COMP 4211: Machine Learning

Programming Assignment 1

Authors

HUANG, I-wei - 20824074

March 6, 2024

Contents

1 Part 1: Data Exploration and Preparation

Data exploration and preparation are crucial steps in the data analysis process. Understanding the data and identifying potential quality issues ensures a more complete data preprocessing step, leading to better performance overall. Part 1 explores the features of the **train.csv** dataset.

1.1 [Q1] Dataset Overview

Size of the Dataset:

The dataset contains 3539 instances, 31 features, one regression target, and one classification target.

Feature Types:

Numerical features: 'C6', 'C14', 'C16', 'C17', 'C18', 'C19', 'C20', 'C21', 'C22', 'C23', 'C24', 'C25', 'C26', 'C27', 'C28', 'C29', 'C30'.

Categorical features: 'C0', 'C1', 'C2', 'C3', 'C4', 'C5', 'C7', 'C8', 'C9', 'C10', 'C11', 'C12', 'C13', 'C15'.

1.2 [Q2] Missing Values

Table 1 shows the features with missing values and the proportion of the missing values.

Feature	Proportion of missing values (%)
C0	0.819
C4	1.385
C5	3.052
C8	2.317
C9	4.012
C11	4.465
C12	4.804
C13	3.391
C15	3.899
C17	4.182
C20	4.832
C22	0.452
C23	0.791
C25	4.069
C29	4.521

Table 1: Missing Values

Missing data can disrupt the computations of machine learning models such as linear regression or neural networks. Furthermore, they may not only result in a loss of information and thus reduced statistical power, but if not handled properly, they may also introduce bias into the analysis.

1.3 [Q3] Feature Distribution

Numerical features:

The numerical features can be classified as discrete or continuous based on their values and distributions.

Discrete: 'C14', 'C16', 'C17', 'C18', 'C19', 'C21', 'C22', 'C23', 'C24', 'C25', 'C27'.

Continuous: 'C6', 'C20', 'C26', 'C28', 'C29', 'C30'.

Table 2 and Figure 1 illustrate the distribution of the first three numerical features.

	C6	C14	C16
mean	66.325	23.204	0.684
median	66.550	20.000	0.000
range	47.500	53.000	20.000
[min, max]	[47.5, 95]	[17, 70]	[0, 20]
variance	43.811	55.155	5.174

Table 2: Distribution of the first three numerical features

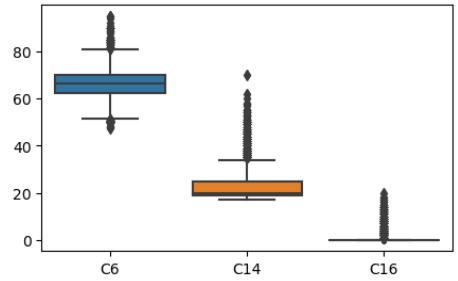

Figure 1: Boxplot of the first three numerical features

Categorical features:

By examining the categorical features and values, they can be classified as binary, nominal, or ordinal.

Binary: 'C4', 'C8', 'C9', 'C10', 'C11', 'C12', 'C13', 'C15'.

Nominal: 'C0', 'C1', 'C3', 'C7'.

Ordinal: 'C2', 'C5'.

Figure 2 displays the category counts and distributions of the first three categorical features.

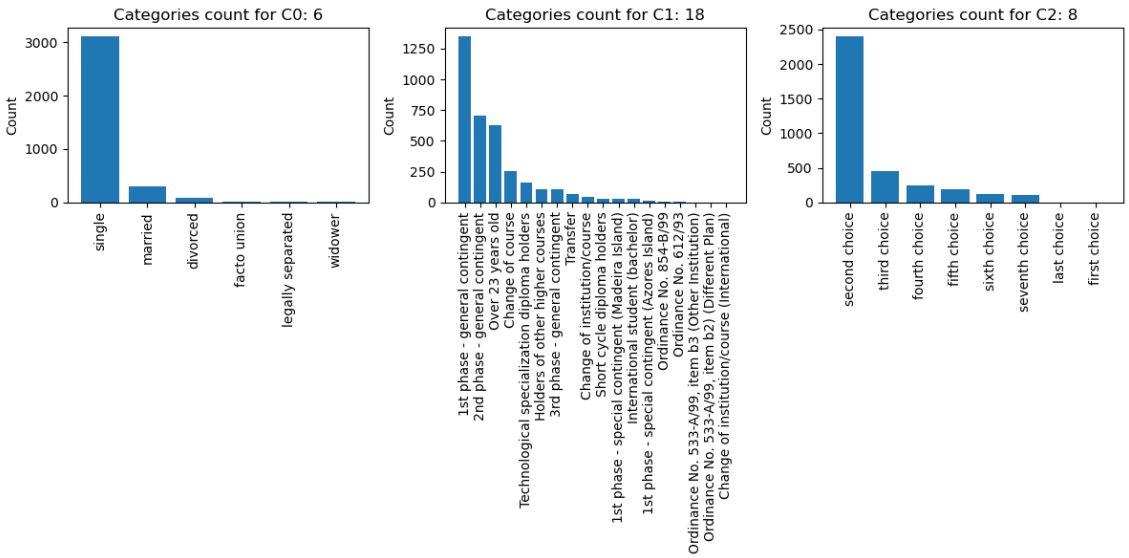

Figure 2: Barplot of the first three categorical features

1.4 [Q4] Outliers

In Section 1.4, two methods are used to identify potential outliers: Inter-Quartile Range (IQR) and Z-Score.

IQR:

The IQR is calculated as the difference between the third quartile (Q3) and the first quartile (Q1), where Q3 and Q1 are the 75th and 25th percentile of the dataset, respectively. The data points that fall below $Q1 - 1.5 \times IQR$ or above $Q3 + 1.5 \times IQR$ are identified as outliers.

Z-Score:

The data points that are less than $\mu - 3 \times \sigma$ or greater than $\mu + 3 \times \sigma$ are identified as outliers, where μ and σ are the average value and standard deviation of a particular feature.

Potential outliers are identified by applying the two outlier detection methods to the dataset. Table 3 shows the number of outliers detected by various methods.

	C6	**C14**	**C16**
IQR	147	340	454
Z-Score	18	79	114

Table 3: Number of outliers of the first three numerical features

Outliers can have a significant impact on preprocessing and modeling strategies. They are often considered noise or errors in the data, and their extreme values can distort statistical measures such as mean and standard deviation, affecting normalization, scaling, or imputation techniques.

Additionally, models that are susceptible to outliers, like linear regression, can be affected by them in terms of performance and stability. They can distort the relationship between variables, leading to biased parameter estimates and poor predictive performance.

1.5 [Q5] Correlation Analysis

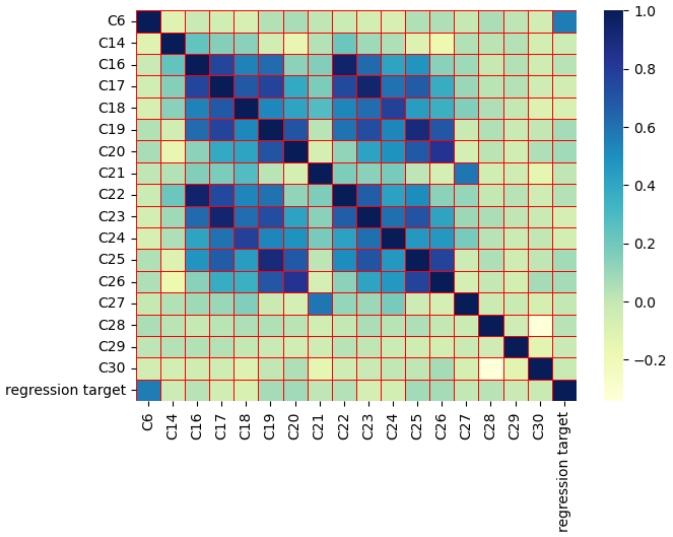

Figure 3: Heatmap of the numerical features pairwise correlation

The pairwise correlation of the numerical features, including the regression targets, is shown using a heatmap in Figure 3.

Strong correlations exist between the features 'C16', 'C17', 'C18', 'C19', 'C20', 'C22', 'C23', 'C24', 'C25', and 'C26', as Figure 3 illustrates. This may indicate a redundancy in the information they provide; in this case, including all these highly correlated features in a predictive model may not offer additional value and can potentially introduce multicollinearity issues.

Another strong correlation to highlight is the high correlation between 'C6' and the regression target, which makes the feature potentially valuable for predicting the target variable.

1.6 [Q6] Initial Thoughts on Preprocessing

Based on the exploration of the dataset in this section, several preprocessing steps are necessary for the regression models or neural networks to run. These steps include handling missing data and encoding categorical variables.

Handling missing data is essential for regression models or neural networks to run successfully because missing data can cause errors or failures in the calculation. Regression models and neural networks rely on mathematical operations and calculations to estimate parameters, make predictions, or update weights. When missing data is encountered during these computations, it can lead to mathematical errors or undefined results. Similarly, categorical data needs to be encoded into numerical form to ensure compatibility with these mathematical operations. Therefore, by imputing appropriate missing data and converting categories into numeric representations, the models can process and interpret the data correctly.

Other challenges identified during exploration that do not affect running models but might affect model performance are data scaling and normalization, handling outliers, and feature selection.

Features in the dataset may have different scales or units of measurement. This disparity can cause certain features to dominate the learning process, as models tend to give more weight to features with larger scales. By standardizing or normalizing the data, the features are brought to a similar scale. This prevents any particular feature from dominating the learning process and ensures that the model takes all features into account equally.

However, as mentioned in Section 1.4, outliers can distort statistical measures such as the mean and standard deviation. As a result, the standardization process may be skewed, with the majority of data points compressed into a narrow range. This can result in poor feature representation and have a negative impact on model performance. Likewise, the extreme values of the outliers can heavily influence the range. Consequently, the normalization may result in a compressed range for the majority of the data, leading to loss of information and potentially affecting model performance.

Feature selection can have a significant impact on model performance as well, choosing a subset of relevant features from a larger set can lead to several beneficial outcomes. First, it enhances model simplicity, as unnecessary or irrelevant features are excluded. Second, it helps to reduce overfitting by reducing the number of features from which the model must learn. Lastly, it decreases redundancy by eliminating features that provide similar information. These positive effects collectively contribute to improved model performance.

2 Part 2: Data Preprocessing Techniques

Preprocessing is an indispensable step in the machine learning pipeline, serving as the foundation upon which the performance of the models is built. Part 2 applies various advanced preprocessing techniques to the dataset.

2.1 [Q7] Handling Missing Values

As shown in Table 1, there are missing values in multiple features. These features can be classified into different data types, such as discrete numerical data, continuous numerical data, or categorical data. Some common imputation strategies, such as mean, median, mode, and constant imputation, will have a different impact on feature distribution and model performance.

Categorical data commonly use mode imputation because they are not numerical and thus cannot have math operations like average or median. Filling the missing data with the most frequent value is a simple way to handle categorical missing data. Mean and median imputations are commonly used for numerical data to preserve the overall distribution. Discrete data is preferred to be imputed with the median because the average will often result in decimal points and have to be further processed. For continuous data, the preferred strategy depends on the distribution of the feature. If the feature is approximately Gaussian distributed, then mean imputation is preferred. However, if the data is skewed to one side, the average value may not represent the central tendency accurately. In this case, the median may be more appropriate to preserve the overall distribution. Constant imputation is used in scenarios when the missingness itself is meaningful. For example, a missing value in a survey can be imputed with 'Unknown' to indicate an absent response. However, in the **train.csv** dataset, the meaning of each column is unknown, therefore, it may not be suitable to use constant imputation and assume the meaning of the missing values.

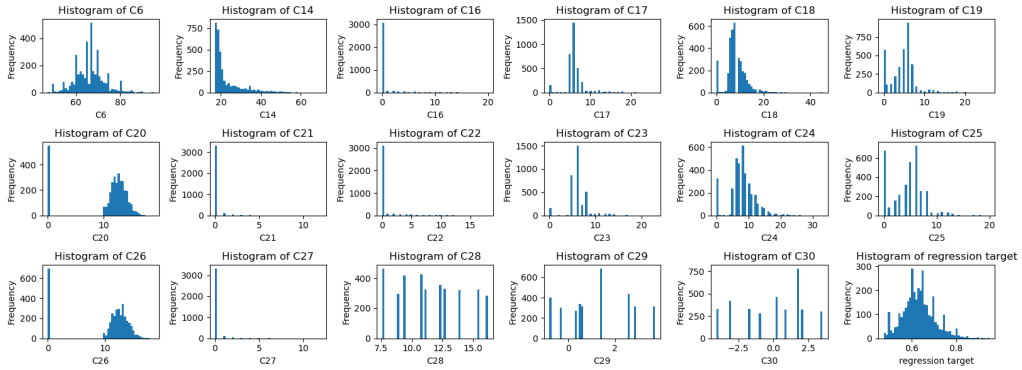

Figure 4: Distribution of numerical data

Figure 4 illustrates the distribution of numerical data. For example, since 'C6' is continuous and normal distributed, it is a good candidate for mean imputation. In contrast, 'C20' is skewed to one side. Its mean value is 10.625, while its median is 12.328. Therefore, imputing the median value will be more suitable to preserve the distribution. After analyzing data distributions, the imputation method for the features is as follows:

Mean: 'C6'.

Median: 'C14', 'C16', 'C17', 'C18', 'C19', 'C20', 'C21', 'C22', 'C23', 'C24', 'C25', 'C26', 'C27', 'C28', 'C29', 'C30'.

Mode: 'C0', 'C1', 'C2', 'C3', 'C4', 'C5', 'C7', 'C8', 'C9', 'C10', 'C11', 'C12', 'C13', 'C15'.

Constant: *None*.

2.2 [Q8] Normalization and Standardization

As mentioned in Section 1.6, scaling the data to a similar scale prevents any particular feature from dominating the learning process. Common scaling techniques are standardization, normalization, and robust standardization. Standardization is useful when the data distribution is approximately normal, it transforms data by subtracting the mean and dividing by the standard deviation, resulting in a zero mean and a standard deviation of 1. Normalization rescales the data to a specific range, typically between 0 and 1. It is useful when the absolute values of the data are not as important as their relative positions within the range. Robust standardization is similar to standardization but uses the median and IQR instead. It is less sensitive to outliers compared to standardization and can be useful when the data contains extreme values that could influence the mean and standard deviation. Table 4 shows the first numerical feature column ('C6') of the first ten samples before and after standardization.

C6		
Index	**Before**	**After**
0	65.0	−0.200
1	65.0	−0.200
2	59.5	−1.031
3	66.55	0.034
4	71.0	0.706
5	70.0	0.555
6	57.5	−1.333
7	65.5	−0.125
8	70.0	0.555
9	80.0	2.066

Table 4: First ten samples of C6 before and after scaling

2.3 [Q9] Encoding Categorical Variables

Common encoding techniques consist of one-hot encoding and ordinal encoding. One-hot encoding is suitable when the categorical variable is nominal and has no inherent order or hierarchy. On the contrary, ordinal encoding is appropriate when the categorical variable has an ordered relationship, and the categories can be assigned numerical values accordingly. Table 5 shows the first categorical feature column ('C0') of the first ten samples before and after one-hot encoding.

	Before	**After**				
Index	**C0**	**C0_facto _union**	**C0_legally _separated**	**C0 _married**	**C0 _single**	**C0 _widower**
0	divorced	0.0	0.0	0.0	0.0	0.0
1	single	0.0	0.0	0.0	1.0	0.0
2	single	0.0	0.0	0.0	1.0	0.0
3	married	0.0	0.0	1.0	0.0	0.0
4	single	0.0	0.0	0.0	1.0	0.0
5	single	0.0	0.0	0.0	1.0	0.0
6	single	0.0	0.0	0.0	1.0	0.0
7	single	0.0	0.0	0.0	1.0	0.0
8	single	0.0	0.0	0.0	1.0	0.0
9	single	0.0	0.0	0.0	1.0	0.0

Table 5: First ten samples of C0 before and after encoding

2.4 [Q10] Feature Selection

Feature selection enhances model simplicity by removing unnecessary or irrelevant features. Ensemble feature selection is used in this task, which combines the results of six feature selection modules from `scikit-learn`: `VarianceThreshold`, `SelectKBest`, `SelectFromModel`, `SelectFpr`, `SequentialFeatureSelector`, and `RFE` (Recursive Feature Elimination). Features selected by half or more modules are the final selection.

`VarianceThreshold` removes features with a variance below a certain threshold. `SelectKBest` evaluates each feature independently and ranks them according to their relevance to the target variable, the top K features are selected, and the rest are removed. `SelectFromModel` works by training a model and utilizing the feature importance scores provided by the model. Features with importance scores above a specified threshold or a certain percentile are selected, while others are discarded. `SelectFpr` performs feature selection based on the false positive rate, features are selected if their p-values are below a given threshold. `SequentialFeatureSelector` uses a greedy search algorithm to select features incrementally or eliminate them iteratively. `RFE` works by training a model and recursively eliminating the least relevant feature based on the model's feature weights or importance.

Two sets of features are selected, one for a regression task and the other for a classification task. The feature selection methods, namely `SelectFromModel`, `SequentialFeatureSelector`, and `RFE`, are applied differently due to the different models used in each task. For the regression task, the features are selected using `LinearRegression()`, while for the classification task, `SGDClassifier(loss='log_loss')` is used to select the features.

Regression: 'C0_widower', 'C1_Ordinance No. 854-B/99', 'C1_Over 23 years old', 'C3_Animation and Multimedia Design', 'C5', 'C6', 'C7_Cuban', 'C7_Mexican', 'C7_Turkish', 'C19', 'C25', 'C26', 'C20'.

Classification: 'C3_Animation and Multimedia Design', 'C5', 'C6', 'C10', 'C11', 'C14', 'C19', 'C23', 'C25'.

2.5 [Q11] Feature Engineering

After examining Figure 3, one approach that could possibly enhance the performance of a linear regression model is to apply feature interactions on the variables 'C16', 'C17', 'C18', 'C19', 'C20', 'C22', 'C23', 'C24', 'C25', and 'C26'. As mentioned in Section 1.5, these features show strong correlations. However, including all of these highly correlated features in a predictive model can potentially introduce issues related to multicollinearity. To mitigate this problem, a possible solution is to aggregate the values of these features by summing them and creating a new column to replace them. This approach can potentially enhance the performance of the linear regression model by reducing multicollinearity and capturing the combined effects, leading to improved predictive accuracy and generalization to new data.

3 Part 3: Regression

3.1 Linear Regression

In Section 3.1, seven linear regression models are built using the `LinearRegression()` module from `scikit-learn`, where each of the first three models uses only one feature. The seventh model explores the relationship between linear combinations of the six selected features and the regression target. The six features are a subset of the regression feature selection set from Section 2.4: 'C1_Ordinance No. 854-B/99', 'C5', 'C6', 'C19', 'C20', and 'C26'. A training and testing set is split using the `train_test_split` submodule in `scikit-learn`, with 80% for training and 20% for validation. `random state` is set to 4211 for reproducibility.

3.2 [Q12], [Q13]

The R2 score is typically expressed as a value between 0 and 1, with 1 indicating a perfect fit and 0 indicating no relationship between the features and the target. A negative R2 score occurs when the regression model performs worse than a model that predicts the target mean. The mean squared error (MSE) measures the average squared difference between the predicted and actual values of the target. A lower MSE value suggests a better fit of the regression model to the data.

Predictions on the validation set are made after training the seven models using the training set. The validation R2 score and MSE of each model are shown in Table 6.

Model features	**R2 score**	**MSE**
C1_Ordinance No. 854-B/99	0.012	0.005
C5	0.047	0.005
C6	0.346	0.003
C19	−0.008	0.005
C20	−0.009	0.005
C26	−0.008	0.005
Combined	0.370	0.003

Table 6: R2 score and MSE of the seven linear regression models

The combined model using all six features performs the best, followed by models fitted with 'C6', 'C5', and 'C1_Ordinance No. 854-B/99'. Models using 'C19', 'C20', and 'C26' have negative R2 scores, indicating that they cannot capture the underlying patterns and variability in the data.

3.3 [Q14]

In mathematical terms, weights refer to the coefficient assigned to each independent variable. These weights determine the strength and direction of the relationship between the independent variables and the dependent variable. For a binary categorical independent variable, the weight indicates how much the dependent variable is expected to change for a category change in the binary variable. A categorical feature with more than two possible values can be encoded using one-hot encoding. One-hot encoding transforms each category into a separate binary variable, indicating whether the observation belongs to that particular category. By including these binary variables in the linear model, the model can account for the effects of categorical variables with more than two possible values.

'C1' is an example of how a categorical feature with more than two possible values can be formulated as the independent variable of a linear model. It is first one-hot encoded, and then its encoded feature 'C1_Ordinance No. 854-B/99' is selected in the model in Section 3.1.

3.4 Feedforward Neural Networks

In this section, feedforward neural networks with three hidden layers and different numbers of hidden units $H \in \{1, 8, 32, 128\}$ are built using `MLPRegressor()` from `scikit-learn` and trained with the six features selected in Section 3.1. The hyperparameter `early stopping` is set to 'True' to avoid overfitting, the others remain their default values.

3.5 [Q15]

Each model is repeated three times, and the mean and standard deviation of the training time and R2 score are calculated. Table 5 and Figure 6 show the model settings and performance of the different neural networks.

H	Early stopping	Mean train time	Std train time	Mean R2 score	Std R2 score
1	True	0.916	0.445	0.034	0.050
8	True	2.436	0.061	0.343	0.013
32	True	1.266	0.571	0.380	0.023
128	True	1.302	0.262	0.393	0.009

Figure 5: Model setting and performance of regression neural networks

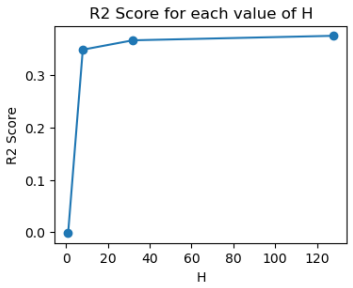

Figure 6: R2 score of regression neural networks

My hypothesis before the training was that the training time and R2 score would increase as the number of hidden units increased. The more hidden units, the more parameter the model has to train, and the more non-linearity it can describe. According to Table 5, the R2 score stays true to the hypothesis, however, the training time did not.

3.6 [Q16]

A comparison of the training time and R2 score of the linear regression model in Section 3.1 and the best regression neural network model is done. Table 7 shows the training time and the R2 score of the linear regression model and the feedforward neural network.

Model	Train time	R2 score
Linear Regression	0.005	0.370
Feedforward Neural Network	1.302	0.393

Table 7: Comparison of linear regression and regression neural network

From Table 7, it can be derived that although the feedforward neural network has a higher R2 score, its training time is 280 times the linear regression model. This proposes a dilemma of choice between faster training time and higher performance.

4 Part 4: Classification

4.1 Logistic Regression

In Section 4.1, gradient-descent-based logistic regression models with different learning rates $\eta \in \{0.0001, 0.001, 0.01, 0.1\}$ using `SGDClassifier()` from `scikit-learn` are trained with the classification feature selection set from Section 2.4. The hyperparameter `eta0` is chosen from one of the values in the learning rates. `loss` is set to 'log_loss' for the cross-entropy loss learning process. `learning_rate` is set to 'optimal' to decrease η gradually during the learning process to enhance convergence. Since the solution found may depend on the initial weight values, each model is repeated three times. A training and testing set is split using the `train_test_split` submodule in `scikit-learn`, with 80% for training and 20% for validation. `random state` is set to 4211 for reproducibility.

4.2 [Q17], [Q19]

After training the models using the training set, predictions on the validation set are made. The corresponding mean and standard deviation of the training time, accuracy, and F1 score are shown in Table 8.

η	Mean train time	Std train time	Mean Accuracy	Std Accuracy	Mean F1 score	Std F1 score
0.0001	0.021	0.006	0.857	0.022	0.845	0.029
0.001	0.020	0.003	0.857	0.012	0.853	0.008
0.01	0.024	0.002	0.856	0.026	0.853	0.022
0.1	0.017	0.004	0.856	0.004	0.853	0.002

Table 8: Training time, accuracy, and F1 score of the logistic regression models

The models with different learning rates have very similar performance in training time, accuracy, and F1 score. This indicates that they have converged to good local minima.

4.3 [Q18]

The Receiver Operating Characteristic (ROC) curve is important to examine because it evaluates a classification model's performance across different thresholds. By plotting the true positive rate against the false positive rate, the ROC curve visualizes the model's ability to discriminate between the positive and negative classes. The area under the curve (AUC) provides a single scalar value that summarizes the model's performance. A higher AUC indicates better overall discrimination power. Figure 7 plots the ROC curve of the logistic regression model with $\eta = 0.1$. The AUC value is 0.907.

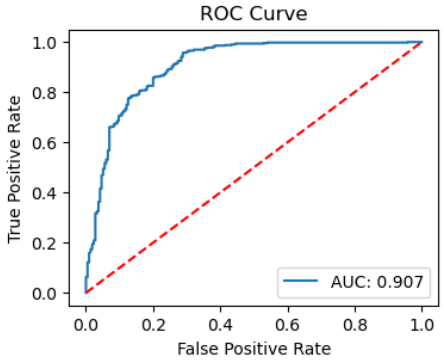

Figure 7: ROC curve of logistic regression model

4.4 Feedforward Neural Networks

In this section, feedforward neural networks with three hidden layers and different numbers of hidden units $H \in \{1, 8, 32, 128\}$ are built using `MLPClassifier()` from `scikit-learn` and trained with the classification feature selection set from Section 2.4. The hyperparameter `early stopping` is set to 'True' to avoid overfitting, `solver` is set to 'sgd' for gradient-descent-based optimization algorithm. The others remain their default values.

4.5 [Q20], [Q23]

Each model is repeated three times, and the mean and standard deviation of the training time, accuracy, and the F1 score are calculated. Table 9 illustrates the model settings and performance of the different neural networks.

H	Mean train time	Std train time	Mean Accuracy	Std Accuracy	Mean F1 score	Std F1 score
1	0.098	0.006	0.435	0.185	0.285	0.203
8	0.466	0.162	0.791	0.067	0.745	0.121
32	0.825	0.372	0.851	0.014	0.844	0.015
128	1.500	0.433	0.863	0.013	0.855	0.015

Table 9: Model setting and performance of classification neural networks

Unlike the results of the regression feedforward neural networks in Table 5, the training time, accuracy, and F1 score all increase as the number of hidden units increases. This corresponds to my original hypothesis, which is that as the number of hidden units increases, there are more parameters the model has to train, resulting in a longer training time. With the increased hidden layer size, the feedforward neural network model is able to describe more non-linearity, potentially obtaining a better performance.

4.6 [Q21]

Figure 8 displays the accuracy and F1 score for each value of H. A small gap between the accuracy and F1 score can be seen, this may be due to class imbalance in the dataset. In the presence of class imbalance, accuracy can be misleading and biased towards the majority class. F1 score, on the other hand, combines precision and recall, providing a balanced evaluation of the model's performance. In the case of the **train.csv** dataset, there are 2434 'success' and 1105 'failure'. The F1 score may be lower due to the lower recall for the minority class.

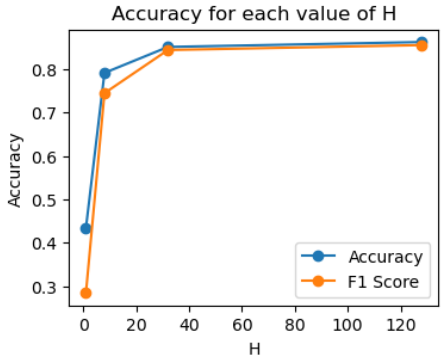

Figure 8: Accuracy and F1 score of classification neural networks

4.7 [Q22]

A comparison of the training time, accuracy, and F1 score of the logistic regression model in Section 4.1 and the best classification neural network model is made. Table 10 shows the training time, accuracy, and F1 score of the logistic regression model and the feedforward neural network.

Model	**Train time**	**Accuracy**	**F1 score**
Logistic Regression	0.022	0.874	0.869
Feedforward Neural Network	2.009	0.863	0.856

Table 10: Comparison of logistic regression and classification neural network

From Table 10, it can be derived that not only does the feedforward neural network have lower accuracy and F1 score, but its training time is 90 times the logistic regression model. This suggests that logistic regression is simple and efficient enough for the classification task on the **train.csv** dataset.

5 Part 5: Performance Enhancement

5.1 Preprocessing Validation

In this section, we explore the impact of different preprocessing techniques on model performance by constructing and evaluating multiple pipelines. `Pipeline` and `ColumnTransformer` modules from `scikit-learn` are utilized to enable efficient experimentation with various preprocessing strategies. Evaluation of how different preprocessing combinations affect the performance of a three-hidden-layer classification neural network is done. The hyperparameters of the model are `hidden_layer_sizes=(32,32,32)`, `early_stopping=True`, and `solver='sgd'`.

5.2 [Q24] Combination A

For numerical features, `StandardScaler` and `SimpleImputer` with 'mean' strategy for imputation is applied. For categorical features, `OneHotEncoder` is used for encoding, and `SimpleImputer` with 'mode' strategy for imputation is applied. Feature selection or feature engineering was not performed.

5.3 [Q25] Combination B

For numerical features, `MinMaxScaler` and `SimpleImputer` with 'zero' strategy for imputation is applied. Categorical features are imputed with `SimpleImputer` 'mode' strategy. Ordinal features are encoded with `OrdinalEncoder`, and the remaining are encoded with `OneHotEncoder`. Feature selection or feature engineering was not performed.

5.4 [Q26] Combination C

Combination C is a custom combination where I choose a different preprocessing technique for numerical and categorical features based on my hypothesis of what might work best.

Numerical features are imputed with `KNNImputer` with `k=5`. Discrete numerical features are scaled using `RobustScaler`, and continuous numerical features are scaled using `StandardScaler`. Categorical features are imputed with `SimpleImputer` 'mode' strategy. Ordinal features are encoded with `OrdinalEncoder`, and the remaining are encoded with `OneHotEncoder`. Feature selection or feature engineering was not performed.

Combination	Mean Accuracy	Std Accuracy	Mean F1 score	Std F1 score
A	0.757	0.060	0.716	0.097
B	0.696	0.001	0.573	0.002
C	0.846	0.010	0.840	0.012

Table 11: Performance of different preprocessing technique combinations

Table 11 shows the results of the different preprocessing technique combinations. Combination C outperforms the other combinations.

5.5 Hyperparameter Tuning

In Section 5.5, we utilize `GridSearchCV` in the `model_selection` submodule from `scikit-learn` to tune the hyperparameters of a classification three-hidden-layer feedforward neural network model. Grid search is done by performing a k-fold cross-validation on each combination of the hyperparameters. In this task, k is set to 5, and the average over the cross-validation is used to estimate the generalization performance.

```
param_grid = {
    'hidden_layer_sizes': [(8,8,8), (32,32,32)],
    'activation': ['relu', 'tanh'],
    'solver': ['sgd', 'adam', 'lbfgs'],
    'early_stopping': [True],
    'random_state': [4211]
}
```

The hyperparameters to tune are shown in the above code snippet. 12 combinations in total between different hidden-layer sizes, activation functions, and optimization algorithms are examined. `random state` is set to 4211 for reproducibility and `early stopping` is set to 'True' to avoid overfitting.

Data preprocessing is done according to the best preprocessing techniques in Section 5.1 and feature selection based on features selected in Section 2.4 for training and testing.

5.6 [Q27]

Table 12 shows the performance of the top five performing combinations of the hyperparameter settings in the grid search.

Rank	**Acivation function**	**Optimization Algorithm**	**Hidden-layer size**	**Mean Accuracy**	**Std Accuracy**
1	tanh	adam	(8,8,8)	0.870	0.010
2	relu	adam	(32,32,32)	0.868	0.012
3	relu	adam	(8,8,8)	0.865	0.011
4	relu	lbfgs	(8,8,8)	0.864	0.011
5	tanh	adam	(32,32,32)	0.863	0.016

Table 12: Performance of five combinations of the hyperparameter settings

Upon observing the results, it is shown that the accuracies are similar. One result that stood out is that four out of the top five performing combinations utilize the 'adam' optimization algorithm. 'adam' stands for Adaptive Moment Estimation, an extension of the stochastic gradient descent algorithm that adapts the learning rate dynamically for each parameter. This suggests that 'adam' outperforms the standard stochastic gradient descent.