1.00      0.93         7

    accuracy                           0.68        65
   macro avg       0.45      0.58      0.50        65
weighted avg       0.60      0.68      0.63        65
```

Linear SVM model is better than Naive bayes model, since it achieves higher overall accuracy and has well performance in terms of precision, recall, and F1-score across multiple classes, showing higher predictive capability for the given dataset.

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

```
# Reading "Salary Data.csv" file
salary_glass_data = pd.read_csv("Salary_Data.csv")
salary_glass_data.head()
```

[16] ✓ 0.0s

	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

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

[17] ✓ 0.0s


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

[17] ✓ 0.0s

```
# training Linear Regression Model
linear_model = LinearRegression()
linear_model.fit(x_train, y_train)
```

[19] ✓ 0.0s

```
▾ LinearRegression
LinearRegression()
```

```
# predicting the test data using Linear Regression Model
y_pred = linear_model.predict(x_test)
print(y_pred)
```

[20] ✓ 0.0s

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

```
# calculating mean square error
mean_squared_error(y_test, y_pred)
```

[21] ✓ 0.0s

```
27563856.32651745
```

```
# visualizing train data set using scatter plot
plt.scatter(x_train, y_train, color="blue")
plt.xlabel("Years Of Experience")
plt.ylabel("Salary")
plt.title("Experience vs Salary - Train Data")
```

[22] ✓ 0.3s

```
Text(0.5, 1.0, 'Experience vs Salary - Train Data')
```

</>

```
# visualizing test data set using scatter plot
plt.scatter(x_test, y_test, color="red")
plt.xlabel("Years Of Experience")
plt.ylabel("Salary")
plt.title("Experience vs Salary - Test Data")
```

[23] ✓ 0.2s

```
Text(0.5, 1.0, 'Experience vs Salary - Test Data')
```