Student Name: Mohammad Mohammad Beigi
Student ID: 99102189
Subject: Machine Learning

Deep Learning - Dr. E. Fatemizadeh
Assignment 1

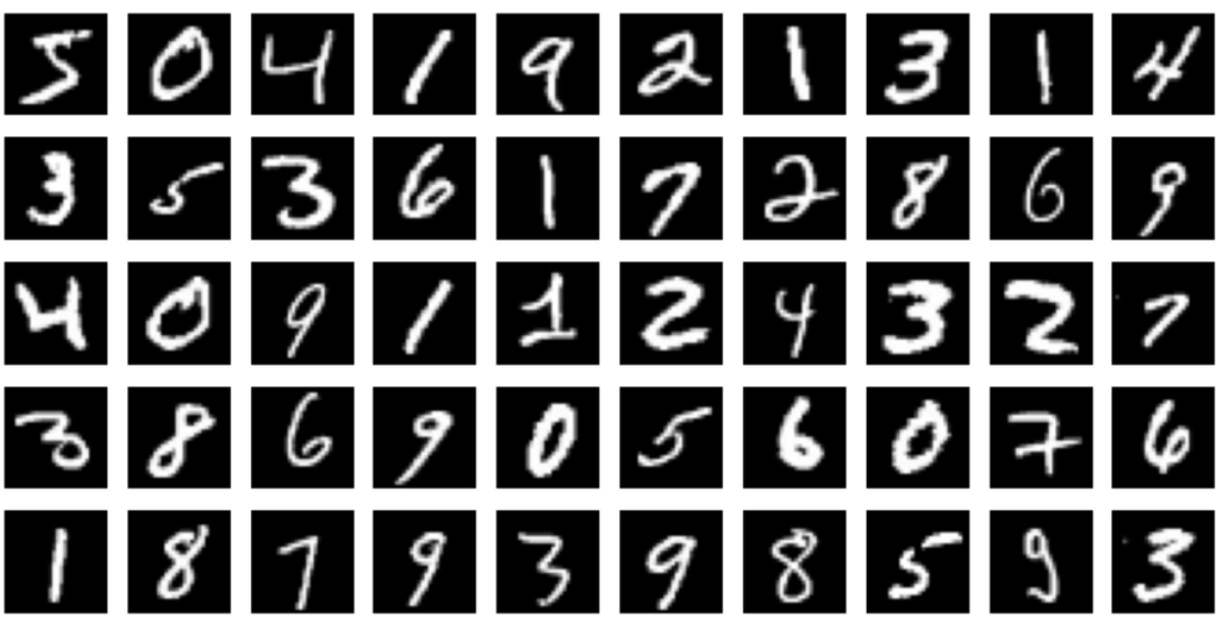
Question 1

First we load the data set:

```
[3]
    from keras.datasets import mnist
    (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

Then we show 50 first of this images:

```
# here show 50 first of this images
def show_images(num_images,X):
    #inputs dataset and number of images wants to show
    #output plot images
    plt.figure(figsize=(10, 5))
    for i in range(num_images):
        plt.subplot(5, 10, i + 1)
        plt.imshow(X[i] / 255.0, cmap='gray')
        plt.axis('off')
    plt.show()
show_images(50,x_train)
```



Then we divide data to maximum value to scale the dataset to [0 1]:

```
#scale the data set to [0 1]
#divide data to maximum value .
x_train = x_train / x_train.max()
x_test = x_test / x_test.max()
x_train_reshaped = x_train.reshape(60000, 784)
x_test_reshaped = x_test.reshape(10000, 784)
```

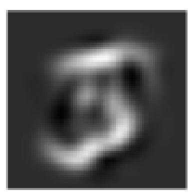
Then we calculate covariance of data:

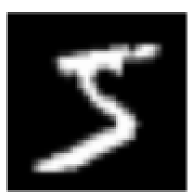
```
#calculate the covariance matrix and the
covariance = np.cov(x_train_reshaped.T)
```

Then by SVD decomposition we calculate the total variance from eigenvalues and find the first k component that contains the eplained_variance of the total variance.:

```
explained_variance = 0.7
U, S, Vh = np.linalg.svd(covariance, full_matrices=True)
## you can change this variable to get more component of datasets.
#calculate the total variance from eigenvalues and find the first k
NormSumOfS = np.cumsum(S)/np.sum(S)
index = np.argmax(NormSumOfS > explained_variance)
V = Vh.T
```

Then we calculate the dimentionally reduced data and plot it above the original data:
EV = 0.7:





EV = 0.9:

```
F = np.dot(np.dot(x_train_reshaped,V[:, :index]), V[:, :index].T)
#plot the dimentionally reduced data
#plot the original data
F_reshaped = F.reshape(60000, 28, 28)
show_images(1,F_reshaped)
show_images(1,x_train)
```

By the following function we apply PCA on train data and project train and test data to the new space:

```
def do_pca(n_components, data, data1):
    # Create a PCA instance with the desired number of components
    pca = PCA(n_components=n_components)

    # Fit the PCA model to your data and transform the data
    projected_data = pca.fit_transform(data)
    projected_data1 = pca.transform(data1)
    return projected_data, projected_data1
```

By the following function we learn a random forest with train data and test it on test data and return accuracy:

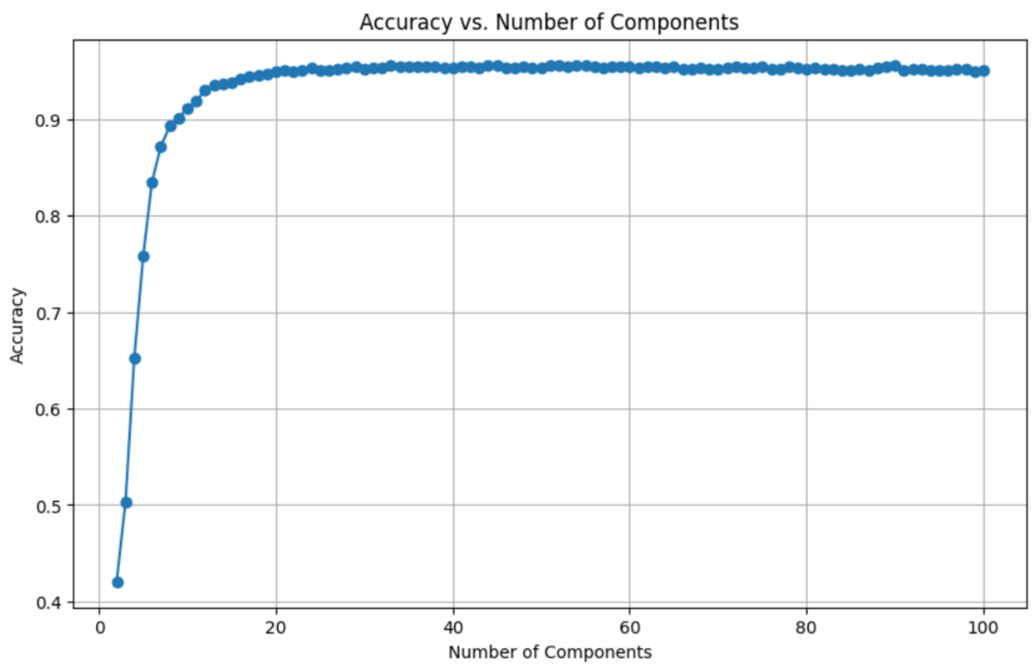
```
def ML_model(X_train, X_test, Y_train, Y_test, print_output=True):
    # You can configure the model by setting hyperparameters such as the number of trees, max depth, etc.
    model = RandomForestClassifier(n_estimators=100)  # For classification
    # model = RandomForestRegressor(n_estimators=100)  # For regression

    # Fit the model on the training data
    model.fit(X_train, y_train)

    # Make predictions on the test data
    Y_pred = model.predict(X_test)

    # Calculate the accuracy of the model
    acc = accuracy_score(Y_test, Y_pred)

    if print_output:
        print(f"Accuracy: {acc * 100:.2f}%")

    return acc
```

Now we use different number of PCs to learn RF model and save the accuracies per each number of PC:

```python
import matplotlib.pyplot as plt

# Initialize lists to store accuracy and number of components
acc_list, pc_list = [], []

for pc in range(2, 101):
    # Perform PCA with the current number of components
    projected_train_data, projected_test_data = do_pca(pc, x_train_reshaped, x_test_reshaped)

    # Train the SVM model and calculate Loading...
    accuracy = ML_model(projected_train_data, projected_test_data, y_train, y_test, print_output=False)

    # Append the accuracy and number of components to the lists
    acc_list.append(accuracy)
    pc_list.append(pc)
```

Now we plot the accuracies against number of components:

```python
plt.figure(figsize=(10, 6))
plt.plot(pc_list, acc_list, marker='o', linestyle='-')
plt.title('Accuracy vs. Number of Components')
plt.xlabel('Number of Components')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```



Then we print the number of components that maximize the accuracy and the max accuracy:

```python
max_index = acc_list.index(max(acc_list))

print("Index of maximum value:", max_index+2)

print("Index of maximum value:", max(acc_list))
```

```
Index of maximum value: 51
Index of maximum value: 0.9564
```

Question 2

By using the following function we can calculate entropy of an input array:

```python
def entropy(y: pd.Series):
    """
    Calculate the entropy of a target variable (y).
    """
    class_labels = np.unique(y)
    entropy = 0
    for cls in class_labels:
        entropy = entropy - len(y[y == cls]) / len(y) * np.log2(len(y[y == cls]) / len(y))

    return entropy
```

By using the following function we can calculate information gain between a parent and its children:

```python
def information_gain(parent, left_child, right_child):
        ''' function to compute information gain '''
        gain = entropy(parent) - (len(left_child) / len(parent)*entropy(left_child) + len(right_child) / len(parent)*entropy(right_child))
        return gain
```

By using the following function we can calculate the information gain for all features in a DataFrame:

```python
def information_gains(X: pd.DataFrame, y: pd.Series):
    """
    Calculate the information gain for all features in a DataFrame (X) with respect to the target variable (y).
    """
    information_gains_dict = {}
    for column in X.columns:
        information_gains_dict[column] = information_gain(X[column], yl, yr)

    return information_gains_dict
```

Now we define Node class with the properties below. Note that each node can be a decision node or a leaf node:

```python
class Node():
    def __init__(self, feature_index=None, threshold=None, left=None, right=None, info_gain=None, value=None):
        ''' constructor '''
        self.feature_index = feature_index
        self.threshold = threshold
        self.left = left
        self.right = right
        self.info_gain = info_gain
        self.value = value
```

Now we train our classifier by defining DecisionTreeClassifier class:
By the following function we find the best threshold and best feature to split:

```python
def get_best_split(self, X, Y, ns, num_features):
    dataset = np.concatenate((X, Y.reshape(-1, 1)), axis=1)
    best_split = {}
    max_info_gain = -float("inf")
    # all the features
    for feature_index in range(num_features):
        feature_values = dataset[:, feature_index]
        possible_thresholds = np.unique(np.floor(feature_values))
        for threshold in possible_thresholds:
            dataset_left, dataset_right = self.split(dataset, feature_index, threshold)
            if len(dataset_left)>0 and len(dataset_right)>0:
                y, left_y, right_y = dataset[:, -1], dataset_left[:, -1], dataset_right[:, -1]
                curr_info_gain = information_gain(y, left_y, right_y)
                if curr_info_gain>max_info_gain:
                    best_split["feature_index"] = feature_index
                    best_split["threshold"] = threshold
                    best_split["dataset_left"] = dataset_left
                    best_split["dataset_right"] = dataset_right
                    best_split["info_gain"] = curr_info_gain
                    max_info_gain = curr_info_gain
    return best_split
```

By the following recursive function we build our decision tree.

```python
def build_tree(self, X, Y, curr_depth=0):
    ns, num_features = np.shape(X)
    if curr_depth<=self.max_depth:
        best_split = self.get_best_split(X, Y,  num_features)# find the best split
        if best_split["info_gain"]>0:
            ns, num_features = np.shape(X)
            left_subtree = self.build_tree(best_split["dataset_left"][:, :10], best_split["dataset_left"][:, 10], curr_depth+1)
            right_subtree = self.build_tree(best_split["dataset_right"][:, :10], best_split["dataset_right"][:, 10], curr_depth+1)
            return Node(best_split["feature_index"], best_split["threshold"],
                        left_subtree, right_subtree, best_split["info_gain"])
    leaf_value = self.calculate_leaf_value(Y)
    return Node(value=leaf_value)
```

By the following piece of Code we load the MNIST dataset and reshape the data from 28x28 matrices to 784 arrays and then initialize PCA with 10 components and fit PCA on the training data and transform both the training and test data and then normalize the data by dividing by the maximum absolute value and finally multiply the results in 20 to have 40 thresholds at most.

```python
# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# Reshape the data from 28x28 matrices to 784 arrays
x_train = x_train.reshape(x_train.shape[0], 784)
x_test = x_test.reshape(x_test.shape[0], 784)
# Initialize PCA with 10 components
n_components = 10
pca = PCA(n_components=n_components)
# Fit PCA on the training data and transform both the training and test data
x_train_pca = pca.fit_transform(x_train)
x_test_pca = pca.transform(x_test)
# Calculate the maximum absolute value in the training and test data
max_abs_train = np.max(np.abs(x_train_pca))
max_abs_test = np.max(np.abs(x_test_pca))
# Normalize the data by dividing by the maximum absolute value
x_train_pca_normalized = x_train_pca / max_abs_train
x_test_pca_normalized = x_test_pca / max_abs_test
# to have a bound for number of thresholds in each node:
x_train_pca_normalized *= 20
x_test_pca_normalized *= 20
```

The results of classifying the MNIST dataset by our classifier are:
Depth = 13:

```python
[8]  classifier = DecisionTreeClassifier( max_depth=13)
     classifier.fit(x_train_pca_normalized,y_train)
```

```python
[9]  Y_pred = classifier.predict(x_test_pca_normalized)
     from sklearn.metrics import accuracy_score
     accuracy_score(y_test, Y_pred)
```

```
0.8438
```

Depth = 8:

```python
classifier = DecisionTreeClassifier( max_depth=8)
classifier.fit(x_train_pca_normalized,y_train)
```

```python
[6]  Y_pred = classifier.predict(x_test_pca_normalized)
     from sklearn.metrics import accuracy_score
     accuracy_score(y_test, Y_pred)
```

```
0.7906
```

Depth = 3:

```
[7] classifier = DecisionTreeClassifier( max_depth=3)
    classifier.fit(x_train_pca_normalized,y_train)
```

```
[8] Y_pred = classifier.predict(x_test_pca_normalized)
    from sklearn.metrics import accuracy_score
    accuracy_score(y_test, Y_pred)
```

```
0.5646
```

Question3

Load data and print the shape:

```
import pandas as pd
data = pd.read_csv('/content/sample_data/Heart_Disease_Dataset.csv')
# Print the shape of the data
print("Shape of the data: ", data.shape)
```

Check for missing values in each column and Print the missing values count for each column:

```
# Check for missing values in each column
missing_values = data.isnull().sum()

# Print the missing values count for each column
print("Missing values in each column:")
print(missing_values)
```

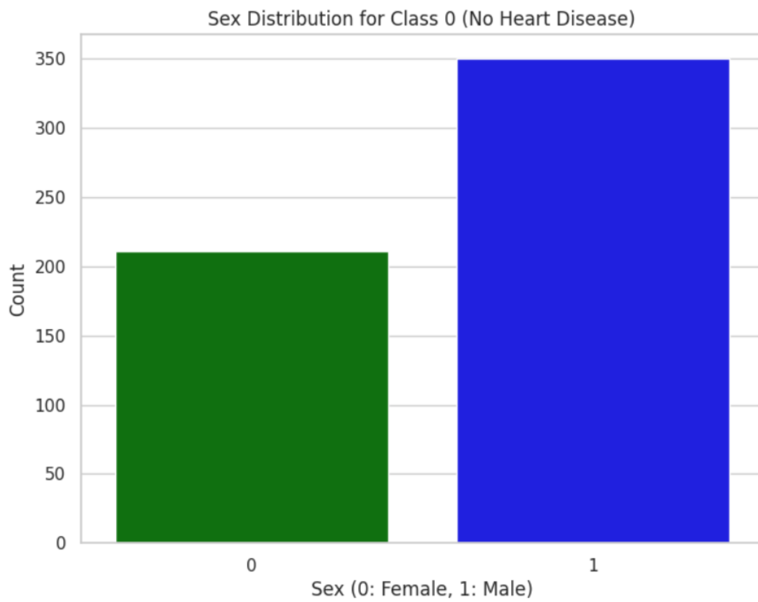
```
Missing values in each column:
age                    0
sex                    0
chest pain type        0
resting bp s           0
cholesterol            0
fasting blood sugar    0
resting ecg            0
max heart rate         0
exercise angina        0
oldpeak                0
ST slope               0
target                 0
dtype: int64
```

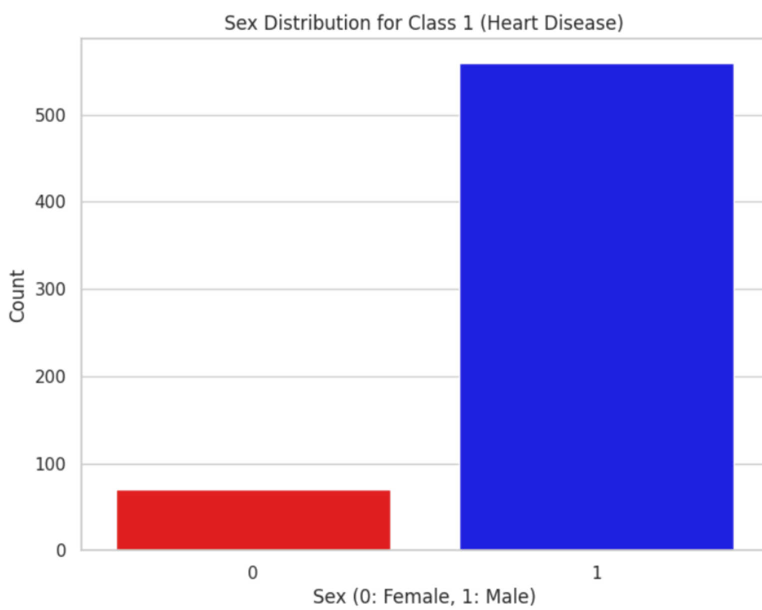
Check for the class balance:

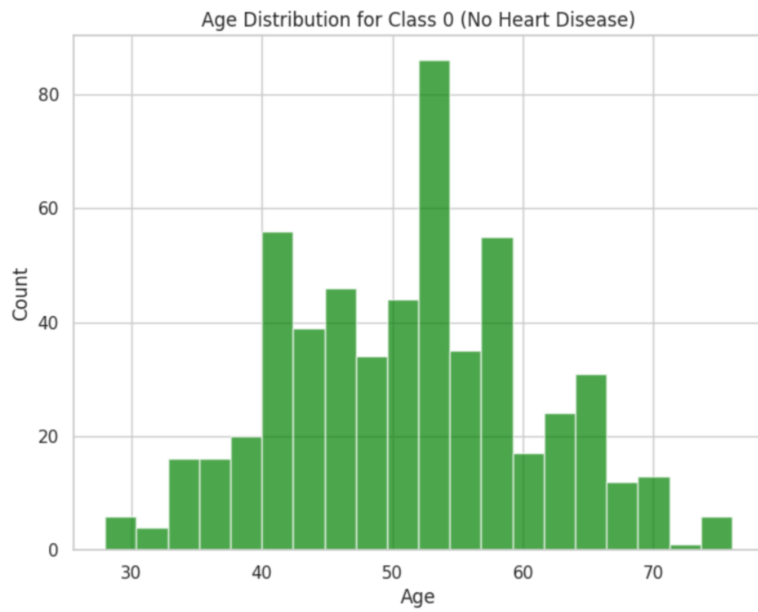
```
# 'target' is the name of the target variable
class_balance = data['target'].value_counts()

# Print the class balance
print("Class balance:")
print(class_balance)
```

```
Class balance:
1    629
0    561
Name: target, dtype: int64
```

Plot the sex and age distribution for class 0 and 1:







Calculate Z-scores for all data and remove outliers by the criteria abs(Z) > 3:

```python
import numpy as np
from scipy.stats import zscore
# Calculate Z-scores for all data
z_scores = zscore(data)

# Set a threshold for identifying outliers
threshold = 3

# Find outliers
outliers = (np.abs(z_scores) > threshold).any(axis=1)

# Print the outliers
print("Outliers:")
print(data[outliers])
```

```python
# Remove outliers from the dataset
data = data[~outliers]
print(data.shape)
```

Complete outliers are in the notebook.

Outliers:

	age	sex	chest pain type	resting bp s	cholesterol
30	53	1	3	145	518
76	32	1	4	118	529
109	39	1	2	190	241
149	54	1	4	130	603
167	50	1	4	140	231
242	54	1	4	200	198
325	46	1	4	100	0
366	64	0	4	200	0
371	60	1	4	135	0
391	51	1	4	140	0
400	61	1	3	200	0
450	55	1	3	0	0
593	61	1	4	190	287
618	67	0	3	115	564
704	59	1	1	178	270
734	56	0	4	200	288
761	54	1	2	192	283
773	55	1	4	140	217
793	51	1	4	140	298
852	62	0	4	160	164
978	62	0	4	160	164
1010	55	1	4	140	217
1013	56	0	4	200	288
1039	67	0	3	115	564
1070	59	1	1	178	270
1075	54	1	2	192	283
1078	51	1	4	140	298
1172	58	1	4	114	318

	fasting blood sugar	resting ecg	max heart rate	exercise angina
30	0	0	130	0
76	0	0	130	0

Then apply several SVMs and the results are:

```
Kernel: linear
Accuracy: 0.7564469914040115
Precision: 0.7696629213483146
Recall: 0.7569060773480663
F1 Score: 0.7632311977715877

Kernel: rbf
Accuracy: 0.667621776504298
Precision: 0.6217228464419475
Recall: 0.9171270718232044
F1 Score: 0.7410714285714286

Kernel: poly
Accuracy: 0.7220630372492837
Precision: 0.8620689655172413
Recall: 0.5524861878453039
F1 Score: 0.6734006734006733
```

```
Best RBF C: 0.2
Best RBF Gamma: 1.9
Accuracy with best hyperparameters: 0.8522349570200572
```