DATS 6202

Machine Learning I

Prof. Amir Jafari

Heart Disease Prediction

Individual Report

Jiwoo Suh

Table of Contents

1. Introduction

2. Description of Individual Work & Portion

3. Results

3.1. EDA

3.2. K-fold Cross Validation

3.3. Random Forest

3.4. MLP

3.5. Other models

4. Summary and Conclusions

5. Percentage of Code

6. References

1. Introduction

Heart disease is one of the leading causes of death globally, and according to the Centers for Disease Control and Prevention (CDC), it is also one of the leading causes of death in the United States. In the US, heart disease accounts for approximately one in every four deaths, and it affects people of most races, including African Americans, American Indians and Alaska Natives, and white people. The prevalence of heart disease is largely due to factors such as high blood pressure, high cholesterol, smoking, diabetes, obesity, physical inactivity, and excessive alcohol consumption. Detecting and preventing these factors is therefore crucial in healthcare.

With recent advancements in computational developments, machine learning methods can now be applied to detect patterns from large datasets that can predict a patient's condition, including the likelihood of heart disease. The Behavioral Risk Factor Surveillance System (BRFSS) dataset, collected annually by the CDC, provides a Wealth of information on the health status of US residents. The dataset includes responses to questions about various health indicators and risk factors, including those related to heart disease. The purpose of this report is to explore the BRFSS dataset and to apply machine learning algorithms to predict the likelihood of heart disease based on selected variables.

2. Description of Individual Work and Portion

My team and I discussed and worked all together as a team to achieve our goal. In terms of deviding work, I was responsible of 1) Initial EDA for demographic variables, 2) K-fold cross validation in data preprocessing, 3) Modelling for Random Forest and MLP and Evaluation, 4) Other classifiers.

3. Results

3.1. EDA

Initially, we categorized all feauture to perform EDA based on the characteristic of each variable. I oversaw the feauture regarding to anthropometric factors including BMI, Sex, Race, Age Category, and Generation Health. For BMI, which is the only numerical variable in this category, I tried to see the distribution of it by using histogram and boxplot and compare the distribution by heart disease. For the rest of features, I checked its distribution by barchart and piechart to see the importance of features. As a result, I concluded that there is no significant difference between the group whether they have heart disease or not based on EDA as the dataset is imbalanced. Therefore, my team and I decided to perform feature selection step with a tool.

3.2. K-fold Cross Validation

I tried to train and evaluate models using k-fold cross validation with k=10. My initial code for trying k-fold cross validation with the base model which is Logistic Regression is below:

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
```

```
# Initialize k-fold
kfold = KFold(n_splits=10)

# Train and evaluate logistic regression model using k-fold cross validation
for train_idx, test_idx in kfold.split(X):
    X_train, y_train = X.iloc[train_idx], y.iloc[train_idx]
    X_test, y_test = X.iloc[test_idx], y.iloc[test_idx]

    # Initialize logistic regression model
    model = LogisticRegression()

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

    # Evaluate the model on the test data
    score = model.score(X_test, y_test)
    print(f"Accuracy: {score:.4f}")
```

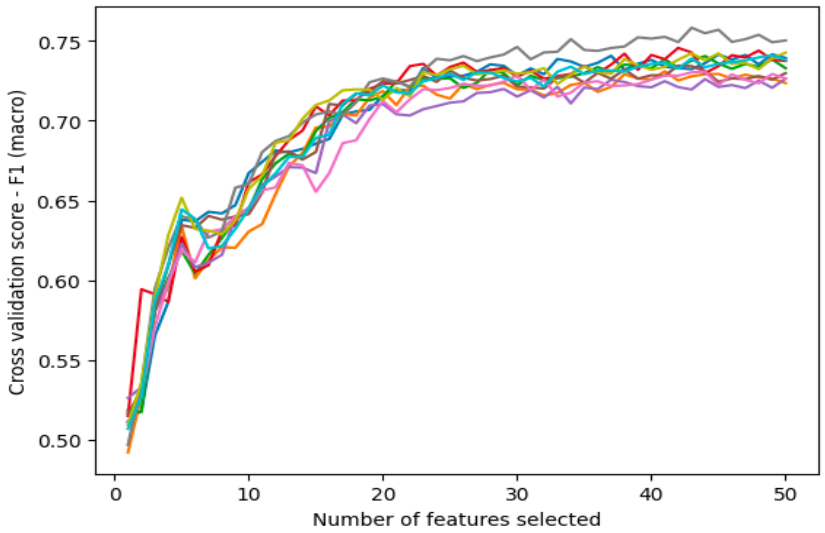
After, my team and I decided to put cross validation part inside the pipeline, so this code is not included in the final code.

3.3. Model - Random Forest

I tried to train Random Forest model. The number of features selected by the random forest algorithm was 43 including various demographic, lifestyle, and health-related factors such as BMI, smoking status, physical activity, and chronic conditions like asthma and diabetes.

The cross-validation scores obtained from the random forest model ranged between 0.719 and 0.758, which are lower than the logistic regression model. However, the scores are still above 0.7, indicating that the model is performing moderately well in predicting the presence of heart disease. It is also worth noting that the scores are relatively consistent across the 10 folds of cross-validation, which suggests that the model is not overfitting the data.

```
Number of features selected: 43

Selected features: ['BMI', 'PhysicalHealth', 'MentalHealth', 'SleepTime',
'Smoking_No', 'Smoking_Yes', 'AlcoholDrinking_Yes', 'Stroke_No', 'Stroke_Yes',
'DiffWalking_No', 'DiffWalking_Yes', 'Sex_Female', 'Sex_Male', 'AgeCategory_18-
24', 'AgeCategory_25-29', 'AgeCategory_30-34', 'AgeCategory_35-39',
'AgeCategory_40-44', 'AgeCategory_45-49', 'AgeCategory_50-54', 'AgeCategory_55-
59', 'AgeCategory_60-64', 'AgeCategory_65-69', 'AgeCategory_70-74',
'AgeCategory_75-79', 'AgeCategory_80 or older', 'Race_Black', 'Race_Hispanic',
'Race_White', 'Diabetic_No', 'Diabetic_Yes', 'PhysicalActivity_No',
'PhysicalActivity_Yes', 'GenHealth_Excellent', 'GenHealth_Fair',
'GenHealth_Good', 'GenHealth_Very good', 'Asthma_No', 'Asthma_Yes',
'KidneyDisease_No', 'KidneyDisease_Yes', 'SkinCancer_No', 'SkinCancer_Yes']

Cross-validation score:
[0.73234942 0.72785062 0.73768522 0.74276655 0.71956437 0.73420739 0.73035926
0.75826772 0.74224344 0.73894325]
```



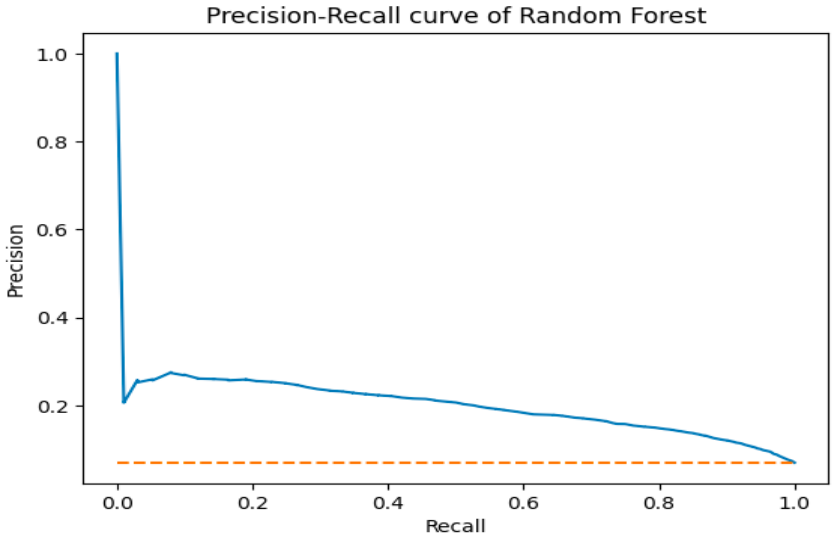
Upon the cross validation given below, random forest did not experience a dramatic drop and the performance is better when more features are selected.

The precision, recall, f1-score, and support were reported for both the training and testing datasets.

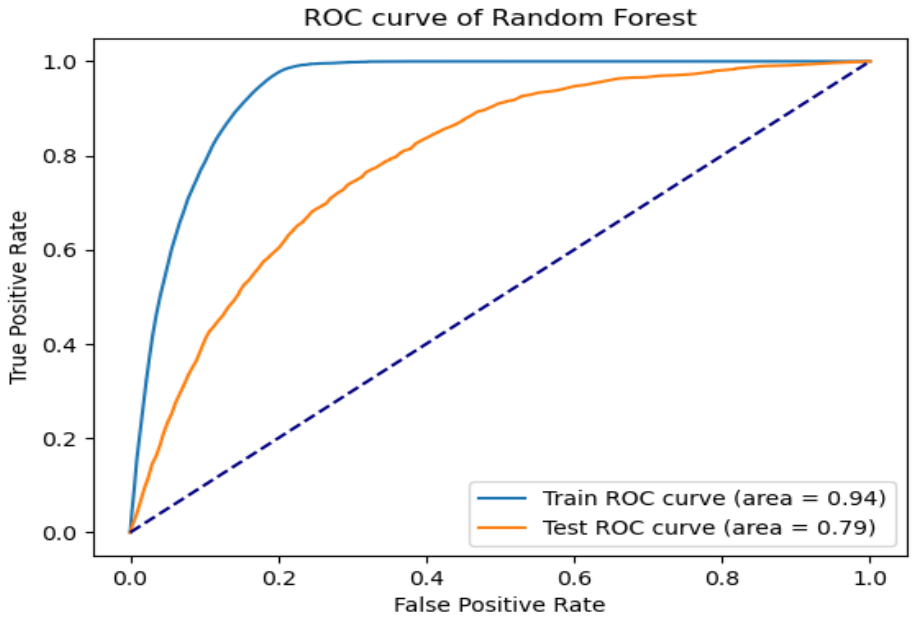
```
Train:
              precision    recall  f1-score   support

           0       1.00      0.73      0.84    167914
           1       0.21      1.00      0.35     12462

    accuracy                           0.74    180376
   macro avg       0.61      0.86      0.60    180376
Weighted avg       0.95      0.74      0.81    180376

---------------------------------
ROC AUC score: 0.9394879628419884
---------------------------------

Test:
              precision    recall  f1-score   support

           0       0.97      0.70      0.82     41926
           1       0.16      0.74      0.26      3169

    accuracy                           0.71     45095
   macro avg       0.57      0.72      0.54     45095
Weighted avg       0.92      0.71      0.78     45095

---------------------------------
                  Predicted No Disease  Predicted Disease
Actual No Disease                 29480              12446
Actual Disease                      829               2340
```

```
---------------------------------
ROC-AUC score: 0.7897017520854901
---------------------------------
Balanced accuracy: 0.7207734579071057
---------------------------------
```

The Random Forest classifier achieved an accuracy of 74% on the training dataset and an accuracy of 71% on the testing dataset. The precision of predicting the presence of heart disease was 16% and the recall was 74%.
The confusion matrix shows that the model predicted 29,480 true negatives (i.e., correctly predicted 'No Disease') and 2,340 true positives (i.e., correctly predicted 'Disease'). On the other hand, the model predicted 829 false negatives (i.e., predicted 'No Disease' when the actual class was 'Disease') and 12,446 false positives (i.e., predicted 'Disease' when the actual class was 'No Disease'). The balanced accuracy was reported as 0.72. Balanced accuracy considers the fact that the dataset is imbalanced and gives equal weight to both classes. This score is closer to the recall of the minority class than the overall accuracy score, indicating that the model is performing better at correctly identifying the presence of heart disease than the absence of heart disease.



The Precision-Recall curve is a visual representation of the trade-off between the precision and recall of a binary classifier like the Random Forest model. Precision measures the proportion of true positives (correctly predicted positive cases) among all predicted positive cases, while recall measures the proportion of true positives among all actual positive cases. The recall score was high for both classes, indicating that the model was able to identify most of the instances of both classes. However, the precision rate is low, indicating that the model produces many false positives when predicting heart disease.



The precision of predicting the absence of heart disease was 97% and the recall was 74%. The ROC-AUC score, which is a measure of the classifier's ability to distinguish between positive and negative classes, was 0.792. This could indicate that the model has a moderate ability to discriminate between the two classes, However the imbalance in the dataset should be considered.

3.4. Model - Multi Layer Perceptron

The Multi-Layer Perceptron (MLP) is a type of artificial neural network that consists of multiple layers of interconnected nodes or neurons. Each neuron in a layer receives input from the neurons in the previous layer and processes the input using an activation function to produce an output.

The MLP can be used for both classification and regression tasks. For classification tasks, the output layer of the MLP consists of multiple neurons, each corresponding to a different class label. The output of the MLP is a probability distribution over the class labels, and the class with the highest probability is predicted as the output.

For Multi Layer Perceptron, I used a different pipeline since the grid search is not working for it. Initially, for feature selection for MLP, we tried chi-squared test to see the statistical significance for each feature. With the 40 percent of features to keep from chi-squared test, the selected features were Smoking', 'Stroke', 'DiffWalking', 'Sex', 'AgeCategory', 'Diabetic', 'PhysicalActivity', 'GenHealth', 'KidneyDisease', 'SkinCancer', 'BMI'.

```
from sklearn.feature_selection import SelectPercentile, chi2

FeatureSelection = SelectPercentile(score_func=chi2, percentile=40)
X_new = FeatureSelection.fit_transform(x_train, y_train)
print('X_new shape is', X_new.shape)
print('selected features are:', FeatureSelection.get_support())
```

```
print('number of selected features:', FeatureSelection.get_support().sum())
print('features names are', FeatureSelection.get_feature_names_out())

selectedf40 = ['Smoking', 'Stroke', 'DiffWalking', 'Sex', 'AgeCategory',
'Diabetic',
      'PhysicalActivity', 'GenHealth', 'KidneyDisease',
      'SkinCancer', 'BMI']
x_clean2 = x_clean[selectedf40]
x_encoded2 = pd.get_dummies(x_clean2)
```

However, it turned out that the selected feature with this step could not improve the result of MLP model. Therefore, we decided to keep all features.

The input layer is a flatten layer that flattens the input data. The hidden layers consist of Dense layers with ReLU activation function and Dropout layers for regularization. The output layer is a Dense layer with a sigmoid activation function that produces a binary classification output. The model is compiled with the Adam optimizer, binary cross-entropy loss function, and F1 score as the evaluation metric. Adam optimizer is a stochastic gradient descent optimizer that is commonly used in deep learning. It is an adaptive learning rate optimization algorithm that combines the advantages of two other popular optimization algorithms: AdaGrad and RMSProp.

Adam optimizer calculates adaptive learning rates for each parameter based on estimates of the first and second moments of the gradients. The first moment is the average of the gradients, and the second moment is the average of the squared gradients. The algorithm uses these moments to update the parameters, which leads to faster convergence and better generalization.

Binary cross-entropy loss function is a popular loss function used for binary classification problems. It is used to measure the difference between the predicted probabilities and the actual labels. The formula for binary cross-entropy loss is:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Where N is the number of samples, y is the actual label of the i-th sample (either 0 or 1) and p(y) is the predicted probability of the point being 1 for all N points.

The data is oversampled by the Synthetic Minority Over-sampling Technique (SMOTE) method to balance the dataset as I wanted to see the result from the train dataset with oversampling. Before oversampling, I split our test dataset into a test set and validation set with the test size of 0.5. After data splitting, the size of test set and validation set is below here:

```
x_test: (5636, 50)
x_val: (5637, 50)
y_test:(5636, 1)
y_val:(5637, 1)
```

The reason I used the SMOTE method is to see the difference between oversampling and undersampling. Since there is no significant difference in result from the f1-score, which is our evaluation, I decided to see how MLP is doing with oversampled data.

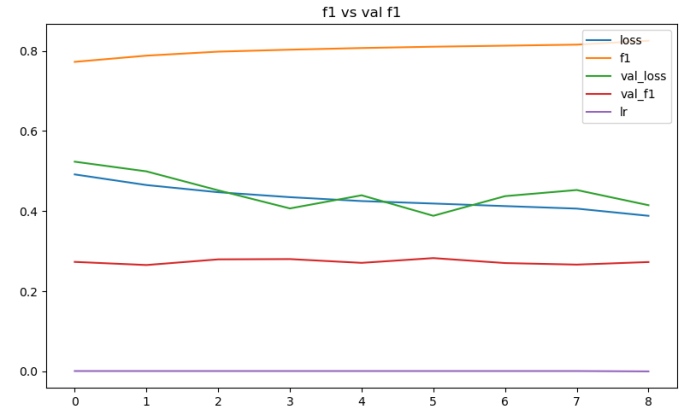
```
Before OverSampling, counts of label '1': [12462]
Before OverSampling, counts of label '0': [167914]

After OverSampling, the shape of train_X: (335828, 50)
After OverSampling, the shape of train_y: (335828, 1)

After OverSampling, counts of label '1': [167914]
After OverSampling, counts of label '0': [167914]
```

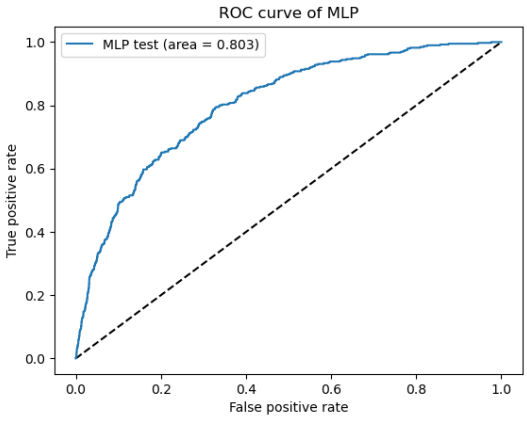
Therefore, after oversampling, I have 167,914 counts for heart disease ‘Yes’ from 12,462.
The model is then trained for 25 epochs with early stopping and learning rate reduction on plateau callbacks. The EarlyStopping callback monitors the validation loss and stops the training process when the loss does not improve for a certain number of epochs, while the ReduceLROnPlateau callback reduces the learning rate when the validation loss does not improve for a certain number of epochs. These callbacks help prevent overfitting and improve generalization.

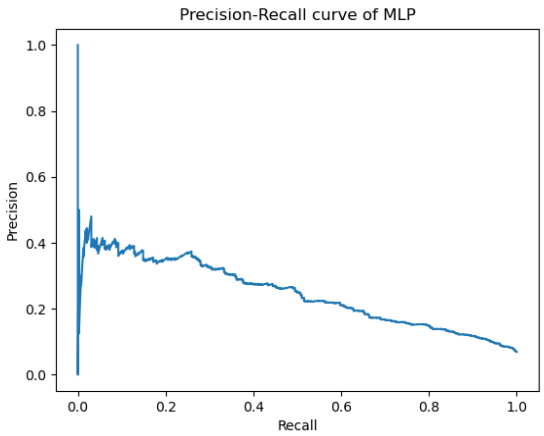
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 50)                0

 dense_1 (Dense)             (None, 256)               13056

 dropout (Dropout)           (None, 256)               0

 dense_2 (Dense)             (None, 256)               65792

 dropout_1 (Dropout)         (None, 256)               0

 dense_3 (Dense)             (None, 256)               65792

 dropout_2 (Dropout)         (None, 256)               0

 dense_4 (Dense)             (None, 128)               32896

 batch_normalization (BatchN  (None, 128)              512
 ormalization)

 dense_5 (Dense)             (None, 1)                 129

=================================================================
Total params: 178,177
Trainable params: 177,921
Non-trainable params: 256
```

The MLP model with 4 hidden layers and an output layer. The input layer is a Flatten layer that flattens the input data. The hidden layers consist of Dense layers with ReLU activation function and Dropout layers for regularization. The output layer is a Dense layer with a sigmoid activation function that produces a binary classification output.
The model has 178,177 parameters, which is relatively small for the number of layers and neurons used. The model summary shows that the majority of the parameters (130,560) are in the first hidden layer. The model is compiled with the Adam optimizer, binary cross-entropy loss function, and F1 score as the evaluation metric. The F1 score is manually calculated to compare the performance of this model with the previous model since the test set was a single batch.

```
10495/10495 [==============================] - 24s 2ms/step - loss: 0.4918 - f1: 0.7727 - val_loss: 0.5236 - val_f1: 0.2734 - lr: 0.0010
Epoch 2/25
10495/10495 [==============================] - 23s 2ms/step - loss: 0.4653 - f1: 0.7882 - val_loss: 0.4994 - val_f1: 0.2656 - lr: 0.0010
Epoch 3/25
10495/10495 [==============================] - 22s 2ms/step - loss: 0.4473 - f1: 0.7982 - val_loss: 0.4521 - val_f1: 0.2797 - lr: 0.0010
Epoch 4/25
10495/10495 [==============================] - 22s 2ms/step - loss: 0.4350 - f1: 0.8032 - val_loss: 0.4069 - val_f1: 0.2804 - lr: 0.0010
Epoch 5/25
10495/10495 [==============================] - 22s 2ms/step - loss: 0.4253 - f1: 0.8072 - val_loss: 0.4397 - val_f1: 0.2710 - lr: 0.0010
Epoch 6/25
10495/10495 [==============================] - 22s 2ms/step - loss: 0.4191 - f1: 0.8105 - val_loss: 0.3885 - val_f1: 0.2829 - lr: 0.0010
Epoch 7/25
10495/10495 [==============================] - 22s 2ms/step - loss: 0.4127 - f1: 0.8131 - val_loss: 0.4376 - val_f1: 0.2705 - lr: 0.0010
Epoch 8/25
10495/10495 [==============================] - 22s 2ms/step - loss: 0.4065 - f1: 0.8157 - val_loss: 0.4529 - val_f1: 0.2666 - lr: 0.0010
Epoch 9/25
10495/10495 [==============================] - 24s 2ms/step - loss: 0.3885 - f1: 0.8255 - val_loss: 0.4150 - val_f1: 0.2731 - lr: 1.0000e-
```







```
>>> loss, accuracy = model.evaluate(x_test,y_test)
705/705 [==============================] - 1s 1ms/step - loss: 0.3948 - f1: 0.2676
```

The model achieves an F1 score of 0.8 on the test data, but only 0.26 for the validation set. This large difference between the F1 scores suggests that the model may not generalize well to new data, and more work is needed to improve its performance.

Overall, the model architecture and hyperparameters seem appropriate, and the callbacks help prevent overfitting. However, the poor generalization of the model indicates that further analysis and optimization are required, such as fine-tuning the hyperparameters, increasing the size of the dataset, or changing the model architecture.

3.5. Other Classifiers

In this section, I would like to address other models that I also tried but could not obtain better result. The first classification report it from train set, and the other report is from test set. To sum up, XGBoost has F1-score of 0.55, and 0.57 for AdaBoost. AdaBoost has the same F1-score with Logistic Regression. However, we decided to keep Logistic Regression because the speed of training was better with Logistic Regression.

3.5.1. XGBoost

```
Classifier: XGBoost
Selected features: ['Smoking_No', 'Stroke_No', 'DiffWalking_No', 'Sex_Female',
'AgeCategory_18-24', 'AgeCategory_25-29', 'AgeCategory_30-34', 'AgeCategory_35-
39', 'AgeCategory_40-44', 'AgeCategory_45-49', 'AgeCategory_50-54',
'AgeCategory_60-64', 'AgeCategory_65-69', 'AgeCategory_70-74', 'AgeCategory_75-
79', 'AgeCategory_80 or older', 'Diabetic_Yes', 'GenHealth_Excellent',
'GenHealth_Fair', 'GenHealth_Good', 'GenHealth_Very good', 'KidneyDisease_No',
'SkinCancer_No']
Number of features selected: 23
              precision    recall  f1-score   support

           0       0.98      0.72      0.83    167914
           1       0.18      0.81      0.29     12462

    accuracy                           0.72    180376
   macro avg       0.58      0.77      0.56    180376
weighted avg       0.93      0.72      0.79    180376

---------------------------------
ROC AUC score: 0.8391825221352976
---------------------------------
              precision    recall  f1-score   support

           0       0.98      0.71      0.82      5246
           1       0.17      0.79      0.28       390

    accuracy                           0.72      5636
   macro avg       0.57      0.75      0.55      5636
weighted avg       0.92      0.72      0.79      5636

---------------------------------
                   Predicted No Disease  Predicted Disease
Actual No Disease                  3735               1511
Actual Disease                       83                307
---------------------------------
ROC-AUC score: 0.8324097969637428
```

```
---------------------------------
---------------------------------
Balanced accuracy: 0.7495752563613791
```

3.5.2. AdaBoost

```
Classifier: AdaBoost
Selected features: ['BMI', 'PhysicalHealth', 'MentalHealth', 'SleepTime',
'Smoking_Yes', 'AlcoholDrinking_Yes', 'Stroke_No', 'Stroke_Yes',
'DiffWalking_No', 'Sex_Male', 'AgeCategory_18-24', 'AgeCategory_25-29',
'AgeCategory_30-34', 'AgeCategory_35-39', 'AgeCategory_40-44', 'AgeCategory_45-
49', 'AgeCategory_50-54', 'AgeCategory_55-59', 'AgeCategory_60-64',
'AgeCategory_65-69', 'AgeCategory_70-74', 'AgeCategory_75-79', 'AgeCategory_80
or older', 'Race_Asian', 'Race_Black', 'Diabetic_Yes', 'GenHealth_Excellent',
'GenHealth_Fair', 'GenHealth_Good', 'GenHealth_Poor', 'GenHealth_Very good',
'Asthma_No', 'KidneyDisease_Yes', 'SkinCancer_No']
Number of features selected: 34
              precision    recall  f1-score   support

           0       0.98      0.74      0.84    167914
           1       0.18      0.79      0.29     12462

    accuracy                           0.74    180376
   macro avg       0.58      0.76      0.57    180376
weighted avg       0.92      0.74      0.80    180376

---------------------------------
ROC AUC score: 0.835853964117905
---------------------------------
              precision    recall  f1-score   support

           0       0.98      0.74      0.84      5246
           1       0.18      0.78      0.29       390

    accuracy                           0.74      5636
   macro avg       0.58      0.76      0.57      5636
weighted avg       0.92      0.74      0.80      5636

---------------------------------
                  Predicted No Disease  Predicted Disease
Actual No Disease                 3859               1387
Actual Disease                      86                304
---------------------------------
ROC-AUC score: 0.838058056443493
---------------------------------
---------------------------------
Balanced accuracy: 0.7575476309178177
---------------------------------
```

3.6. Evaluation

To evaluate the performance of the machine learning models, several metrics have been used in the project. The metrics used in the code include cross-validation score, classification report, ROC AUC score, confusion matrix, precision-recall curve, and ROC curve.

In an imbalanced dataset, there is a significant difference in the number of samples between the majority class and the minority class. This means that a model can achieve high accuracy by simply predicting the majority class for all samples, but it may perform poorly on the minority class, which is the class of interest. Therefore, accuracy is not a suitable metric for evaluating model performance in an imbalanced dataset.

On the other hand, precision, recall, and F1-score are metrics that take into account the number of true positives, false positives, and false negatives, and are more appropriate for evaluating model performance in an imbalanced dataset. Precision measures the proportion of true positive predictions among all positive predictions, while recall measures the proportion of true positive predictions among all actual positive samples. F1-score is a harmonic mean of precision and recall and gives equal Weight to both metrics. In an imbalanced dataset, these metrics are particularly important because they help to evaluate how Well the model is performing on the minority class.

$$\mathrm{F1}_{macro} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{TP}}{\mathrm{TP} + \frac{1}{2}(\mathrm{FP} + \mathrm{FN})}$$

For F1-score to evaluate MLP models, I decided to compute recall and precision manually since the built in metrics are removed from Keras because of the probability for misleading from a batch-wise computation. However, since the test dataset is a single batch, I computed F1-score manually to compare the performance for MLP with previous models.

```
def f1(y_true, y_pred):
    def recall(y_true, y_pred):
        """Recall metric.

        Only computes a batch-wise average of recall.

        Computes the recall, a metric for multi-label classification of
        how many relevant items are selected.
        """
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
        recall = true_positives / (possible_positives + K.epsilon())
        return recall

    def precision(y_true, y_pred):
        """Precision metric.

        Only computes a batch-wise average of precision.

        Computes the precision, a metric for multi-label classification of
        how many selected items are relevant.
        """
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
        return precision
    precision = precision(y_true, y_pred)
    recall = recall(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

4. Summary and Conclusions

In conclusion, the heart disease prediction task using machine learning algorithms was explored in this study. Two popular classifiers were implemented by me, including random forest and multi-layer perceptron, and their performances were evaluated using F1 score as the metric.

Based on the results I obtained, random forest has an F1-score of 0.54, and 0.26 for MLP. I also tried XGBoost and AdaBoost but they did not provide any significant improvement on the test set in terms of F1-score compared to the four classifiers that my team and I tried earlier.

Despite achieving acceptable performances, the overall performance of the models does not give optimal results. Therefore, further analysis and optimization are required to improve the performance of the models. Possible avenues of exploration include fine-tuning the hyperparameters for each classifier and implementing dimension-reduction techniques in the dataset to eliminate irrelevant features and reduce the complexity of the models.

In summary, I could earn a hands-on experience with applying machine learning algorithm to real world data and learn the difficulty of data preprocessing with highly imbalanced data. This study serves as a useful starting point for exploring machine learning algorithms in heart disease prediction, but there is still room for improvement and optimization. With further investigation, it is possible to develop more accurate and reliable models that can be useful in clinical decision-making and improving outcomes.

5. Code Percentage

According to the calculation example in the instruction, around 60% is my own code.

6. References

- Neural Network Design (2nd Ed), by Martin T Hagan, ISBN 0971732116
- https://www.cdc.gov/brfss/annual_data/annual_2020.html
- https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease
- Diederik P. Kingma and Jimmy Lei Ba. Adam : A method for stochastic optimization. 2014. arXiv:1412.6980v9
- How to Calculate Precision, Recall, and F-Measure for Imbalanced Classification: https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/