D. Lalitha Prasanna

Deep Learning

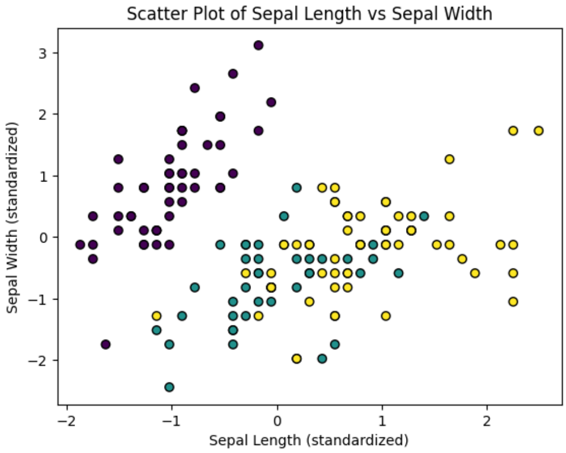
1. The Iris flower dataset consists of 50 samples from each of 3 species of Iris ((Iris setosa, Iris virginica and Iris versicolor). 4 features were measured from each sample: the length and the width of the sepals and petals, in centimetres.
   Link for dataset download: https://archive.ics.uci.edu/ml/datasets/Iris
   (i) Load the 4-dimensional iris dataset.
   (ii) Pre-process the data: Standardize the features by subtracting the mean value and scaling it to unit variance = 1).σ = 0 and standard deviation μ(i.e each feature will have mean
   (iii) Visualize the iris data using scatter plot. (i.e plot the graph between the 3 selected features: sepallength, sepal-width, petal-length).
   (iv) Using PCA perform the dimension reduction. (i.e project the original 4-d data into new 3-d data. The new components are the three main dimensions of variations).
   (v) Using explained variance show the variance attributed to each principal component and plot the bar chart explained variance Vs PCA feature. From the plot show that only two features are significant and the third feature's variance is not significant. Note that only 2 Principal components are needed.
   (vi) Using PCA perform the dimension reduction to project 4-d data into 2-d data.
   (vii) Visualize the 2D projection using scatter plot. (i.e PC1 Vs PC2)

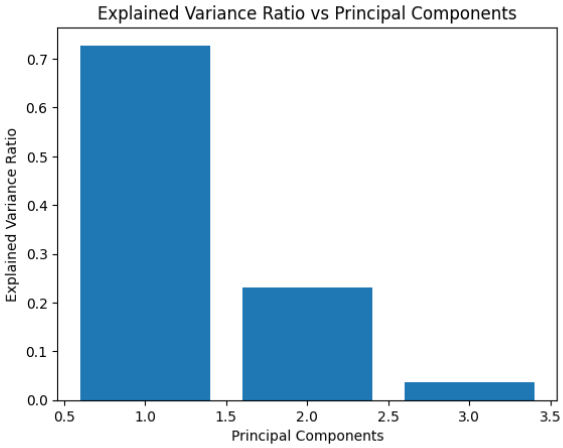
CODE :

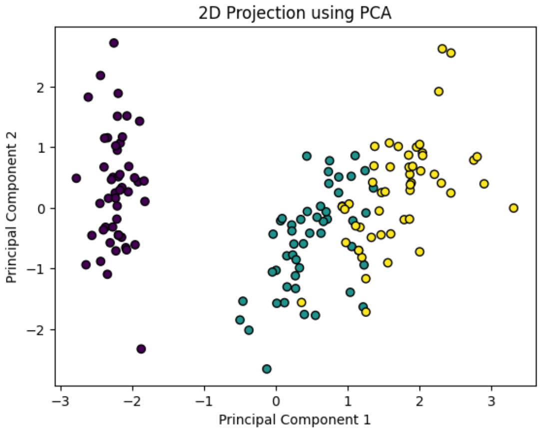
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# (i) Load the 4-dimensional iris dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data"
names = ["sepal_length", "sepal_width", "petal_length", "petal_width", "class"]
iris_data = pd.read_csv(url, names=names)

# (ii) Pre-process the data: Standardize the features
features = iris_data.drop("class", axis=1)
scaled_features = StandardScaler().fit_transform(features)

# (iii) Visualize the iris data using scatter plot
class_mapping = {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}
class_numeric = iris_data['class'].map(class_mapping)
plt.scatter(scaled_features[:, 0], scaled_features[:, 1], c=class_numeric,
cmap='viridis', edgecolor='k')
plt.xlabel('Sepal Length (standardized)')
plt.ylabel('Sepal Width (standardized)')
plt.title('Scatter Plot of Sepal Length vs Sepal Width')
plt.show()

# (iv) Using PCA perform dimension reduction to 3 dimensions
pca = PCA(n_components=3)
pca_result = pca.fit_transform(scaled_features)

# (v) Show variance attributed to each principal component
explained_variance_ratio = pca.explained_variance_ratio_
print("Explained Variance Ratio:", explained_variance_ratio)

# Plot the bar chart of explained variance vs PCA feature
```

```
plt.bar(range(1, len(explained_variance_ratio) + 1), explained_variance_ratio,
align="center")
plt.xlabel('Principal Components')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance Ratio vs Principal Components')
plt.show()

# (vi) Using PCA perform dimension reduction to 2 dimensions
pca_2d = PCA(n_components=2)
pca_result_2d = pca_2d.fit_transform(scaled_features)

# (vii) Visualize the 2D projection using scatter plot
plt.scatter(pca_result_2d[:, 0], pca_result_2d[:, 1], c=class_numeric,
cmap='viridis', edgecolor='k')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('2D Projection using PCA')
plt.show()
```

OUTPUT:







2.

(i) Load breast cancer Wisconsin (diagnostic) dataset and explore the data.

(ii) Print the top 5 records of the feature set and target labels of the loaded data.

(iii) Visualize the relationship between features using pair plot. Also check the correlation between the features and find which pair of features has strong correlation.

(iv) Divide the dataset into training and test data.

(v) Build the SVM model to fit the training set and perform the prediction on the test set.

(vi) Estimate the accuracy of the SVM classifier model by comparing the actual test set values with predicted values. Also check the precision and recall of the model.

(vii) Create the confusion matrix for the classifiers performance on the test data.

CODE

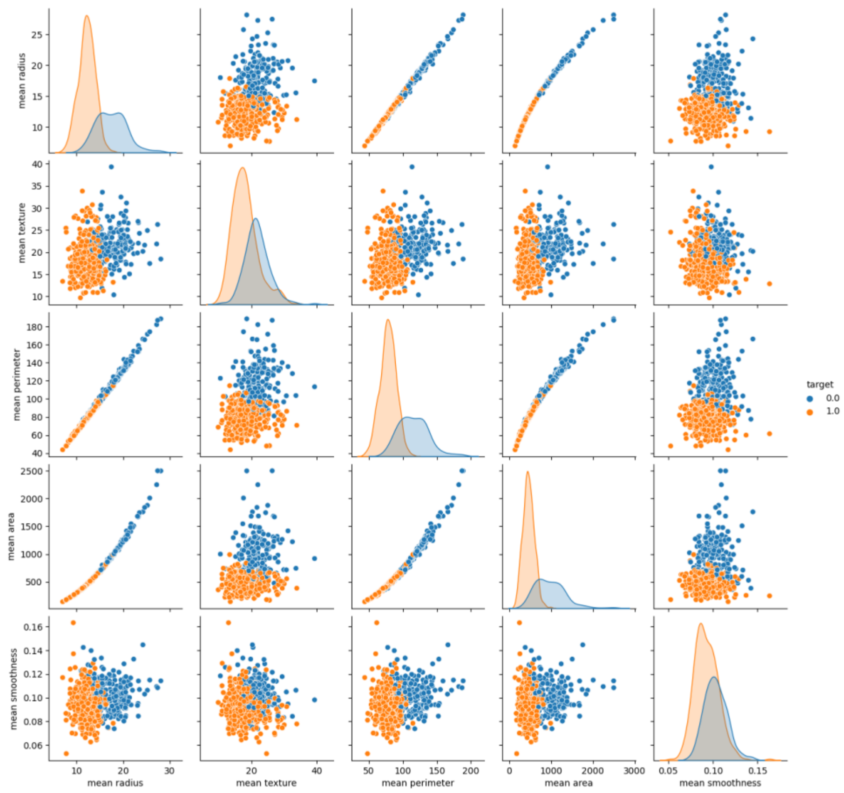
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score,
confusion_matrix

# (i) Load breast cancer Wisconsin (diagnostic) dataset and explore the data
cancer = load_breast_cancer()
data = pd.DataFrame(np.c_[cancer['data'], cancer['target']],
columns=np.append(cancer['feature_names'], ['target']))

# (ii) Print the top 5 records of the feature set and target labels
print("Top 5 records of the feature set and target labels:")
print(data.head())

# (iii) Visualize the relationship between features using pair plot
sns.pairplot(data, hue='target', vars=cancer['feature_names'][:5])
plt.show()

# Check the correlation between features
correlation_matrix = data[cancer['feature_names']].corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()

# (iv) Divide the dataset into training and test data
X_train, X_test, y_train, y_test = train_test_split(cancer['data'],
cancer['target'], test_size=0.2, random_state=42)

# (v) Build the SVM model and perform predictions on the test set
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)

# (vi) Estimate the accuracy, precision, and recall of the SVM classifier model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)

# (vii) Create the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
```

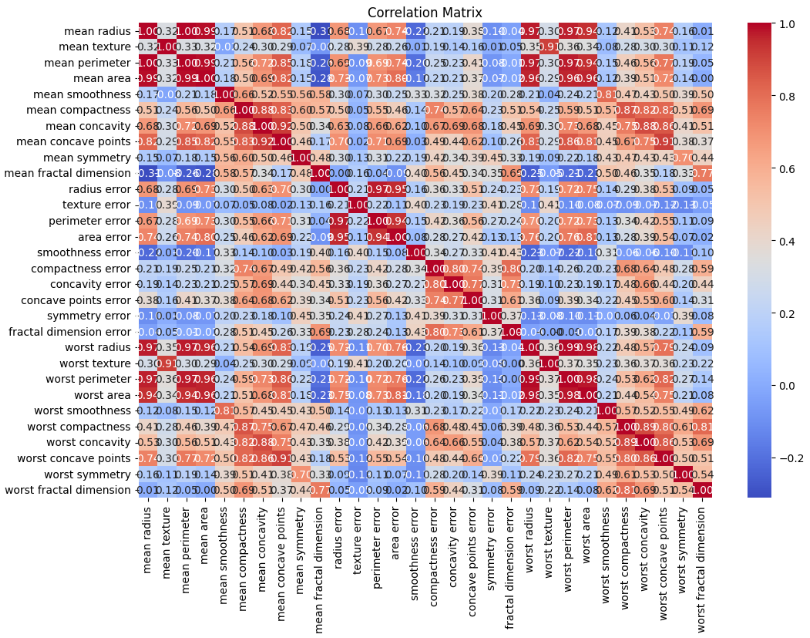
```
print(conf_matrix)
```

OUTPUT

```
Top 5 records of the feature set and target labels:
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
0                 0.07871  ...          17.33           184.60      2019.0
1                 0.05667  ...          23.41           158.80      1956.0
2                 0.05999  ...          25.53           152.50      1709.0
3                 0.09744  ...          26.50            98.87       567.7
4                 0.05883  ...          16.67           152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points
\
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   worst symmetry  worst fractal dimension  target
0          0.4601                  0.11890     0.0
1          0.2750                  0.08902     0.0
2          0.3613                  0.08758     0.0
3          0.6638                  0.17300     0.0
4          0.2364                  0.07678     0.0

[5 rows x 31 columns]
```

D. Lalitha Prasanna






```
Accuracy: 0.956140350877193
Precision: 0.9459459459459459
Recall: 0.9859154929577465
Confusion Matrix:
[[39  4]
 [ 1 70]]
```

3. Load the dataset for Naïve Bayes Classifier. Split the data into train and test sets and build the Naïve Bayes classifier on the training data. Evaluate the performance of the trained model on the test dataset.

CODE

```
# Import necessary libraries
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# Load the dataset (Iris dataset used as an example)
iris = load_iris()
X, y = iris.data, iris.target

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Build the Naïve Bayes classifier on the training data
naive_bayes_model = GaussianNB()
naive_bayes_model.fit(X_train, y_train)

# Evaluate the performance on the test dataset
y_pred = naive_bayes_model.predict(X_test)

# Calculate and print accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Display classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Display confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

OUTPUT

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

4. Implement the 4 different multilayer perceptron models to classify handwritten digits (MNIST dataset) and compare the performance of these models. (Note: MNIST images provided by scikit learn libraries are 8x8 pixel images).
   (i) MLP model with three hidden layer sizes: 400, 150, 50 with ReLU activation
   (ii) MLP model with three hidden layer sizes: 400, 150, 50 with logsig activation
   (iii) MLP model with three hidden layer sizes: 62, 32, 8 with ReLU activation
   (iv) MLP model with two hidden layer sizes: 32, 16 with ReLU activation

CODE

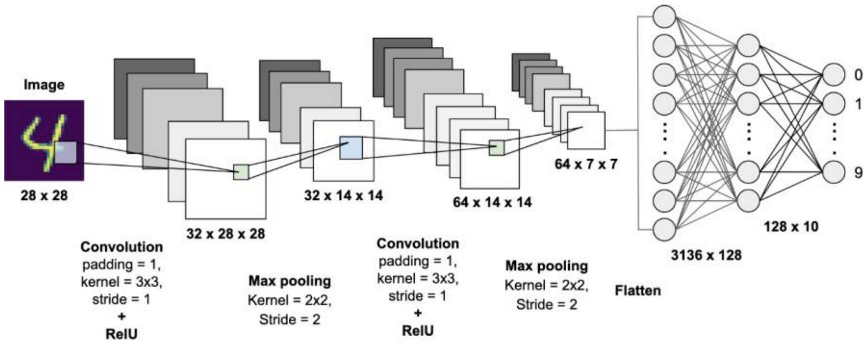
```
# Import necessary libraries
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the MNIST digits dataset
digits = load_digits()
X, y = digits.data, digits.target

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# (i) MLP model with three hidden layer sizes: 400, 150, 50 with ReLU
activation
mlp_model1 = MLPClassifier(hidden_layer_sizes=(400, 150, 50),
activation='relu', random_state=42)
mlp_model1.fit(X_train, y_train)
y_pred1 = mlp_model1.predict(X_test)

# (ii) MLP model with three hidden layer sizes: 400, 150, 50 with logsig
activation
mlp_model2 = MLPClassifier(hidden_layer_sizes=(400, 150, 50),
activation='logistic', random_state=42)
mlp_model2.fit(X_train, y_train)
y_pred2 = mlp_model2.predict(X_test)

# (iii) MLP model with three hidden layer sizes: 62, 32, 8 with ReLU activation
mlp_model3 = MLPClassifier(hidden_layer_sizes=(62, 32, 8), activation='relu',
random_state=42)
mlp_model3.fit(X_train, y_train)
y_pred3 = mlp_model3.predict(X_test)

# (iv) MLP model with two hidden layer sizes: 32, 16 with ReLU activation
mlp_model4 = MLPClassifier(hidden_layer_sizes=(32, 16), activation='relu',
random_state=42)
mlp_model4.fit(X_train, y_train)
y_pred4 = mlp_model4.predict(X_test)

# Compare the performance of these models
```

```
accuracy1 = accuracy_score(y_test, y_pred1)
accuracy2 = accuracy_score(y_test, y_pred2)
accuracy3 = accuracy_score(y_test, y_pred3)
accuracy4 = accuracy_score(y_test, y_pred4)
print("Accuracy for MLP model 1:", accuracy1)
print("Accuracy for MLP model 2:", accuracy2)
print("Accuracy for MLP model 3:", accuracy3)
print("Accuracy for MLP model 4:", accuracy4)
```

OUTPUT

```
Accuracy for Model 1: 0.9833333333333333
Accuracy for Model 2: 0.9722222222222222
Accuracy for Model 3: 0.9694444444444444
Accuracy for Model 4: 0.9694444444444444
```

5. Using CNN classify the handwritten images from MNIST data. Create a simple CNN model as shown in the figure below.



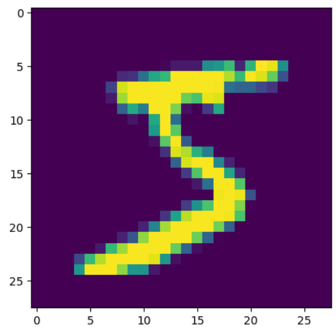
CODE

```
#importing the required libraries
import numpy as np
import tensorflow as tf
import seaborn as sn
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPool2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Dense
#loading data
(X_train,y_train) , (X_test,y_test)=mnist.load_data()
print('X_train: ' + str(X_train.shape))
print('Y_train: ' + str(y_train.shape))
print('X_test:  '  + str(X_test.shape))
print('Y_test:  '  + str(y_test.shape))
import matplotlib.pyplot as plt
%matplotlib inline
plt.imshow(np.squeeze(X_train[0]))
plt.show()
y_train[1]
X_train[1]
#normalizing the pixel values
X_train=X_train/255
X_test=X_test/255
X_train[1]
#defining model
model=Sequential()
#adding convolution layer
model.add(Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)))
#adding pooling layer
model.add(MaxPool2D(2,2))
#adding fully connected layer
model.add(Flatten())
model.add(Dense(100,activation='relu'))
```

```
#adding output layer
model.add(Dense(10,activation='softmax'))
#compiling the model
model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=[
'accuracy'])
#fitting the model
model.fit(X_train,y_train,epochs=10)
y_predicted = model.predict(X_test)
y_predicted_labels = [np.argmax(i) for i in y_predicted]
cm = tf.math.confusion_matrix(labels=y_test,predictions=y_predicted_labels)

plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

OUTPUT

```
X_train: (60000, 28, 28)
Y_train: (60000,)
X_test:  (10000, 28, 28)
Y_test:  (10000,)
```




```
Epoch 1/10
1875/1875 [==============================] - 37s 19ms/step - loss: 0.1541 -
accuracy: 0.9547
Epoch 2/10
1875/1875 [==============================] - 35s 19ms/step - loss: 0.0527 -
accuracy: 0.9845
Epoch 3/10
1875/1875 [==============================] - 38s 20ms/step - loss: 0.0347 -
accuracy: 0.9897
Epoch 4/10
1875/1875 [==============================] - 36s 19ms/step - loss: 0.0240 -
accuracy: 0.9925


Epoch 5/10
1875/1875 [==============================] - 36s 19ms/step - loss: 0.0153 -
accuracy: 0.9952
Epoch 6/10
1875/1875 [==============================] - 34s 18ms/step - loss: 0.0110 -
accuracy: 0.9966
Epoch 7/10
1875/1875 [==============================] - 36s 19ms/step - loss: 0.0079 -
accuracy: 0.9975
Epoch 8/10
```

```
1875/1875 [==============================] - 34s 18ms/step - loss: 0.0075 -
accuracy: 0.9977
Epoch 9/10
1875/1875 [==============================] - 34s 18ms/step - loss: 0.0056 -
accuracy: 0.9981
Epoch 10/10
1875/1875 [==============================] - 35s 19ms/step - loss: 0.0051 -
accuracy: 0.9982
313/313 [==============================] - 2s 6ms/step
Text(95.72222222222221, 0.5, 'Truth')
```