Hong Kong University of Science and Technology

COMP 4211: Machine Learning Spring 2025

Programming Assignment 1

Andreas Wold Sløgedal
Student ID: 21196840

Contents

Introduction

The purpose of this programming assignment is to apply the concepts learned in COMP4211 to explore and analyze a provided dataset (`data.csv`) using machine learning techniques. This assignment involves training linear regression, logistic regression and neural network models to make predictions based on features in the dataset. The models will be evaluated using model accuracy, F1-score and training time to identify the most effective model structure. The goal is to develop predictive models that generalize well to unseen data while analyzing the impact of different modeling techniques.

1 Part 1: Data Exploration and Preparation

Data exploration and preparation are essential steps in machine learning because it makes sure the dataset is clean and structured before modeling. Crucial steps involve understanding feature distribution, recognizing missing values and identifying correlation between features.

1.1 [Q1] Dataset Overview

Using `pd.shape` I am able to examine the structure of the dataset. The dataset contains 891 rows and 11 columns, where the number of rows corresponds to the number of individuals, and the number of columns the number of attributes assigned to each individual. Furthermore, `df.size` identifies the total number of elements in the dataset. This number is beneficial to examine, as it gives the total count of all data points, including missing values.

Missing values in a dataset are important to identify and process according to their structure. Using `df.isna().sum()` I sum over the total number of missing values for each attribute. As a result, I identify that `c2` is missing 177 values, `c6` is missing 687 values, and `c7` is missing 2 values. This corresponds to a proportion of 19.87%, 77.10% and 0.22% respectively.

Before training regression and classification models it is necessary to handle NaN (Not a Number) values to ensure data quality and prevent bias. `c2` has a moderate proportion of missing values (19.87%), which could distort model training. Hence, it could be reasonable to substitute missing values with mean or median imputation. Furthermore, `c6` has a much higher proportion of missing values (77.10%). Using such a large percentage of missing values when training models could impact the model negatively, and it could be beneficial to drop the column depending on the importance of the attribute. Additionally, one could impute using KNN implementation, which considers relationship between other features to predict missing values. Lastly, `c3` has a very low proportion of missing values, hence the impact on model performance is relatively low.

1.2 [Q2] Feature Distribution

Identifying numerical and categorical features in the dataset is important before selecting models. Different models handle numerical and categorical features differently. Categorical features might for example need to be one-hot-encoded before training the model. Consequently, using `df.select_dtypes` I am able to identify numerical values, which are either float (float64) or integer (int64) values. As a result, I identify that `c2`, `c3`, `c4` and `reg_target` are numerical values. Using `df.dtypes` I identify that `c2`, and `reg_target` are continuous features, and that `c3` and `c4` are discrete. Furthermore, summarizing the statistical distribution is done using `pd.describe,` which shows important statistical measures such as mean and standard deviation. Ultimately, I visualize the distribution of the first-in-order numerical (feature `c2`) features using a box plot.

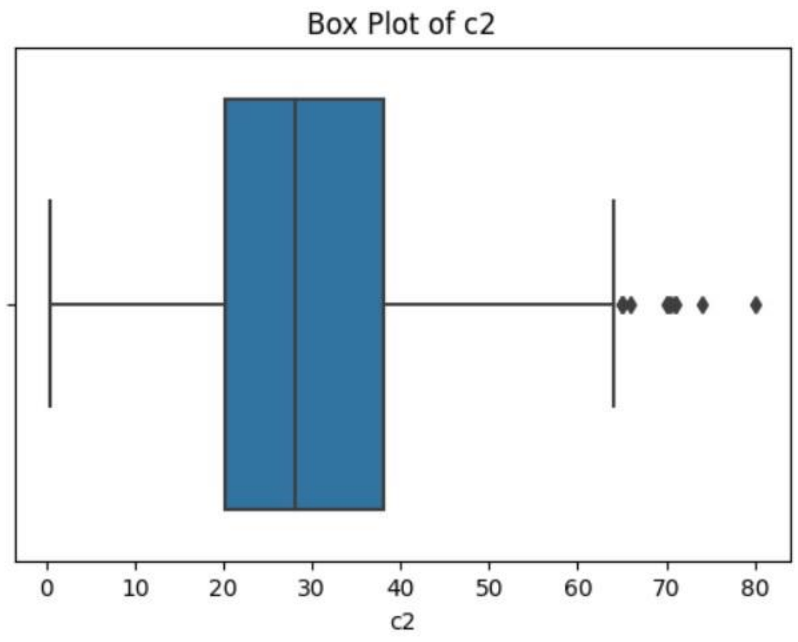


Figure 2.2.1: A box plot which illustrates the first-in-order numerical feature.

Categorical features are the remaining features in the dataset and are characterized by values that represent categories or groups, rather than continuous quantities. These values can be either nominal or ordinal. Identifying these values involves the same procedure as before, using `df.select_dtypes`. As a result, I have found the following distribution:

- **Categorical Features:**
  - **Binary:** `C1`
  - **Nominal:** `C5, C6, C7, C9`
  - **Ordinal:** `C8`
- **Numerical Features:**
  - **Continuous:** `C2,Reg_target`
  - **Discrete:** `C3,C4`
  - **Binary:** `Class_target`

Ultimately, I summarize the count of all categorical values and visualize the distribution of the first-in-order categorical feature using bar plots.

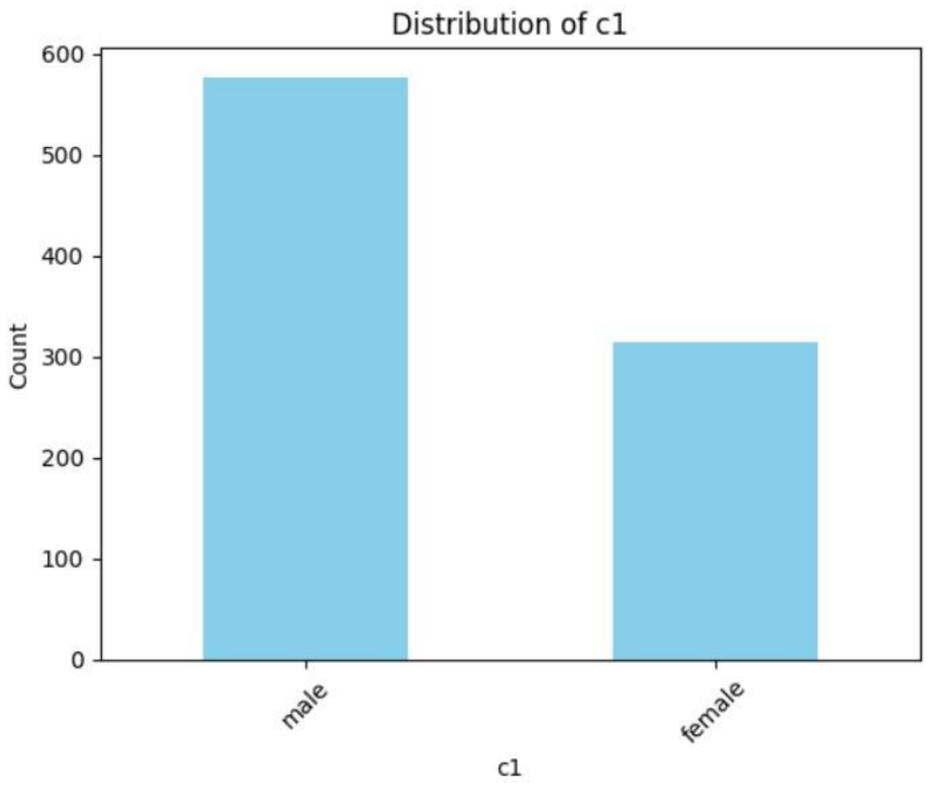
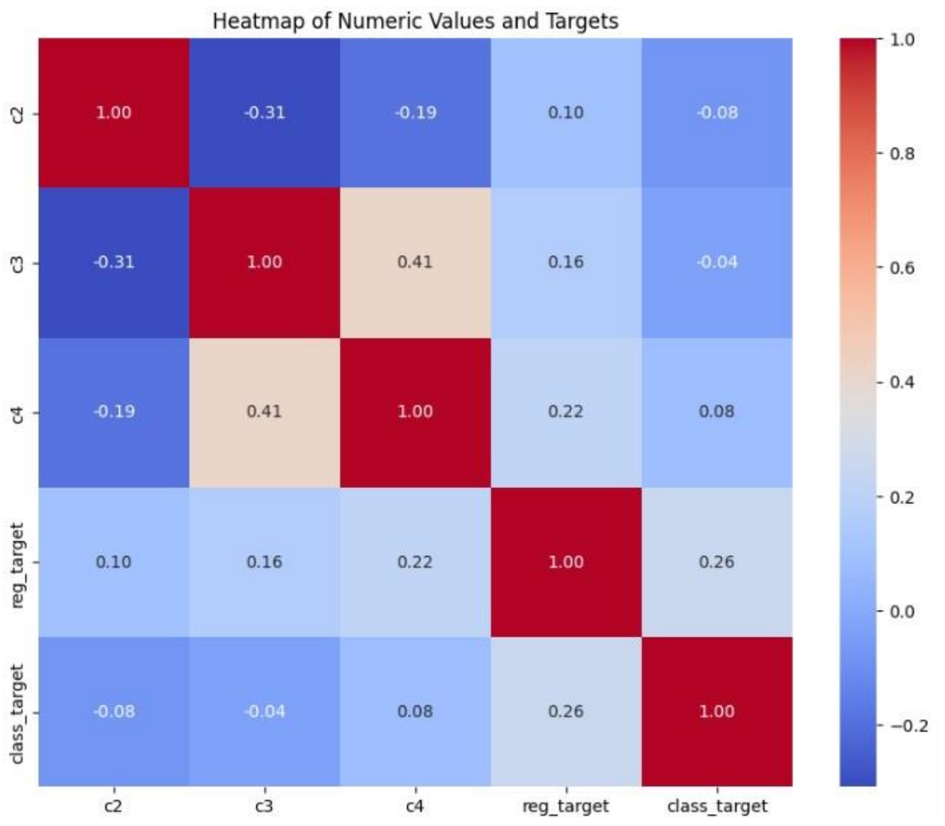


Figure 1.2.2: A bar plot illustrating the first-in-order categorical feature.

1.3 [Q3] Correlation Analysis

Heatmaps are strong visualization tools to identify which features have strong and weak correlations. When features are strongly correlated, it introduces multicollinearity and overfitting and reduces the model's ability to generalize to validation and test data. Weak correlation between features can lead to underfitting and reduced predictive power, ultimately increasing the complexity of the model without improving its performance. Consequently, it is necessary to handle strongly and weakly correlated features under data preprocessing before testing models.

Using `matplotlib` and `seaborn` I can construct a heatmap which shows the correlation between every two numerical features in the dataset.



	c2	c3	c4	reg_target	class_target
c2	1.00	-0.31	-0.19	0.10	-0.08
c3	-0.31	1.00	0.41	0.16	-0.04
c4	-0.19	0.41	1.00	0.22	0.08
reg_target	0.10	0.16	0.22	1.00	0.26
class_target	-0.08	-0.04	0.08	0.26	1.00

Figure 1.3.1: A correlation matrix of numeric features in the dataset.

As a result, it becomes easy to highlight both strong and weak correlations. As we can see, multicollinearity occurs due to the same features being both rows and columns in the correlation matrix. Multicollinearity needs to be considered during data preparation. Aside from that, there are no features that exhibit particularly high or low correlation with one another.

2 Part 2: Data Preprocessing Techniques

Data processing is an essential step in data science to ensure that variables are suitable for training models. Common techniques include handling missing data, feature engineering and one-hot-encoding. Ultimately, data preprocessing improves model performance because it enhances data quality and reduces bias.

2.1 [Q4] Handling Missing Values

Handling missing values is crucial for several reasons. Many models are unable to handle NaN values and will produce biased results. Which preprocessing technique to use highly depends on the number of missing values in each feature. If a feature contains many missing values, for example >70%, it is beneficial to remove the whole column to eliminate all potential bias. Hence, I dropped `c6` from the dataset. On the other hand, `c2` contains a moderate number of missing values, hence it would be more beneficial to keep the data in this column and impute the median value of the column for the missing values. When the percentage of missing values is very low (0.2% in this case), it is advantageous to use the most frequent value. It ensures the missing values are substituted with common values, which are less likely to manipulate the data.

2.2 [Q5] Normalization and Standardization

Regression and classification models often perform better when data is scaled because the scale of features might vary within the dataset. Features might have different units, which confuse the model and make it converge during the initial training process. Ultimately, we wish to optimize model performance with normalization and standardization.

Before processing the values in `c2` range from a minimum of 2 to a maximum of 54. After processing with `StandardScaler` range from approximately -2.10 to 1.89 where the data centers around 0 with a standard deviation of 1. This way of standardizing data is sensitive to outliers.

The `MinMaxScaler` compresses the data within a fixed range of [0,1]. While this transformation makes all values fall within the specified range, extreme outlier may still distort the scaling of other values, making it less applicable for datasets with large outliers.

Finally, I applied `RobustScaler` which uses median and Inter Quartile Range (IQR) to scale the data from [-2.0,2.0]. This way of scaling is more robust to outliers because it scales data based on the central 50% of data, limiting the influence of outliers in the dataset.

2.3 [Q6] Encoding Categorical Variables

The first step of this exercise is to remove features with a substantial amount of unique value (i.e., those with >70% unique values). This is because features with too many unique values will not provide useful patterns for the linear regression models. Consequently, `c5` and `c9` are dropped from the data set.

When training machine learning models, it is essential to convert categorical variables into numerical format using encoding techniques. `OneHotEncoder` is preferred when the feature has no apparent order, e.g., when the feature is nominal. `OneHotEncoder` is used on feature `c7` because it has no apparent order (categories: S, C, Q).

On the other hand, `OrdinalEncoder` is preferred when categorical variables have a clear ordering or ranking. This encoding technique is used on feature `c8` because it has a clear ordering (categories: Lower Class = 0, Middle Class = 1, Upper Class = 2). Since each category is assigned to an integer, the machine learning model can easily interpret the relative order and recognize patterns based on the order.

Before encoding, the first the first categorical feature is `c1`, which represents “male” and “female”. After applying `OneHotEncoder` on this feature, transform the feature into binary representation, where male = 1 and female = 0. This is crucial when training models because linear regression models generally perform better with binary data compared to string-based categorical variables.

2.4 [Q7] Feature Engineering

Initially the dataset contains the full names of all individuals in the `c9` column. Since all the names are unique identifiers, this information does not serve any purpose when modelling and is removed. However, their titles (e.g., Mr., Mrs., Master., and Miss.) contain variable categorical information. Their titles can give information about their age, social status and relationship status. Hence, it is beneficial to initialize new binary columns in the dataset for each title. Adding these binary columns ensures that the model focuses on information that is descriptive and reduces complexity within the model.

3 Part 3: Regression

Regression models are essential in machine learning to predict outcomes based on input data. Based on data preprocessing and correlation analysis, the regression uncovers relationships between variables and helps forecast future values based on new inputs.

3.1 [Q8] Linear Regression

Before constructing any regression models, it is important to split the preprocessed dataset into training data and validation data. The linear regression models are trained on the training dataset, which is 80% of the whole dataset. Afterwards, the regression model's performance is validated on unseen validation data, which consists of the remaining 20%.

I have built three linear regression models:

- Regression model 1: Uses numerical feature `c2` to determine its correlation with the regression target (`reg_target`).
- Regression model 2: Uses numerical feature `c3` to determine its correlation with the regression target (`reg_target`).
- Regression model 3: Uses numerical feature `c4` to determine its correlation with the regression target (`reg_target`).
- Regression model 4: Uses ordinal feature `c8` to determine its correlation with the regression target (`reg_target`).
- Regresssion model 5: Combines all features.

The first regression model reports an R^2 equivalent to zero, which means that `c2` does not have any predictive power on the regression target. There is no relationship between the regression target, hence it might be beneficial to remove this feature from the model. The Mean Square Error (MSE), however, is relatively high (3661), which suggests that the predictions made on the validation data is far from the original data. For MSE to decrease, a larger dataset would be necessary to pick up on potentially existing patterns in feature `c2`.

The same goes for regression models two and three. Features `c3` and `c5` have explanation power of 0.18% and 3.88% respectively. This suggests that the features have a weak relationship with the regression target. Additionally, the MSE values are both very high, implying that both models' predictions are far from the actual values.

The fourth regression model performs substantially better than the first one, with a R^2 of 0.2986. This suggests that there is a moderately strong relationship between `c8` and the regression target. Whilst this does not indicate perfect fit, it shows that `c8` explains roughly 30% of the variance in the regression target. The MSE is still high with a value of 2566, which suggests that there is room for improvement in the model. However, this result confirms that the predictions of regression model 2 are better than those in model 1.

The third model combines the numerical- and ordinal features which results in a slightly better R^2 of 0.3072. The explanatory power to the model comes mostly from `c8`, but the slight improvement indicates that `c2` has some explanatory power combined with other features. The MSE decreases as the features are combined into one model. Overall, regression model 3 performs the best, which indicates that a combination of features in the regression model has more explanatory power on the regression target.

3.2 [Q9] Regression model with a nominal categorical feature

For this exercise, it is worth noting that the training time will vary slightly each time the model is trained.

As already explained, nominal categorical features do not have a defined order or ranking. To formulate these features as an independent variable of a linear model, `OneHotEncoder` is used to represent the variable on binary format. A new column is created for all the features, except one, to avoid multicollinearity, and they are assigned binary values.

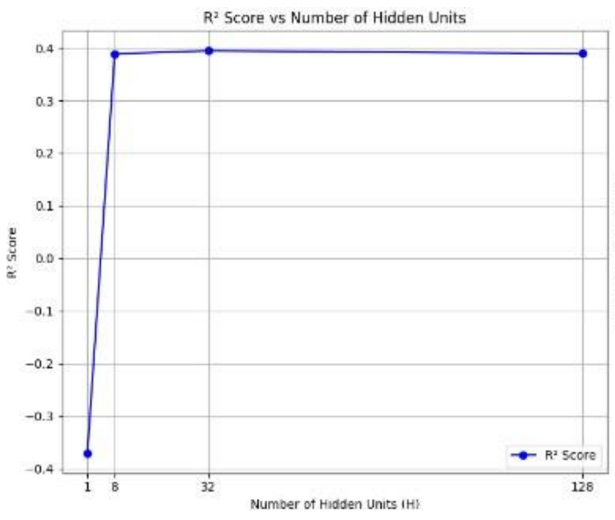
The updated model includes `c2`, `c3`, `c4`, `c8` and the `OneHotEncoded` versions of `c7` (i.e., `c7_Q`, `c7_S`). Compared to the previous linear regression models, R^2 has increased to 0.3827 which suggests that the `OneHotEncoded` versions of `c7` has explanatory effect on the regression target. Additionally, the MSE decreased to 2258.5, though this value remains relatively large. Nonetheless, it is apparent that more independent variables used in the model increase the explanatory effect and decreases mean squared error.

3.3 Feedforward Neural Networks

For this exercise, it is worth noting that the training time will vary slightly each time the model is trained.

In FeedForward neural network models data flows in one direction, from the input layer, through one or more hidden layers, to the output layer. These models can model complex, non-linear relationships between the features in the data and the target variable.

In this exercise I specifically use Multi-Layer Perceptron Regressor models with the features `c2`, `c3`, `c4` and `c8`. There are four hidden units $H \in \{1,8,32,128\}$ where each unit learns a different feature from the input data. This means that H = 1 has very limited capacity for learning patterns, while H = 128 may be better at learning complex relationships, but fall in the risk of overfitting if data is limited. The model's R^2 score, with its corresponding hidden layer value and training time, is reported in line-plots.



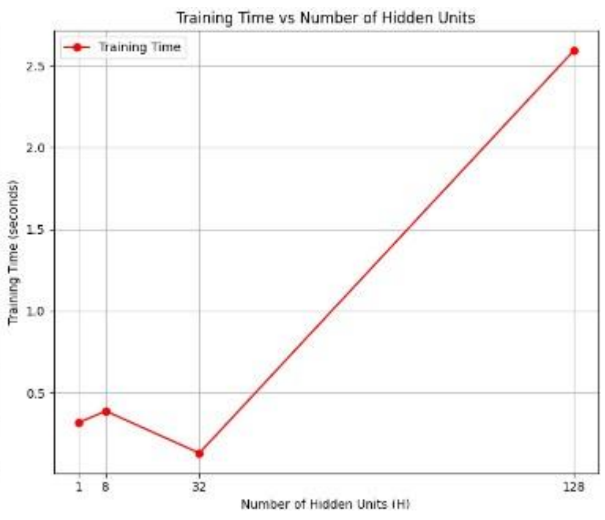


Figure 3.3.1: Line-plots of model's performance. R^2 score/Training time (seconds) on y-axis, and the number of hidden units (H) on the x-axis.

3.4 [Q10] Performance of Neural Network model

With one hidden unit, the model had a training time of 0.4 seconds and a R^2 score on the validation set of -0.3701. Such a low R^2 score indicates that the neural network model performs worse than a simple baseline model and confirms that the model struggles with learning patterns with a low number of hidden units and performs poorly because of it.

With 8 hidden units, the model performs the best (R^2 score on validation set of 0.3888). The training time increased slightly to 0.48 seconds. The model's R^2 score stays stable at 32 hidden features with a score of 0.3946 and a training time of 0.24 seconds.

Lastly. The training model also performs well at 128 hidden units with a score of 0.3892 and a training time of 1.87 seconds. The R^2 score remains relatively stable at 8, 32 and 128 hidden units. It is evident that the models need more than one hidden unit to detect any patterns in the data set. However, as the hidden units become exponentially larger, so do the training time. This suggests that 32 hidden units strike a balance, allowing the model to effectively detect the necessary patterns without excessive computation costs.

3.5 [Q11] Comparison Between Models

The best neural network model had a R^2 score on the validation set of 0.3946 and a training time of 0.22 seconds. The linear regression model, on the other hand, had a R^2 score of

0.3827 and a training time of 0.0033 seconds. As expected, the linear regression model has faster training time because it is relatively simple and computationally efficient. Given similar R^2 scores, it suggests that the neural network does not provide a substantial advantage in capturing patterns within the data. Instead, it might be learning noise rather than meaningful relationships, leading to potential overfitting. In comparison, the linear regression model adapts better to the seemingly linear relationships in the data set.

The features in the linear regression model explain more of the variance in the regression target as well. This implies that the neural network learns the noise in the training data, rather than understanding the patterns in the data, hence overfits. In contrast, the linear regression model may generalize to simpler patterns, which suggests that the relationship between the independent variables and the regression target is close to linear.

3.6 [Q12] Difference Between Metrics in Neural Network Model

As the hidden units increase from 1 to 128 there is a clear trend between R^2 and MSE. Initially, with 1 hidden unit, the model performs worst with a R^2 score of -0.3701 and the highest MSE. As the number of hidden units increases to 8, the R^2 improves to a score of 0.388, and the MSE decreases to 2214. Furthermore, when the hidden units increase to 32, R^2 continues to improve and MSE decreases. However, beyond 32 hidden units, the performance stagnates and MSE stagnates. R^2 and MSE have an inverse relationship: when R^2 increases, MSE decreases.

4 Part 4: Classification

Whilst linear regression models focus on predicting continuous values, classification models' goal is to predict whether the target variable (`class_target`) is binary or not. Logistic regression models estimate probabilities using the sigmoid function and aim to minimize cross-entropy loss.

4.1 [Q13] Logistic Regression Model Performance

The logistic regression model uses Stochastic Gradient Descent (SGD) with `log_loss` loss function and a constant learning rate `eta` tested with values {1.5, 2, 5}. Model with eta = 2 scored the best logistic regression results with an accuracy of 73.18%. The training time is also approximately equal to zero. Moreover, the F1 score is 57.89% which suggests that the model performs moderately and struggles with false positives and false negatives. I chose to use `learning_rate = "constant"` which makes the learning rate fixed through the training.

4.2 [Q14] ROC Curve and AUC Score of Logistic Regression Model

The Receiver Operating Characteristic (ROC) curve is a graphical representation of a binary classifier's performance. It plots the proportion of correct identifications against incorrectly identified positives. The Area Under the Curve (AUC) is the area under the ROC curve and quantifies the overall performance of the logistic regression model.

An AUC score equal to 1 means that the model makes perfect predictions. A score greater than 0.5 means that the model performs better than random guessing. This best logistical regression model has an AUC score of 0.7468, which suggests that the model has moderate classification ability.

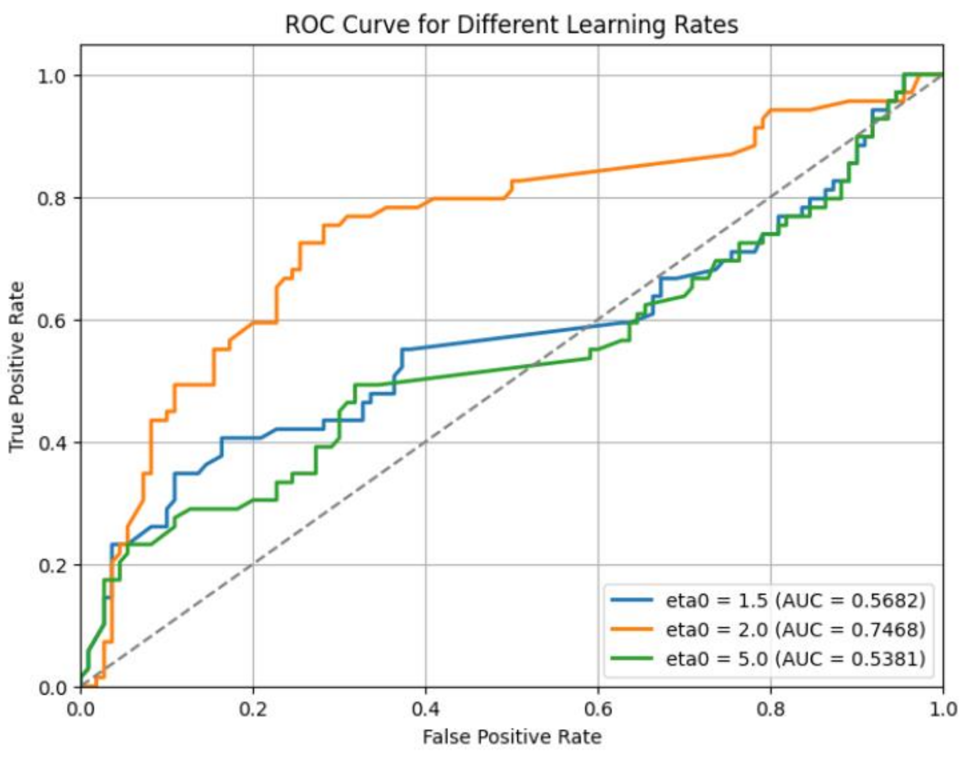


Figure 4.2.1: The ROC Curve for different lerning rates.

4.3 [Q15] Neural Network Classifiers

A neural network classifier consists of layers that process input data and make predictions. In comparison to neural network regression models, neural network classifiers perform better with complex patterns.

The Neural Network Classifiers were trained with different numbers of hidden units (H = 1,8,32,128), and evaluated their training times, F1-scores, and accuracy for each setting. I am using a simple feedforward Multilayer Perceptron (MPL), implemented with the `Keras` library, where layers are stacked sequentially. `Activation = "relu"` assures that the activation function introduces non-linearity. When compiling the models, `optimizer = "adam"` adjusts learning rates dynamically. Additionally, `binary_crossentropy` measures how far the predicted probability is from the actual class, and `accuracy` tracks the proportion of correctly classified samples. Each experiment was repeated three times, and the mean± standard deviation (std) results are reported.

The results from the models are interesting. At one hidden unit, the model performs poorly with an accuracy of 0.6145 ± 0.0000 and a F1 score of zero. The training time is also the longest out of all the settings (Training Time=7.3818 ± 0.7122 seconds), which suggests that the model is too simple and unable to learn any patterns.

At 8 hidden units, the model's performance is the best with an accuracy of 0.7374 ± 0.0046 and an F1 score of 0.5491 ± 0.0186. The training time also decreases to 4.4285 ± 0.1892 seconds.

At 32 and 128 hidden units there are only small improvements, suggesting diminishing returns with more hidden units. Training time does decrease with more hidden units, however, which shows that the models converge more quickly with more hidden units.

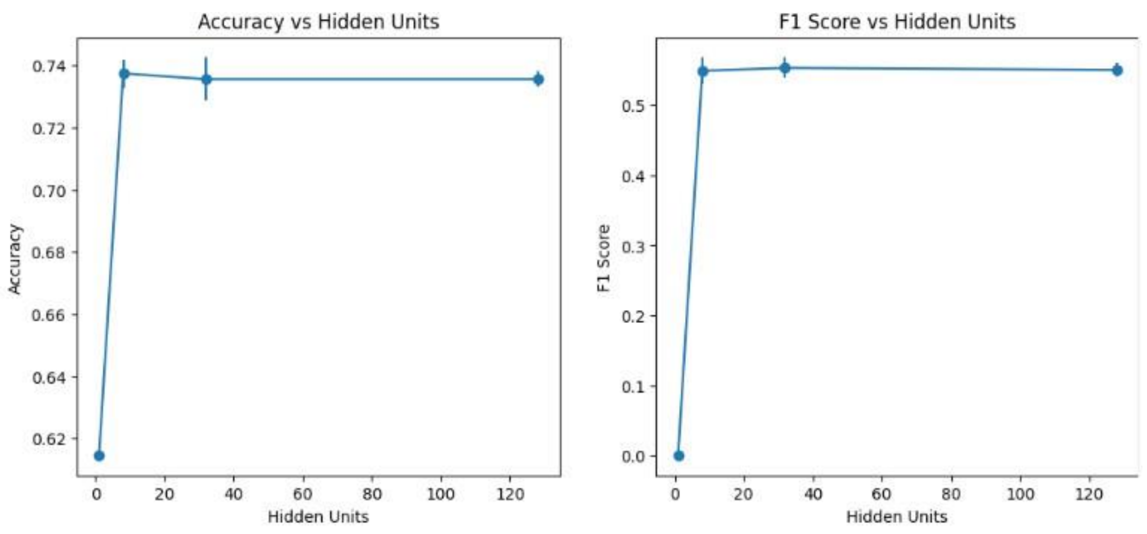


Figure 4.3.1: This figure shows the effect of hidden units of model performance.

4.4 [Q16] Comparison Between Logistic Regression Model and Neural Network Model

The neural network model had accuracy on the validation set of 0.7374 ± 0.0046 and a F1 score of F1=0.5491 ± 0.0186, making it the model with highest accuracy, but also higher standard deviation. In comparison, the best logistic regression model had accuracy of 0.7263 and a F1 score of 0.6316. The neural network has slightly higher accuracy but a lower F1 score, suggesting it is better at overall classification, though it shows more variability in performance across different datasets. Given that the models have similar accuracy, and the logistic regression model outperforms the neural network in terms of F1 score, the logistic regression model is the preferred choice.

4.5 [Q17] ROC Curve of Neural Network for Binary Classification

The ROC curve is necessary to examine neural network classification models because it provides a graphical representation of the model's ability to distinguish between classes at different classification thresholds. The illustration shows True Positive Rate, which is the correct identification, on the y-axis. The False Positive Rate (FPR) is displayed on the x-axis and represents the proportion of actual negatives that are incorrectly classified as positives.

For a model with one hidden unit (H=1), the AUC (Area Under the Curve) score is equal to 0.5, which indicates that the model has moderate prediction power. Despite having few parameters, the model still recognizes patterns in the data to some extent, even though the performance is not optimal.

For a model with 128 hidden units (H=128), the AUC score is equivalent to 0.7472 which is a slight decrease in performance. The model has more parameters to learn from but does not necessarily benefit from it. This suggests that the larger model might introduce bias and overfitting.

Both the models show moderate classification ability. The marginal drop in AUC score at H = 128 shows that it is important to explore and choose optimal model parameters to avoid overfitting.

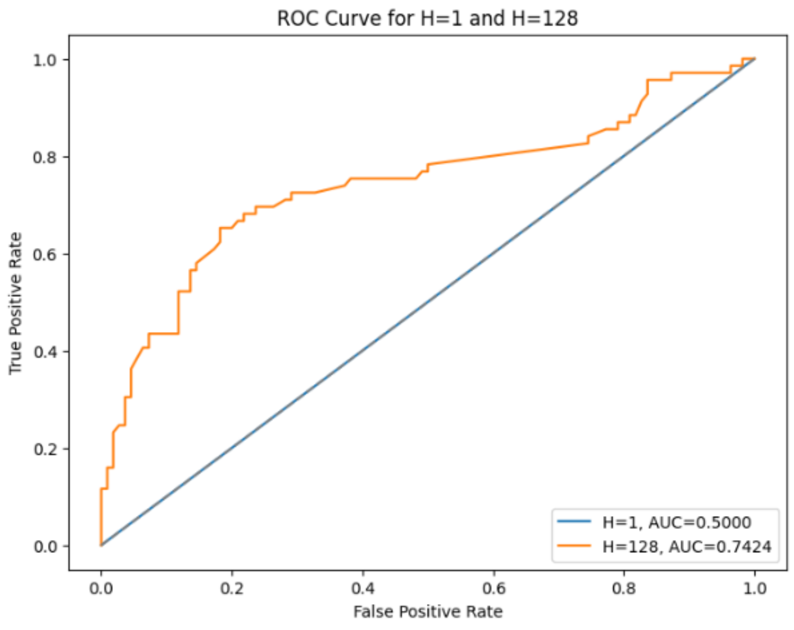


Figure 4.5.1: The ROC curve for H = 1 and H = 128 illustrated for neural network classification model

5 **Bonus**

5.1 [Q18] Preprocessing Validation: Combination A

In this task, a three hidden layer neural network is trained by applying `SimpleImputer` (with mean strategy) for numerical features, `OneHotEncoding` (strategy drop) for categorical features, and `StandardScaler` for normalization. By incorporating `Pipeline` and `ColumnTransformer`, the preprocessing steps were structured effectively, ultimately improving model performance. The neural network model achieved an overall average of 78%, which suggests that it performs moderately at predicting outcomes. The model’s accuracy in predicting that negatives were negatives (class 0 precision) is at 86%. Furthermore, 70% of predicted positives were positive. The validation of this combination was performed using a ROC curve, showing an AUC score of 0.8560, and a Neural Network Classification Report. Underneath is the Neural Network Classification Report and the ROC curve.

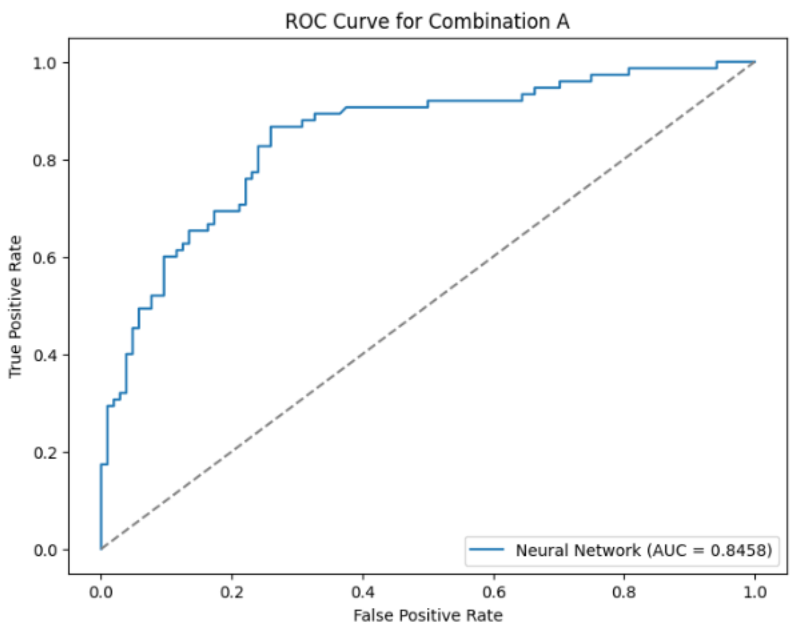


Figure 5.1.1: An illustration of the ROC Curve for combination A.

5.2 [Q19] Preprocessing Validation: Combination B

Combination B uses different imputation strategies from Combination A. This model uses zero imputation for numerical values instead of mean imputation, which avoids potential introduction of bias. Zero imputation can be more beneficial if the data set contains many outliers because it avoids distorting the relationships in the numerical features.

When it comes to the categorical features, combination A used `OneHotEncoder` for all categorical features. Combination B uses `OrdinalEncoder` for ordinal values, and `OneHotEncoder` for other categorical features, which reduces the number of binary features and the overall complexity of the data.

Overall neural network models tend to perform better with more meaningful features and minimalized complexity. The classification report shows an increase in overall accuracy, from 78% to 79%. The F1 scores for both group 0 and group 1 have also increased substantially, indicating that reduced complexity increases prediction accuracy.

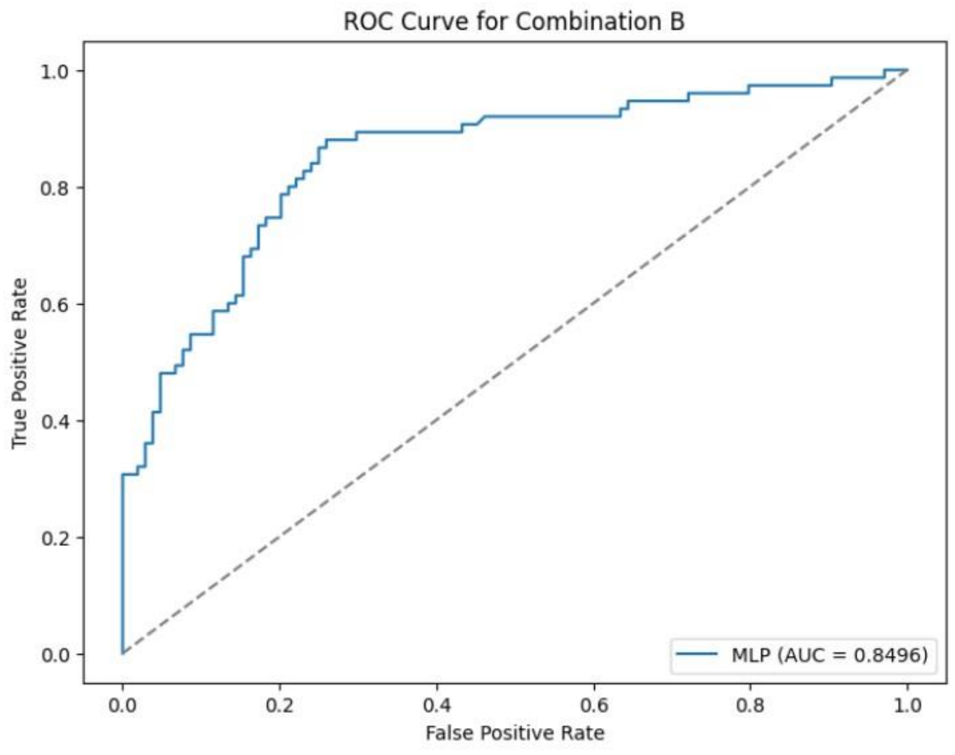


Figure 5.2.1: An illustration of the ROC Curve for combination B.

5.3 [Q20] Preprocessing Validation: Custom Combination

In the final preprocessing combination, I use custom techniques for numerical and categorical features, aiming to outperform the other combinations in terms of accuracy. For numerical features, the model uses `SimpleImputer` with strategy = "median" and normalize using `RobustScaler`. Generally, median imputation is more robust against outliers, which will lead to increased accuracy. Furthermore, the model will incorporate `OrdinalEncoder` for ordinal features, and `OneHotEncoder` for the remaining categorical features, just like Combination B. This combination will also apply `VarianceThreshold` to remove features with low variance, in hopes of reducing noise in the dataset.

The prediction accuracy of this combination was similar the other combinations, in fact a little lower than combination B. `RobustScaler` is generally great for handling outliers, but it may not have been as effective on this dataset. The same goes for imputation `strategy = "median"`, which handles outliers well.

Ultimately, the custom combination incorporates more complexity processing the data, which makes the data harder for the neural network to learn patterns from. Hence, prediction accuracy and F1 score are reduced.

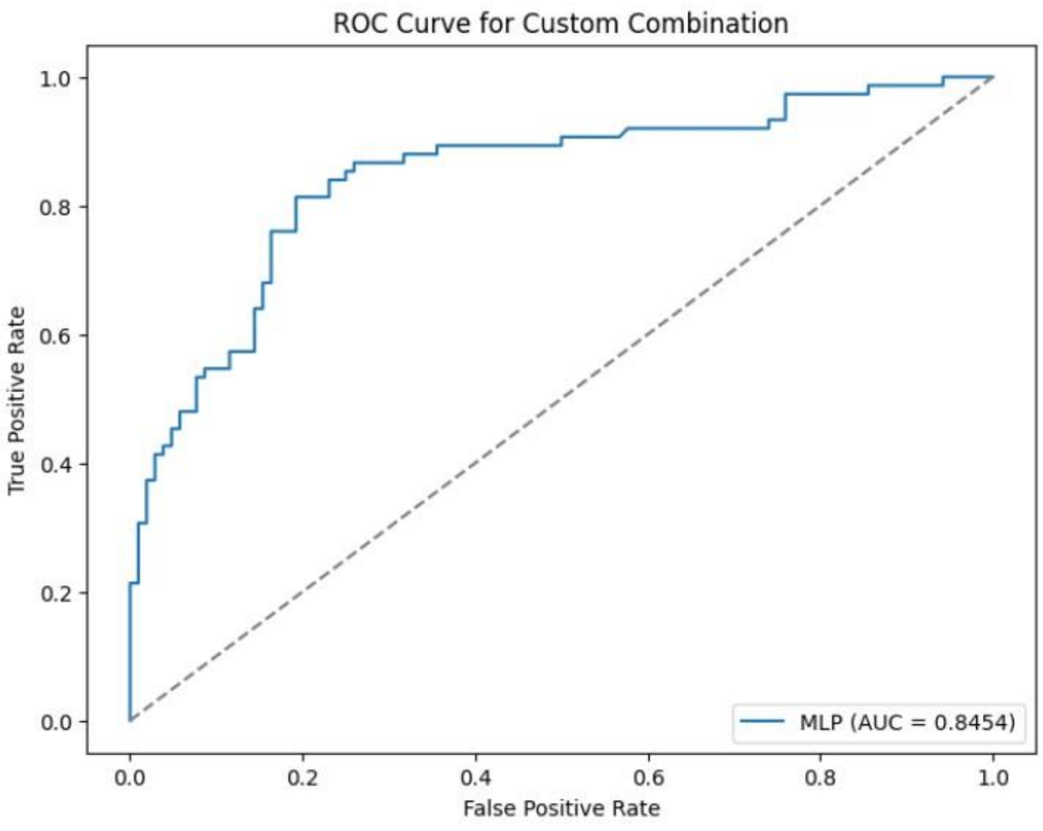


Figure 5.3.1: An illustration of the ROC Curve for Custom Combination.

5.4 [Q21] Hyperparameter Tuning

Hyperparameter tuning is the process of selecting the best possible parameters to optimize a model's performance. This exercise specifically asks to report five combinations of hyperparameter settings and highlight the best setting in terms of validation accuracy.

This model defines two pipelines: one for numerical features and one for categorical features. This way, multiple preprocessing steps are applied consistently to training and validation data. Furthermore, we define a `ColumnTransformer` which combines numerical and categorical pipelines. Ultimately, we define a variable `param_grid` which holds multiple configurations of classification parameters. These configurations are continuously tested using `GridSearchCV` to identify the optimal combination for model performance.

In total the grid tested 96 different hyperparameter calculations using five cross validation sets, totaling in 480 fits. The best models achieved a mean validation accuracy of 80.61% (params 10 and 8). The same models also performed best in terms of standard deviation with a score of 3.55%, proposing stability.

Ultimately, the best hyperparameter settings led to an accuracy of 79.89% on the testing set. This is not a substantial improvement compared to the previous linear regression and classification models trained in this project, which suggests that further hyperparameter tuning might have a minimal effect on accuracy. The model is likely close to its optimal performance given the data received.