Question 2: Confusion Matrix

A confusion matrix is a table used to visualize the performance of a classification machine learning model. It shows how many times the model correctly or incorrectly predicted each class. This helps identify areas for improvement, like high false positives for a specific class, allowing data scientists to refine the model for better accuracy. We would like to see how you can use this tool to analyze the errors of your model.

Q2.1

Get the MNIST data using `fetch_openml`. Since the data is already shuffled, take the first 30,000 elements as your train dataset.

```
"""# Confusion Matrix

## 1.Get the MNIST data using fetch_openml. Since the data is already shuffled, take the first 30,000 elements as your train dataset.

1. get the MNIST data.

### **TO DO: FILL THE BLANK LINE**
"""

from sklearn.datasets import fetch_openml

mnist = fetch_openml('mnist_784', version=1)

X, y = mnist.data, mnist.target
X

"""since the data is already shuffled take the first 30000 elements as your train dataset.

### **TO DO: FILL THE BLANK LINE**
"""

X_train, X_test, y_train, y_test = X[:30000], X[30000:], y[:30000], y[30000:]

"""Use the SGD classifier of Scikitlearn to classify the digits, the random state is set for the sake of reproducibility."""

from sklearn.linear_model import SGDClassifier
sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train)
```

Figure 1: Code Snippet

Q2.2

Preprocess your data using the Standard Scalar preprocessor. First, considering the nature of your data (pixels) and your classifier (SGD), explain why Standard Scalar is a good choice for data preprocessing. Next, compare the Standard Scalar with Min-Max Scaling and with no processing at all, and explain why they might not be a good choice for preprocessing.

```
"""## 2. Use the provided code to preprocess the data and train your model.

### **TO DO: FILL THE BLANK LINEs**
"""

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler() #------ fill this line
X_train_scaled = scaler.fit_transform(X_train.astype("float64"))
```

Figure 2: Code Snippet

Comment: explain why Standard Scalar is a good choice for data preprocessing

Comment: compare the Standard Scalar with Min-Max Scaling and with no processing at all, and explain why they might not be a good choice for preprocessing.

Q2.3

Make a colored diagram of the confusion matrix using `ConfusionMatrixDisplay`. Explain the result. Which numbers are getting misclassified? Is there any specific correlation between any 2 digits?

```
"""## 3. Make a colored diagram of the confusion matrix using ConfusionMatrixDisplay. Explain the result. Which numbers are getting misclassified? Is there any specific correlati

from sklearn.model_selection import cross_val_predict

"""### **TO DO: FILL THE BLANK LINE**"""

from sklearn.metrics import ConfusionMatrixDisplay

y_train_pred = cross_val_predict(sgd_clf, X_train_scaled, y_train, cv=3)
plt.rc('font', size=9)
ConfusionMatrixDisplay.from_predictions(y_train, y_train_pred, display_labels=sgd_clf.classes_, cmap='viridis') #----- fill here
plt.show()
```

Figure 3: Code Snippet

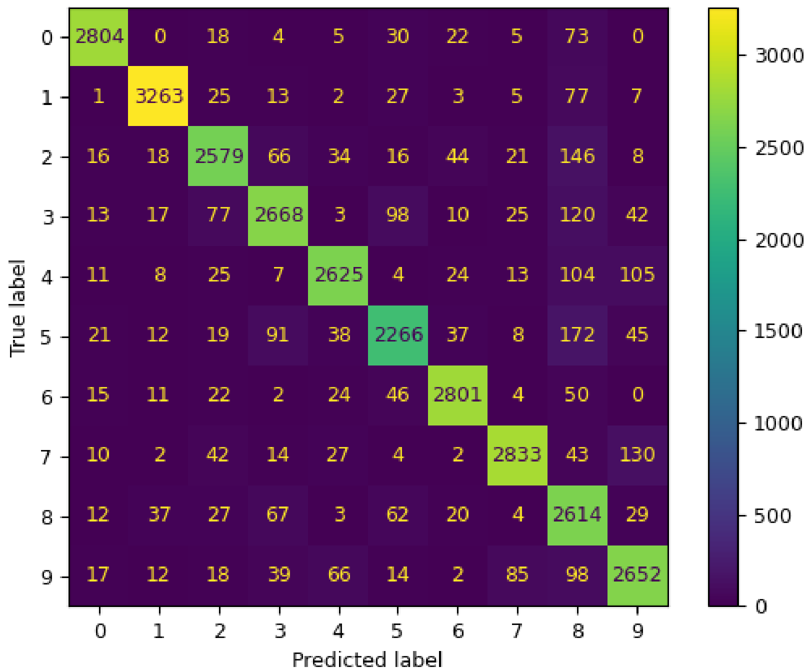


Figure 4: Confussion Matrix

Comment: Explain the result.

Comment: Which numbers are getting misclassified? Is there any specific correlation between any 2 digits?

Q2.4

Normalize the confusion matrix by dividing each value by the total number of images in the corresponding (true) class. How does this help you to get a better understanding of the errors of your model?

```
"""##4. Normalize the confusion matrix by dividing each value by the total number of images in the corresponding (true) class. How does this help you to get a better understandin

### **TO DO: FILL THE BLANK LINE**
"""

plt.rc('font', size=10)
ConfusionMatrixDisplay.from_predictions(y_train, y_train_pred, display_labels=sgd_clf.classes_, cmap='viridis', normalize='true') #----- fill here

plt.show()
```

Figure 5: Code Snippet

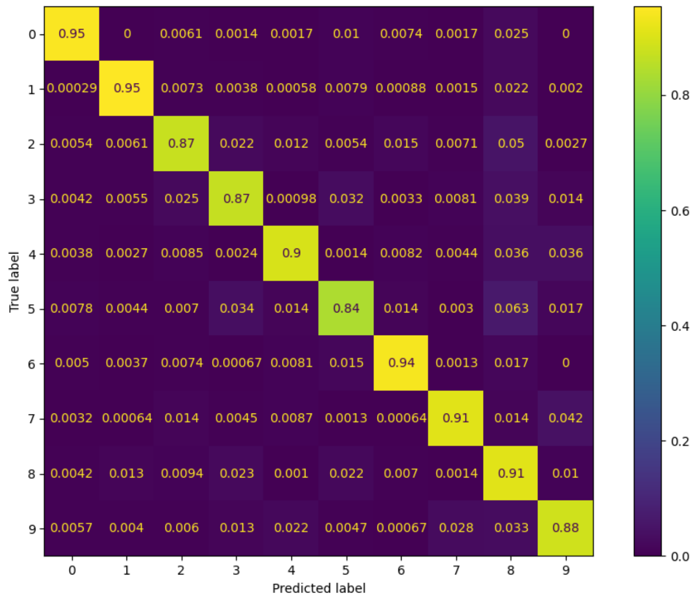


Figure 6: Normalized Confussion Matrix

Q2.5

Although it is not immediately clear from the diagram in the previous section, closer examination reveals that many digits have been incorrectly identified as 8s. To highlight these errors more, one may assign zero weight to the correct predictions. Make the errors in the confusion matrix more significant and put zero weight on the correct predictions (use the `sample_weight` feature). Plot out the updated confusion matrix and explain what you observe.

```
"""## 5. Make the errors more significant and put zero weight on the correct predictions. Explain your results.

### **TO DO: FILL THE BLANK LINE**
"""

sample_weight = (y_train_pred != y_train)
plt.rc('font', size=10)
ConfusionMatrixDisplay.from_predictions(y_train, y_train_pred,
                                        display_labels=sgd_clf.classes_,
                                        normalize="true", values_format=".0%")
plt.show()
```

Figure 7: Code Snippet

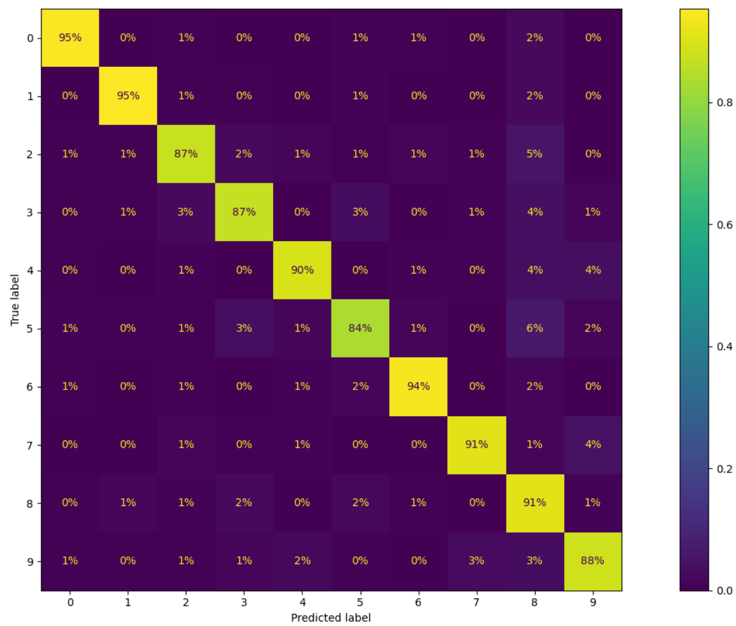


Figure 8: Corrected Confussion Matrix

Conclusion

In conclusion, the MNIST dataset has been successfully loaded from the source on internet. After certain preprocesses to the data, a linear classifier with SGD (Stochastic Gradient Descent) has been used for training a model. A part of the data tested the model. The results constituted a confusion matrix and the matrix has been analyzed with different versions.

Question 3: Nearest Neighbors from Scratch

In this exercise, you will complete the implementation of the k-Nearest Neighbours (KNN) algorithm in the skeleton code. Do not change the regression, classification dataset and query values for the final submission.

Q3.1

Implement the function to calculate the Euclidean distance between two points.

```python
def euclidean_distance(point1, point2):
    # implement this function to return euclidean distance between point1 and point2
    return abs(point1-point2)
```

Figure 1: Code Snippet for Euclidean Distance Function

Q3.2

Implement the functions to calculate mean and mode which will serve as choice functions for the KNN model. For regression tasks, use the mean; for classification tasks, use the mode.

```python
def mean(labels):
    # implement this function to return the mean of the labels.
    sum = 0
    for val in labels:
        sum += val

    return sum/len(labels)
```

Figure 2: Code Snippet for Mean Function

```python
def mode(labels):
    # implement this function to return the mode of the labels.
    occurences = {}
    for val in labels:
        if(val in occurences.keys()):
            occurences[val] += 1
        else:
            occurences[val] = 1

    max_val = 0
    max_occ = 0
    for val in occurences.keys():
        if(occurences[val] > max_occ):
            max_val = val
            max_occ = occurences[val]

    return max_val
```

Figure 3: Code Snippet for Mode Function

Q3.3

Complete the knn function by following the comments provided in the skeleton code and run your custom KNN on your regression and classification data to find the height of the person who is 55 years old and whether an 18-year-old likes pineapple or not.

```python
def knn(data, query, k, distance_fn, choice_fn):
    """...
    neighbor_distances_and_indices = []  # in order with the dataset

    # Calculate the distance between the query example and all the examples in the data.
    index=0
    for val in data.iloc[:,0]:
        neighbor_distances_and_indices.append(list([euclidean_distance(val, query), index]))
        index += 1

    # Sort the distances and return the labels of the k nearest neighbors.
    neighbor_distances_and_indices.sort()

    # Pick the first k entries from the sorted collection
    k_nearest_neighbors = neighbor_distances_and_indices[:k]

    # Get the labels of the selected k entries
    k_nearest_labels = []
    for ls in k_nearest_neighbors:
        k_nearest_labels.append(data.iloc[ls[1], 1])

    # If regression (mean), if classification (mode)
    return choice_fn(k_nearest_labels)
```

Figure 4: Code Snippet for KNN Function

```python
# # Load the Regression Data. The first index consists of age(feature) and the second index is the label. The label is height of the person in
import pandas as pd
import numpy as np

reg_data = pd.read_csv('ece657_a2_q3_data/regression_data.csv') # Load the data

reg_query = np.array([[55]])  # reshape to fit scikit-learn requirements

# Custom KNN Prediction
custom_reg_prediction = knn(reg_data, reg_query, k=3, distance_fn=euclidean_distance, choice_fn=mean)

# print(f"The predicted height of {reg_query[0][0]} years old person: {custom_reg_prediction} cm")
print("Custom KNN Regression Prediction:", custom_reg_prediction)
```

Figure 5: Code Snippet for Regression Prediction

```python
# Load the Classification Data. The first index consists of age(feature) and the second index is the label. The label 0 is for likes pineapple
clf_data = pd.read_csv('ece657_a2_q3_data/classification_data.csv') # Load the data

clf_query = np.array([[18]])  # reshape to fit scikit-learn requirements

# Custom KNN Prediction
custom_clf_prediction = knn(clf_data, clf_query, k=3, distance_fn=euclidean_distance, choice_fn=mode)

# print(f"Would a {clf_query[0][0]} years old person like pinapples: {bool(custom_clf_prediction)}")
print("Custom KNN Classification Prediction:", custom_clf_prediction)
```

Figure 6: Code Snippet for Classification Prediction

Q3.4 (NOT FINISHED)

Using KNN library from scikit-learn. In the skeleton code, add code to run the same datasets with the same k value for `KNeighborsRegressor` and `KNeighborsClassifier`. Did you get the same value for regression and classification for the datasets provided in the skeleton code?

```python
def sklearn_knn_regression(reg_data, reg_query):
    # Initialize the KNN regressor with 3 nearest neighbors
    knn_reg = KNeighborsRegressor(n_neighbors=3)

    # Fit the model on the training data; use all but the last column as features and the last column as the target
    knn_reg.fit(reg_data.iloc[:,:-1], reg_data.iloc[:,-1])

    # Predict the output for the provided query and return the first (and likely only) prediction
    skl_reg_prediction = knn_reg.predict(reg_query)

    return skl_reg_prediction
```

Figure 7: Code Snippet for Scikit-Learn Regression Function

```python
# Scikit-learn KNN Regression
skl_reg_prediction = sklearn_knn_regression(reg_data, reg_query)

print("Scikit-learn KNN Regression Prediction:", skl_reg_prediction)
```

Figure 8: Code Snippet for Scikit-Learn Regression Prediction

```python
def sklearn_knn_classification(clf_data, clf_query):
    # Initialize the KNN classifier with 3 nearest neighbors
    knn_clf = KNeighborsClassifier(n_neighbors=3)

    # Fit the model on the training data; use all but the last column as features and the last column as the target
    knn_clf.fit(clf_data.iloc[:,:-1], clf_data.iloc[:,-1])

    # Predict the class for the provided query and return the first (and likely only) prediction
    skl_clf_prediction = knn_clf.predict(clf_query)

    return skl_clf_prediction
```

Figure 9: Code Snippet for Scikit-Learn Classification Function

```python
# Scikit-learn KNN Classification
skl_clf_prediction = sklearn_knn_classification(clf_data, clf_query)

print("Scikit-learn KNN Classification Prediction:", skl_clf_prediction)
```

Figure 10: Code Snippet for Scikit-Learn Classification Prediction

- Custom KNN Regression Prediction: 128.24666666666667
- Scikit-learn KNN Regression Prediction: [128.24666667]

- Custom KNN Classification Prediction: 0
- Scikit-learn KNN Classification Prediction: [0]
- Custom KNN Classification Prediction: 0
- Scikit-learn KNN Classification Prediction: [0]
- Prediction for weighted KNN: 1

Q3.5

Complete the `weighted_mode` and `knn_weighted` functions in the provided skeleton code to classify whether a 15-year-old likes pineapple or not. For the `KNeighborsClassifier` assign weights proportional to the inverse of the distance from the query point and then classify.

```python
from collections import defaultdict

def weighted_mode(labels, weights):
    # Initialize a defaultdict to store the sum of weights for each label
    sum_weighted_labels = defaultdict(int)

    # Iterate through each label in the labels list and Sum the weights for each label and store in the defaultdict
    for l in labels:
        sum_weighted_labels[l] += weights[l]

    # Determine the label with the maximum sum of weights
    max_weighted_label = max(sum_weighted_labels, key=sum_weighted_labels.get)

    # Return the label that has the highest sum of weights
    return max_weighted_label
```

Figure 11: Code Snippet for `weighted_mode` Function

```python
def knn_weighted(data, query, k, distance_fn, choice_fn, weights):
    neighbor_distances_and_indices = []

    # Calculate the distance between the query example and all the examples in the data.
    index=0
    for val in data.iloc[:,0]:
        neighbor_distances_and_indices.append(list([euclidean_distance(val, query), index]))
        index += 1

    # Sort the distances and return the labels of the k nearest neighbors.
    neighbor_distances_and_indices.sort()

    # Pick the first k entries from the sorted collection
    k_nearest_neighbors = neighbor_distances_and_indices[:k]

    # Get the labels of the selected k entries
    k_nearest_labels = []
    for ls in k_nearest_neighbors:
        k_nearest_labels.append(data.iloc[ls[1], 1])

    # Apply the weighted mode function and return nearest neighbors too
    return choice_fn(k_nearest_labels, weights)
```

Figure 12: Code Snippet for `knn_weighted` Function

```python
weights = {0: 1, 1: 2}

# Query for whether a 15-year-old likes pineapple or not. The classification should be 1 as this exact sample is present in the dataset
# but because of unbalanced dataset this will be predicted as class 0.

clf_query = np.array([[15]])  # reshape to fit scikit-learn requirements

# Custom KNN Prediction
custom_clf_prediction = knn(clf_data, clf_query, k=3, distance_fn=euclidean_distance, choice_fn=mode)

# Scikit-learn KNN Classification
skl_clf_prediction = sklearn_knn_classification(clf_data, clf_query)

print("Custom KNN Classification Prediction:", custom_clf_prediction)
print("Scikit-learn KNN Classification Prediction:", skl_clf_prediction)

clf_prediction = knn_weighted(clf_data, clf_query, k=3, distance_fn=euclidean_distance, choice_fn=weighted_mode, weights=weights)
print("Prediction for weighted KNN:", clf_prediction)
```

Figure 13: Code Snippet for Weighted Classification Prediction

Q3.6

In Scikit-learn, several algorithms involve randomness in their operation and therefore provide a `random_state` parameter to ensure reproducibility of results by controlling the seed of the random number generator. Do `KNeighborsRegressor` and `KNeighborsClassifier` have it?

Yes, both KNeighborsRegressor and KNeighborsClassifier in scikit-learn do have a parameter for controlling the seed of the random number generator to ensure reproducibility of results. This parameter is typically named random_state. Example: `regressor = KNeighborsRegressor(n_neighbors=3, random_state=42)`

Question 5: Data Visualization and Model Selection

Consider the dataset found here. It is a modified version of the popular Heart Disease Dataset originally comprising 303 instances with 13 features, and serves as a multivariate resource for classification tasks in the realm of health and medicine, featuring categorical, integer, and real feature types.

Q5.1

Display basic information about the dataset, including the data types of each column followed by a summary of the dataset's statistics, including count, mean, standard deviation and max values for each column.

```
C:\Users\mapel\OneDrive - University of Waterloo\ECE657\A2>py ece657_a2_q5.py
Basic Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 549 entries, 0 to 548
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   cp       549 non-null    int64
 1   ca       549 non-null    int64
 2   thalach  549 non-null    int64
 3   oldpeak  549 non-null    float64
 4   thal     549 non-null    int64
 5   target   549 non-null    int64
dtypes: float64(1), int64(5)
memory usage: 25.9 KB
None

First few rows of the dataset:
   cp  ca  thalach  oldpeak  thal  target
0   0   3      145      6.2     3       0
1   0   1      141      2.8     3       0
2   0   1      156      0.1     3       0
3   0   2       90      1.0     1       0
4   0   2      165      1.0     3       0

Summary Statistics:
               cp          ca     thalach     oldpeak        thal      target
count  549.000000  549.000000  549.000000  549.000000  549.000000  549.000000
mean     0.561020    1.078324  140.845173    1.492532    2.504554    0.094718
std      0.942901    1.039048   23.077379    1.288169    0.685138    0.293092
min      0.000000    0.000000   71.000000    0.000000    0.000000    0.000000
25%      0.000000    0.000000  125.000000    0.300000    2.000000    0.000000
50%      0.000000    1.000000  143.000000    1.200000    3.000000    0.000000
75%      1.000000    2.000000  160.000000    2.400000    3.000000    0.000000
max      3.000000    4.000000  195.000000    6.200000    3.000000    1.000000
```

Figure 1: Basic information about the dataset

Q5.2

Calculate and display the proportion of each unique value in the 'target' column. This analysis will help you understand the frequency of each category within the target variable, providing insights into the dataset's balance. Is the dataset imbalanced?

```
Distribution of target variable:
target
0    497
1     52
Name: count, dtype: int64

Proportion of each unique value in the target variable:
target
0    0.905282
1    0.094718
Name: proportion, dtype: float64
```

Figure 2: Proportion of each unique value in the `target`

Because the difference in proportions of the target values is too big, we can call the dataset is imbalanced.

Q5.3

Plot histograms for all numerical attributes in the dataset. You will identify all numerical columns and generate histograms with specified bin sizes to visually assess the distribution of these attributes. This visualization is vital for spotting any skewness or outliers in the data and understanding the distribution of numerical variables. Which of the features are skewed? Also, mention if they are left-skewed or right-skewed.

```
Skewness of numerical features:
cp         1.369990
ca         0.587301
thalach   -0.327264
oldpeak    0.790790
thal      -1.212341
target     2.775679
dtype: float64
```

Figure 3: Skewness of numerical values

From the values for skewness, we can say that features `cp`, `ca`, `oldpeak`, and `target` are all right-skewed. `thal` is a left-skewed feature whereas `thalach` is approcimately symmetric.

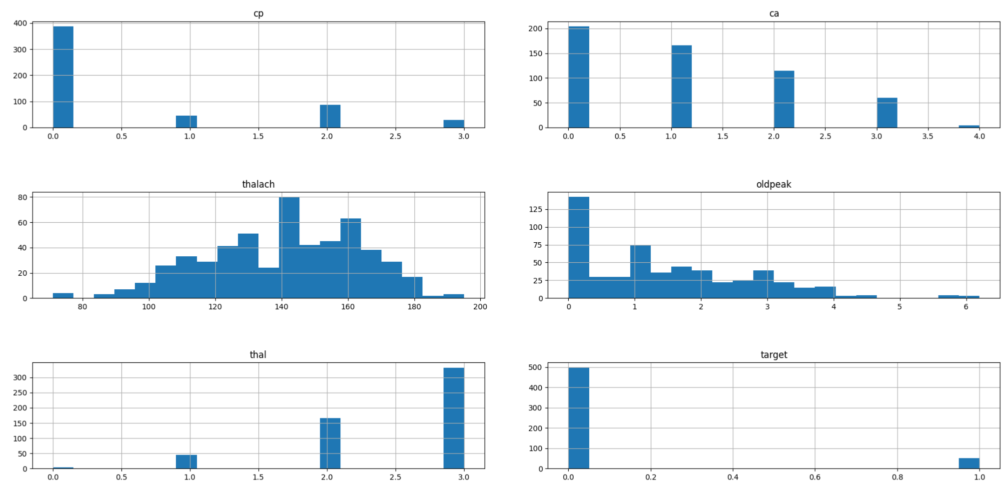


Figure 4: Histograms for attributes in the dataset

Q5.4

Generate a heatmap to visualize the correlations between different variables. Understanding these correlations is crucial for identifying relationships between variables, which can inform feature selection and predictive modelling strategies. Is there a high correlation with features or with the target variable?

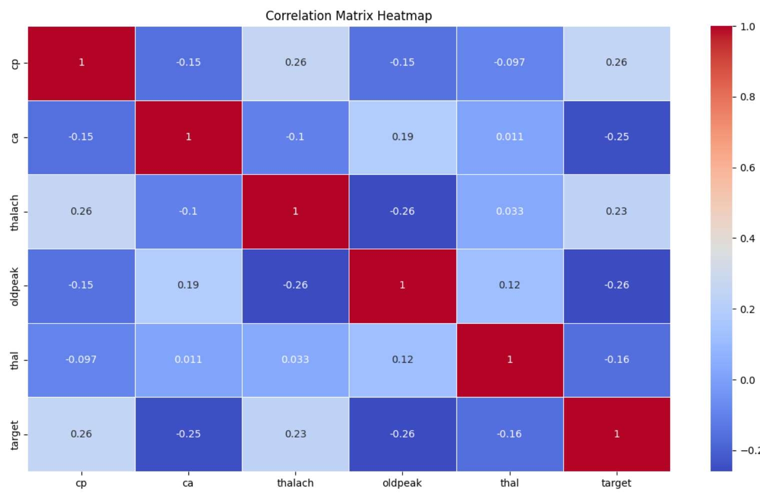


Figure 5: Correlation Matrix Heatmap

When we check the correlation values on the matrix, it can be concluded that the correlations between features and the target variable are relatively weak.

Q5.5

Scale the features of the dataset using Python and pandas along with scikit-learn's `StandardScaler`. Remove the `target` column and apply scaling to the remaining features to standardize them, converting the scaled data back into a `DataFrame` with the original column names. Why is this step important? Report the mean and standard deviation of the scaled features.

Scaling is important to ensure that each feature contributes to the model equally. Otherwise, some features may end up dominate the others.

```
Mean of scaled features:
cp          2.912060e-17
ca         -2.264936e-17
thalach    -1.132468e-16
oldpeak     9.059743e-17
thal       -6.471245e-17
dtype: float64

Standard deviation of scaled features:
cp         1.000912
ca         1.000912
thalach    1.000912
oldpeak    1.000912
thal       1.000912
dtype: float64
```

Figure 6: The mean and standard deviation of the scaled features

Q5.6

Assign the scaled features to X and the target variable to y. Specify the size of the test set to be 20% of the entire dataset, ensure the data is split in a way that maintains the proportion of classes in both training and testing sets by stratifying on y, and set a random state to 25 for reproducibility. Print the shapes of the training and testing sets.

```
Shapes of training and testing sets:
X_train: (439, 5)
X_test: (110, 5)
y_train: (439,)
y_test: (110,)
```

Figure 7: The shapes of training and testing sets

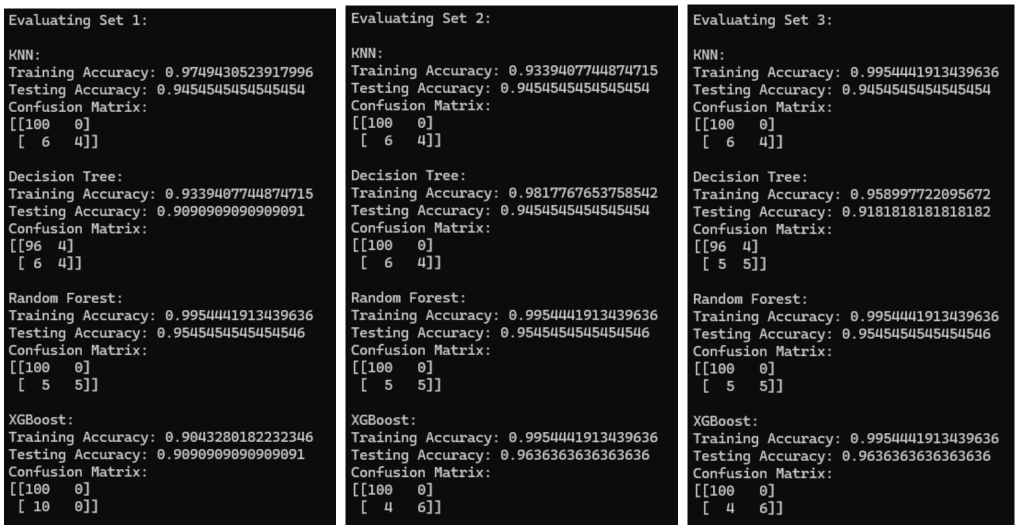
Q5.7

Table 2: Classifier Parameters for Different Sets

Classifier	Set 1	Set 2	Set 3
KNN	n_neighbors=3, weights='uniform'	n_neighbors=10, weights='uniform', p=2	n_neighbors=15, weights='distance', p=1
Decision Tree	max_depth=3, min_samples_split=2, random_state: 25	max_depth=10, min_samples_leaf=2, random_state: 25	criterion='entropy', max_depth=20, min_samples_leaf=4, random_state: 25
Random Forest	n_estimators=50, max_features='auto', random_state: 25	n_estimators=200, max_depth=10, max_features='log2', random_state: 25	n_estimators=350, max_depth=20, min_samples_split=3, random_state: 25
XGBoost	learning_rate=0.01, max_depth=3, n_estimators=50, random_state: 25	learning_rate=0.15, max_depth=10, n_estimators=150, random_state: 25	learning_rate=0.25, max_depth=15, n_estimators=250, random_state: 25

Figure 8: The table provided in the assignment description document

Evaluate various models with different hyperparameters listed in Table 2. Implement these models using sklearn for each set of parameters. Based on training and testing accuracy scores and confusion matrices, determine the best-performing model for health data. Additionally, provide insights into the performance of each model.



```
Evaluating Set 1:

KNN:
Training Accuracy: 0.9749430523917996
Testing Accuracy: 0.9454545454545454
Confusion Matrix:
[[100   0]
 [  6   4]]

Decision Tree:
Training Accuracy: 0.9339407744874715
Testing Accuracy: 0.9090909090909091
Confusion Matrix:
[[96  4]
 [ 6  4]]

Random Forest:
Training Accuracy: 0.9954441913439636
Testing Accuracy: 0.9545454545454546
Confusion Matrix:
[[100   0]
 [  5   5]]

XGBoost:
Training Accuracy: 0.9043280182232346
Testing Accuracy: 0.9090909090909091
Confusion Matrix:
[[100   0]
 [ 10   0]]
```

```
Evaluating Set 2:

KNN:
Training Accuracy: 0.9339407744874715
Testing Accuracy: 0.9454545454545454
Confusion Matrix:
[[100   0]
 [  6   4]]

Decision Tree:
Training Accuracy: 0.9817767653758542
Testing Accuracy: 0.9454545454545454
Confusion Matrix:
[[100   0]
 [  6   4]]

Random Forest:
Training Accuracy: 0.9954441913439636
Testing Accuracy: 0.9545454545454546
Confusion Matrix:
[[100   0]
 [  5   5]]

XGBoost:
Training Accuracy: 0.9954441913439636
Testing Accuracy: 0.9636363636363636
Confusion Matrix:
[[100   0]
 [  4   6]]
```

```
Evaluating Set 3:

KNN:
Training Accuracy: 0.9954441913439636
Testing Accuracy: 0.9454545454545454
Confusion Matrix:
[[100   0]
 [  6   4]]

Decision Tree:
Training Accuracy: 0.958997722095672
Testing Accuracy: 0.9181818181818182
Confusion Matrix:
[[96  4]
 [ 5  5]]

Random Forest:
Training Accuracy: 0.9954441913439636
Testing Accuracy: 0.9545454545454546
Confusion Matrix:
[[100   0]
 [  5   5]]

XGBoost:
Training Accuracy: 0.9954441913439636
Testing Accuracy: 0.9636363636363636
Confusion Matrix:
[[100   0]
 [  4   6]]
```

Figure 9: Results of the model evaluation

Set 2 - XGBoost stands out as the best-performing model (Even Set 3 - XGBoost has the same test accuracy, Set 2 has less max-depth):

- Testing Accuracy: 96.36
- Confusion Matrix: $\begin{bmatrix} 100 & 0 \\ 4 & 6 \end{bmatrix}$

It has a relatively balanced confusion matrix indicating that most of the test samples were correctly classified and handled class imbalances effectively; therefore, this model achieves the highest testing accuracy.

Other models:

- kNNs showed a good consistent performance with high training and testing accuracy for all sets.
- Decision trees had improvement in accuracy with deeper trees but can be prone to overfitting (in Set 2, the high accuracy of training).
- Random Forests seem to be more reliable when it comes the similarity of training and testing accuracies.

Mehmet Ali Pelit
21113622

Q5.8

What are the potential challenges that exist with working with real-world healthcare data for machine learning applications?

Using real healthcare data for machine learning is challenging. One may need to deal with issues like data quality problems, integrating complex data, and handling imbalanced classes. There are also ethical concerns about privacy and bias, and the need for clear explanations makes it more complex. Technical challenges such as figuring out which features to use, making sure models are accurate, and meeting regulations make it even harder. Solving these problems requires teamwork and different areas of expertise to build trustworthy and fair healthcare solutions.