Neural Networks & Deep Learning: ICP1

Rohith Thallapally
700749280

GitHub Link
https://github.com/RohithThallapally/NN-DL_assignment1

Video Link
https://drive.google.com/file/d/1kZGEX9pfL9CVSPXS929L3lkNNOgyT2bY/view?usp=drive_link

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)

```
# 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
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 = pd.read_csv("/Users/rohiththallapally/UCM/Neural networks/NNDL_Code and Data/glass.csv")
glass.head()
```

	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

```
# split the data into training and testing sets
X = glass.drop("Type", axis=1)
y = glass["Type"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# train the Naive Bayes classifier
gnb = GaussianNB()
gnb.fit(X_train, y_train)
```

```
GaussianNB()
```

```
# make predictions on the test set
y_pred = gnb.predict(X_test)
```

```
# evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.5581395348837209

Classification Report:
              precision    recall  f1-score   support

           1       0.41      0.64      0.50        11
           2       0.43      0.21      0.29        14
           3       0.40      0.67      0.50         3
           5       0.50      0.25      0.33         4
           6       1.00      1.00      1.00         3
           7       0.89      1.00      0.94         8

    accuracy                           0.56        43
   macro avg       0.60      0.63      0.59        43
weighted avg       0.55      0.56      0.53        43
```

This code demonstrates the use of Naive Bayes classifier from the scikit-learn library to train and evaluate a model using the glass dataset. The features are separated into X, which contains all columns except the "Type" column, and the target variable is stored in y. The train_test_split function is used to split the data into training and testing sets, with 80% of the data allocated for training (X_train and y_train) and 20% for testing (X_test and y_test).

A Gaussian Naive Bayes classifier object is created using GaussianNB(), and the fit method is called to train the classifier using the training data (X_train and y_train).

The accuracy of the model is calculated using accuracy_score by comparing the predicted labels (y_pred) with the true labels (y_test). The classification_report function generates a comprehensive report containing precision, recall, F1-score, and other metrics for each class.

In summary, this code loads the glass dataset, splits it into training and testing sets, trains a Naive Bayes classifier, makes predictions on the test set, and evaluates the model's accuracy and performance using classification metrics.

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
```

```
# Load the dataset
df = pd.read_csv("/Users/rohiththallapally/UCM/Neural networks/NNDL_Code and Data/glass.csv")
df.head()
```

	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

```
# Split the dataset into training and testing parts
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
# Train the linear SVM model
svc = SVC(kernel='linear', random_state=0)
svc.fit(X_train, y_train)
```

```
SVC(kernel='linear', random_state=0)
```

```
# Predict the labels on the test set
y_pred = svc.predict(X_test)
```

```
# Evaluate the model
print("Accuracy: ", accuracy_score(y_test, y_pred))
print("Classification 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
```

This code demonstrates the use of a linear Support Vector Machine (SVM) classifier from scikit-learn library to train and evaluate a model using the glass dataset.

The dataset glass.csv is loaded into df.

The features are extracted from the DataFrame df using .iloc[:, :-1], which selects all columns except the last one (assuming the last column is the target variable). The target variable is extracted using .iloc[:, -1], which selects the last column. The train_test_split function is used to split the data into training and testing sets, with 80% of the data allocated for training (X_train and y_train) and 20% for testing (X_test and y_test).

An SVM classifier object with a linear kernel is created using SVC(kernel='linear', random_state=0). The fit method is then called to train the classifier using the training data (X_train and y_train).

The trained SVM model is used to predict the labels of the test set (X_test), and the predicted labels are stored in y_pred.

The accuracy of the model is calculated by comparing the predicted labels (y_pred) with the true labels (y_test) using accuracy_score. The classification_report function generates a comprehensive report containing precision, recall, F1-score, and other metrics for each class.

In summary, this code loads the glass dataset, splits it into training and testing sets, trains a linear SVM classifier, makes predictions on the test set, and evaluates the model's accuracy and performance using classification metrics.

Which algorithm got better accuracy? Can you justify why?

Naïve Bayes ' algorithm got better accuracy. The accuracy of Naive Bayes and linear SVM depends on the dataset and problem. Naive Bayes works well with independent or weakly dependent features, while SVM handles complex relationships and non-linear dependencies. To determine the better algorithm, evaluate them using metrics like accuracy, precision, recall, or F1 score through cross-validation or a holdout set.

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.

```
import pandas as pd
import numpy as np
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 given "Salary_Data.csv"
data = pd.read_csv(r"/Users/rohiththallapally/UCM/Neural networks/NNDL_Code and Data/Salary_Data.csv")
data.head()
```

	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

```
# Split the data into input features (X) and target variable (y)
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
```

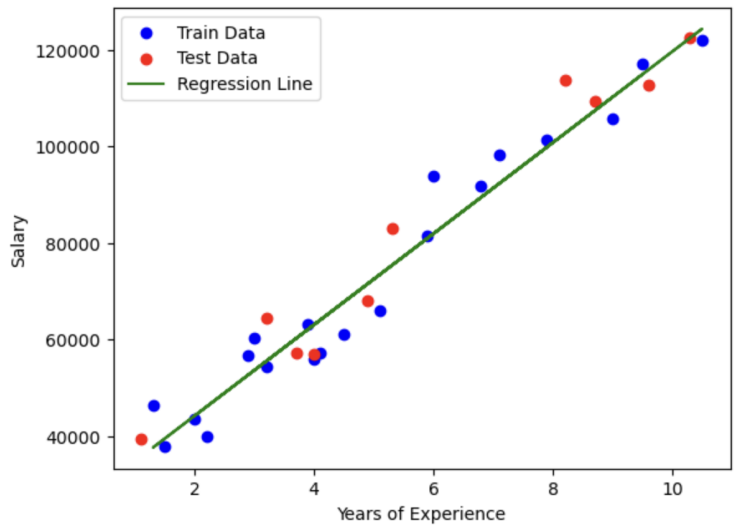
```
#  Split the data into train_test partitions
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=42)
```

```
#  Train and predict the model
model = LinearRegression()
model.fit(X_train, y_train)
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
```

```
#  Calculate the mean squared error
mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)
print("Mean Squared Error (Train):", mse_train)
print("Mean Squared Error (Test):", mse_test)
```

```
Mean Squared Error (Train): 29793161.082422983
Mean Squared Error (Test): 35301898.887134895
```

```
#  Visualize both train and test data using scatter plot
plt.scatter(X_train, y_train, color='blue', label='Train Data')
plt.scatter(X_test, y_test, color='red', label='Test Data')
plt.plot(X_train, y_train_pred, color='green', label='Regression Line')
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.legend()
plt.show()
```



This code demonstrates the use of linear regression to train a model on the given "Salary_Data.csv" dataset and evaluate its performance using mean squared error.

The dataset is loaded from the "Salary_Data.csv" file using the read_csv function from pandas. The dataset is stored in a pandas DataFrame called data.

The features are extracted from the DataFrame data using .iloc[:, :-1], which selects all columns except the last one. The target variable is extracted using .iloc[:, -1], which selects the last column.

The train_test_split function is used to split the data into training and testing sets. Here, 1/3 of the data is allocated for testing (X_test and y_test), and the remaining 2/3 is used for training (X_train and y_train).

A linear regression model object is created using LinearRegression(). The fit method is then called to train the model using the training data (X_train and y_train). Predictions are made on both the training and testing sets using the predict method.

The mean squared error (MSE) is calculated for both the training and testing predictions using the mean_squared_error function from scikit-learn.

The true target values (y_train and y_test) are compared with the predicted values (y_train_pred and y_test_pred).
A scatter plot is created to visualize the training and testing data points. The training data is plotted as blue points, the testing data as red points, and the regression line as a green line. The x-axis represents "Years of Experience," and the y-axis represents "Salary." The plot is displayed using plt.show().

In summary, this code loads the "Salary_Data.csv" dataset, splits it into training and testing sets, trains a linear regression model, makes predictions, calculates the mean squared error, and visualizes the data along with the regression line.