Master’s Thesis 2023   30 ECTS
Faculty of Science and Technology

Extending the Repeated Elastic Net Technique for Multiclass Feature Selection: Performance Analysis and Comparison

Nissan Karki
Master of Science in Data Science

Preface

This thesis is written at the Faculty of Science and Technology at the Norwegian University of Life Sciences (NMBU) in 2023, signifying the conclusion of a two-year master's degree in Data Science.

I wish to express my deepest gratitude to my advisor, Associate Professor Oliver Tomic, for giving me a chance to delve into this significant topic for my master's thesis. His guidance, support, and critique have been invaluable on this journey. Equally, I would like to extend my appreciation to my co-advisors, Professor Cecilia Marie Futsæther and Postdoctoral Fellow Stefan Schrunner, for their insights and constructive feedback. I would also like to acknowledge the Orion High Performance Computing Center (OHPCC) at NMBU for access to the Orion computation cluster. Without their computation resources, this thesis would not have been possible.

A special acknowledgment goes to my classmates at NMBU for their camaraderie throughout this program. To my friends, family, and loved ones, for their unwavering support and words of encouragement, I am forever grateful.

Ås, May 15th 2023

Nissan Karki

Abstract

Feature selection is an integral part of data science, offering the ability to manage the high-dimensional challenges that sometimes curse machine learning. Although machine learning and deep learning models have significantly improved in the field, high-dimensional data still pose problems due to their increased computation and memory requirements. The “short-wide” nature of tabular datasets prevalent in real-world applications create underdetermined mathematical problems characterized by an excess of variables over equations. To address these challenges, feature selection techniques need to not only reduce the dimensionality but also ensure the stability of the selected features.

In this thesis, we extend the Repeated Elastic Net Technique (RENT) feature selection method, originally capable of operating on only binary classification and regression problems, to support multiclass problems. We explore the effectiveness of this new functionality on diverse datasets and compare it with other feature selection techniques. The extension demonstrates higher performance in certain datasets, particularly those with a relatively high number of classes and features. It does not consistently surpass the baseline across all datasets, however, it surpasses or is on par with other feature selection techniques.

RENT’s ability to substantially reduce both the number of features and the overall runtime highlights its potential as a valuable tool in the feature selection domain. The results of this study ultimately advance our understanding of the RENT algorithm and lay the groundwork for future research aimed at improving its application to a broader range of real-world classification problems.

Contents

Chapter 1

Introduction

1.1 Background and Context

Feature selection is a prerequisite in model building. It significantly impacts a model's performance, interpretability, and ability to generalize. Technologies to capture more data from both physical and logical mediums continue to evolve, and the volume, complexity, and dimensions of data in all domains continue to increase. The need for robust feature selection methods has become more pronounced in this data-heavy world. One such method is the Repeated Elastic Net Technique (RENT), which has demonstrated its effectiveness in handling binary classification and regression problems [1].

1.2 Problem Statement

The primary goal of this paper is to extend the RENT feature selection method to support multiclass problems, broadening its use into diverse areas. The current implementation of RENT only supports binary classification and regression tasks, which limits its domain of applications in the real world, unlike other techniques already present in literature [2][3][4][5]

1.3 Proposed Solution

We propose an adaptation of the RENT algorithm using a one-vs-rest approach. This is integrated into the logistic regression model employed by RENT for feature selection. The current implementation of RENT requires the user to provide three thresholds as hyperparameters: the percentage of non-zero weights of each feature, the frequency of coefficient alternation between positive and negative values across models, and the significance of coefficient deviation from zero [1]. In our proposed extension, we update the calculation of these thresholds to accommodate the one-vs-rest technique. This technique generates one model for each class in the target variable, as opposed to binary classification tasks where a single model and one set of weights are produced. This modification would allow RENT to handle multiple classes while preserving its core functionalities efficiently.

1.4 Paper Structure

The paper is structured as follows: Section 2 provides a background on the machine learning models and feature selection techniques, including RENT. Section 3 details the proposed extension of RENT for multiclass problems. Section 4 describes the methodologies applied. Section 5 presents the datasets and experimental setup of this paper. Section 6 provides the empirical evaluation of the proposed method and compares its performance with other techniques. These are further discussed in Section 7. Section 8 concludes the paper. And finally, Section 9 describes the potential avenues for future research.

The Python scripts for this study are present in the GitHub repository: https://github.com/karkinissan/Multiclass_RENT/tree/multi_class

Chapter 2

Theory

2.1 Models

2.1.1 Perceptron

A perceptron is used in supervised learning tasks for binary classification problems. It is a simple yet powerful algorithm that can be used to learn linear decision boundaries within data. Consider $X = \{\mathbf{x^i} \mid i \in \{1, \dots, k\}\}$ as a set of k training samples in a given dataset. Each sample within the dataset is represented as an n-dimensional vector denoted by $\mathbf{x^i} = [x_1, x_2, \dots, x_n]$. The components of this vector correspond to the *input features*. The input signal to the perceptron is a single instance of the dataset. Each component of the input vector is associated with a *weight* w_n. These set of weights $\mathbf{w}$ represent the importance or strength of the respective input features in determining the output prediction. The perceptron's weights are also referred to as its *model parameters*.

The perceptron algorithm computes a weighted sum of the input features and their corresponding weights [6]. This weighted sum is commonly referred to as the *net input*. A bias term w_0x_0 with $x_0 = 1$ is added into the net input to shift the decision boundary away from the origin of the input feature axis. The net input can be expressed as :

$$z = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n \tag{2.1}$$

$$z = \sum_{j=1}^{n}(w_j * x_j) + b = \mathbf{w}^T\mathbf{x} + b \tag{2.2}$$

In this equation, z denotes the net input, w represents the weights associated with the input features, and x the input features. n indicates the number of input features. It is important to note that w_0 is the bias weight, and x_0 is a unit value.

The weights are initially set to a random set of values [7]. During training, the weights are updated to minimize the error between the predicted output and the true output. The output y is determined with the use of a *threshold function*.

For classification tasks, the net input z is passed through a *threshold function* $\phi(z)$ to determine the perceptron's output prediction. Specifically, perceptron uses the *unit step threshold function* which is a mathematical function that takes the net input and produces a binary output of 0 or 1 based on whether the net input is above or below a certain threshold. In the context of the perceptron, the threshold function is used to determine whether the predicted output is a positive or negative class label, with 0 representing the negative class and 1 representing the positive class. For instance, if the threshold is set at 0, any net input greater than or equal to 0 is predicted as a positive class label, while any net input less than 0 is predicted as a negative class label. The unit step function is visualized in Figure 2.1.

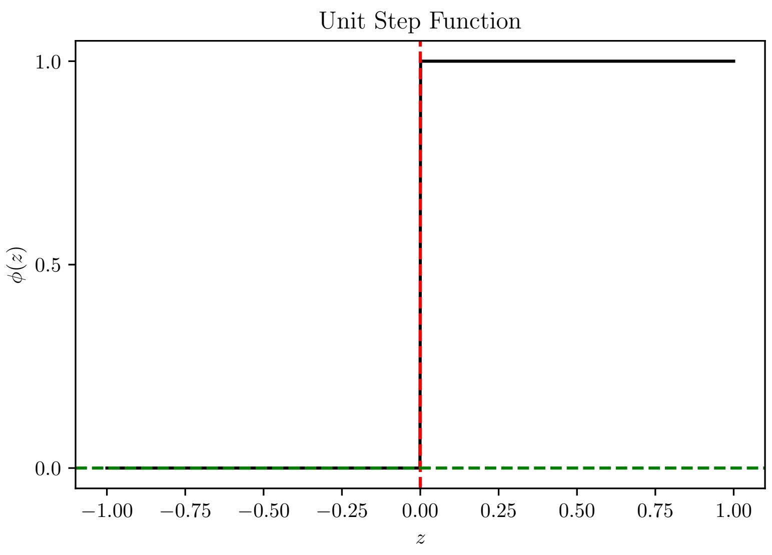


Figure 2.1: The unit step function with threshold at 0. Any value greater than or equal to 0 is set to 1, the rest are set to 0.

The output of the perceptron is the result of the unit step function and can be represented as:

$$\hat{y} = \phi(z) = \begin{cases} 1, & \text{if } z \geq \text{threshold} \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

Where $\hat{y}$ is the output, z is the net input, and the *threshold* is a predefined value for making the decision (0 in the example).

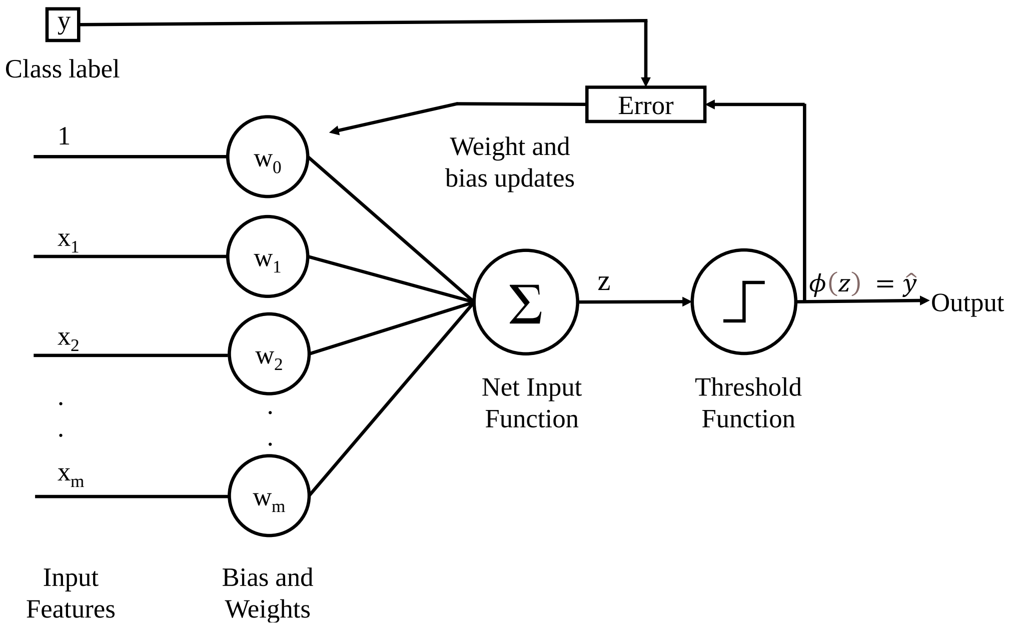


Figure 2.2: Perceptron model.

Figure 2.2 shows the perceptron model and the flow of information through it. During the training process, the perceptron iterates through the training data and adjusts its weights based on the errors made by the model on the predicted output. The weights are left alone if the prediction is correct. However, if the prediction is incorrect, the weights are adjusted so the model output better aligns with the expected output [6]. The learning rule of the perceptron is as follows:

$$w_j^{\text{new}} = w_j^{\text{old}} + \Delta w_j \tag{2.4}$$

where the weight adjustments Δw_j is given by:

$$\Delta w_j = \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)} \tag{2.5}$$

In these equations, Δw_j represents the change in weight for the j^{th} input feature, η denotes the learning rate (a hyperparameter controlling the size of the weight updates), $y^{(i)}$ is the true class label, $\hat{y}^{(i)}$ is the predicted class label for the i^{th} sample, and $x_j^{(i)}$ is the j^{th} input feature of the i^{th} sample.

A perceptron is a linear classifier, i.e., it can only learn linear decision boundaries. For more complex classification problems, other algorithms like support vector machines, random forest or KNN, or deep neural networks may be more appropriate [8]. Perceptron uses the difference between the predicted and actual output as a basis to update its weights. This can lead to sub-optimal convergence and instability in the learning process [9]. These issues are overcome in the Adaline model.

2.1.2 Adaline

Adaptive Linear Neuron (Adaline) in a binary classification model. It is similar to perceptron but differs in how it updates its weights during training. Adaline, as the name suggests, can only perform a linear separation of the data.

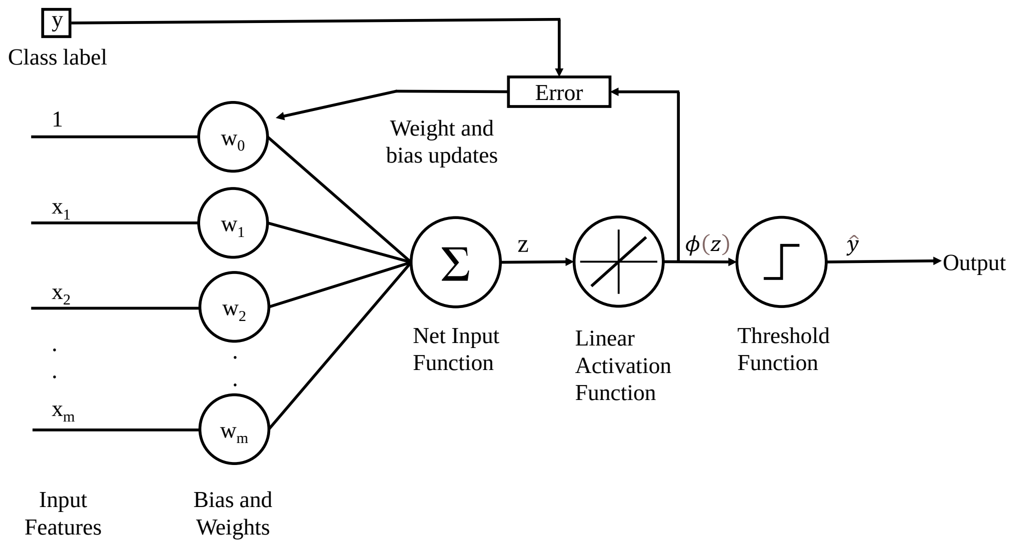


Figure 2.3: Adaline model. It consists of a linear activation function.

As shown in Figure 2.3, the Adaline algorithm uses one more component than the Perceptron: the activation function. The net input is passed through this activation function before it passes through the threshold function. Adaline uses a linear activation function, meaning that its output is a linear transformation of its input. It is a continuous activation function, allowing it to output real-valued predictions rather than just binary classifications like the Perceptron. The activation function can be represented as

$$f(x) = x \tag{2.6}$$

It can be represented graphically by Figure 2.4

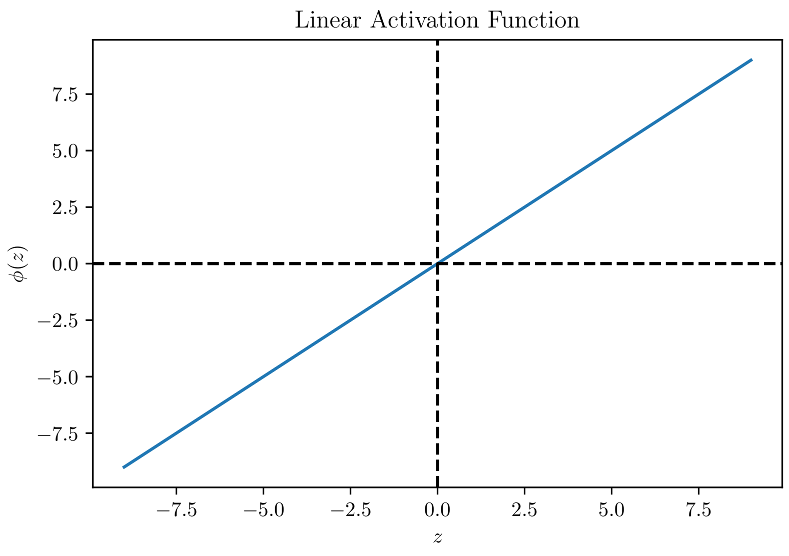


Figure 2.4: Linear activation function. It outputs the same values as the inputs.

Hence the output of the linear activation function in the Adaline model is the same as the net input.

$$\phi(z) = z = \sum_{j=0}^{n}(w_j * x_j) \tag{2.7}$$

During training, the weights of the Adaline model are adjusted using the gradient descent algorithm. This process minimizes the sum of squared error (SSE) between the predicted output and the output from the activation function [10]. This minimized function is referred to as the *cost function*. More details on the gradient algorithm are provided in Section 2.2.

The cost function can be expressed by the equation:

$$J(w) = \frac{1}{2}\sum_{i=1}^{k}(y^{(i)} - \phi(z^{(i)}))^2 \tag{2.8}$$

where $J(w)$ is the cost function, k is the number of samples, $y^{(i)}$ is the true class label of the i^{th} sample, and $\phi(z^{(i)})$ is the output of the activation function for the i^{th} sample.

The learning rule for the Adaline model is as follows:

$$w_j^{\text{new}} = w_j^{\text{old}} + \Delta w_j \tag{2.9}$$

$$\Delta w_j = -\eta \cdot \frac{\partial J(w)}{\partial w_j} \tag{2.10}$$

$$\Delta w_j = \eta \cdot \sum_{i=1}^{k}(y^{(i)} - \phi(z^{(i)}))x_j^{(i)} \tag{2.11}$$

While the linear activation function does allow Adaline to provide a continuous output, it is sensitive to outliers. This can be overcome by the logistic regression model.

2.1.3 Logistic Regression

Logistic regression, despite its name, is a binary classification model. It uses a non-linear activation function to compute the probability of the output label being 0, 1, or anything in between based on the linear combination of the features. The activation function used in this model is the *sigmoid function*, also known as the *logistic function*, shown in Figure 2.5. It is an S-shaped curve that maps the net input to a value in $(0, 1)$. This can be interpreted as the probability of a binary output [11].

The sigmoid function is defined as:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.12}$$

The function approaches 0 when x nears negative infinity. It rises smoothly to 0.5 when x equals 0, and approaches 1 when x nears positive infinity. The curve is

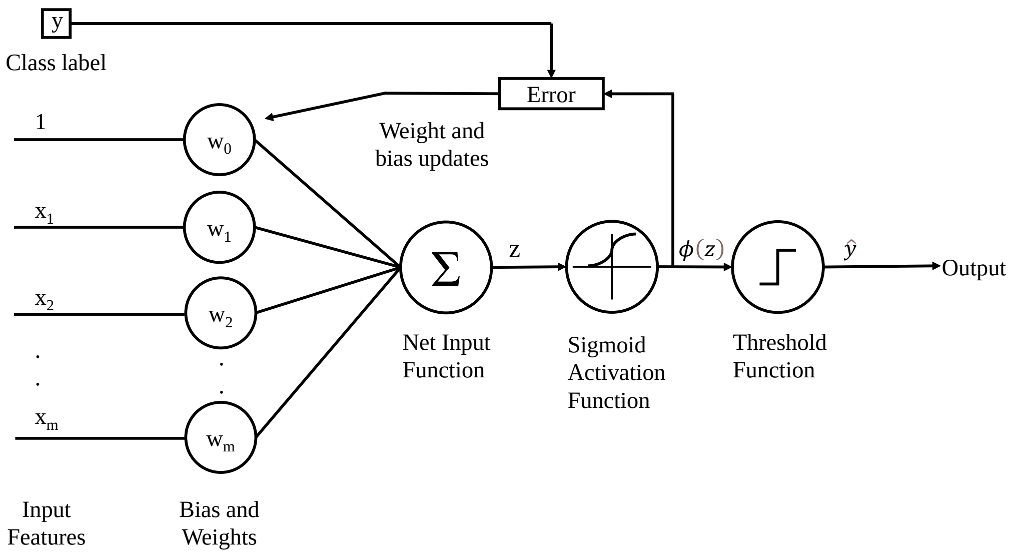


Figure 2.5: Logistic regression model and its use of a sigmoid activation function.

symmetric around $x = 0$ and has a maximum slope at $\phi(x) = 0.5$ or when $x = 0$. This feature makes it useful for modeling probabilities.

In the logistic regression model, the sigmoid function accepts in the net input and returns the output probability.

$$\phi(z) = p(y = 1|x) = \frac{1}{1 + \exp(-z)} \tag{2.13}$$

Where $p(y = 1|x)$ signifies the probability of the output variable being one given the input variables x, *exp* is the exponential function, and z represents the net input.

The logistic regression model is trained in the same way as the Adaline model. The objective is to minimize the cost function, which measures the difference between the predicted probabilities and the actual binary output values. It also uses the gradient descent algorithm for updating weights. The difference is in the logistic regression model's activation function and the cost function. A common cost function used for logistic regression is the *binary cross-entropy loss* [6], which is defined as:

$$J(\mathbf{w}) = \sum_{i=1}^{k} \left[-y^{(i)} \log(\phi(z^{(i)})) - (1 - y^{(i)}) \log(1 - \phi(z^{(i)}))\right] \tag{2.14}$$

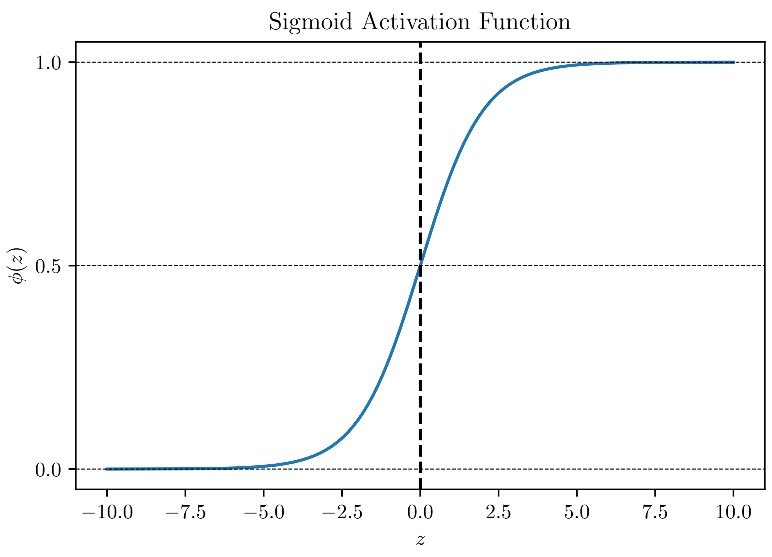


Figure 2.6: Sigmoid activation function.

Where $\phi(z(i))$ is the predicted probability, y is the label, and *log* represents the natural logarithm.

Logistic regression's use of the sigmoid function gives it several advantages over perceptron and Adaline. It outputs probabilities, which provide a better insight into the model's predictions. The sigmoid function is also smooth and differentiable, allowing for an easier application of the gradient descent algorithm during the learning process [9]. It also makes the model less sensitive to outliers compared to Adaline.

2.2 Gradient Descent

The gradient descent is an optimization algorithm commonly used in machine learning to minimize the cost function of a model. The goal of an optimization algorithm is to find the set of model parameters (or weights) that minimize the difference between the predicted output and the true output. Adaline and logistic regression use gradient descent during the learning process.

The cost function can be described by a convex shaped graph resembling a bowl. As the cost function is a function of a model's weights, the graph represents how the cost changes when the model's weights are adjusted. This is represented in figure 2.7.

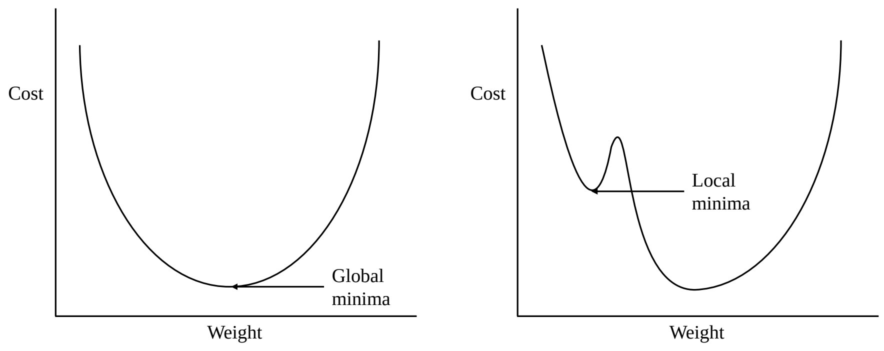


Figure 2.7: Cost functions. Left: an ideal cost function with a single global minima. Right: A cost function with a local and global minima.

The lowest points of the graph are where the cost function has the minimum value. This is referred to as the *global minima*. The goal of the gradient descent algorithm is to update the weights and get the cost to this point. However, depending on the data, there can be several *local minima*, which are the lowest points in their neighbourhood, but not the lowest point in the graph. The *gradient* refers to the slope at any point in the cost-weight graph. The magnitude of the slope determines the rate of change of the cost function with respect to the weights of the model. The gradient descent algorithm starts at a random point in the graph and iteratively adjusts the model's weights in the direction of, and proportionally to, the steepest descent of the cost function.

The idea behind gradient descent is to calculate the gradient (the first-order derivative) of the cost function with respect to each model parameter and update the parameters in the direction of the negative gradient [12]. This process is repeated until the cost function reaches a minimum or a convergence criterion is met. Figure 2.8 visualizes this process. Each iteration is a step and the step size is determined by the learning rate. A small learning rate may lead to slow convergence, while a large learning rate can push the algorithm beyond the optimal solution and even result in divergence [13] as shown in Figure 2.9. Hence the learning rate needs to be tuned to the particular problem.

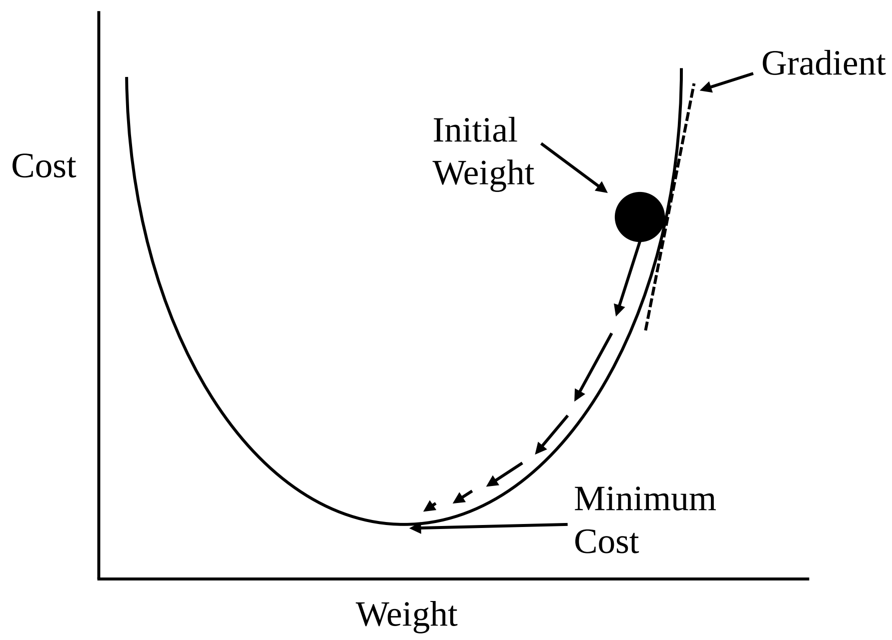


Figure 2.8: Gradient descent. On each iteration, the weights are updated in the direction opposite of the gradient.

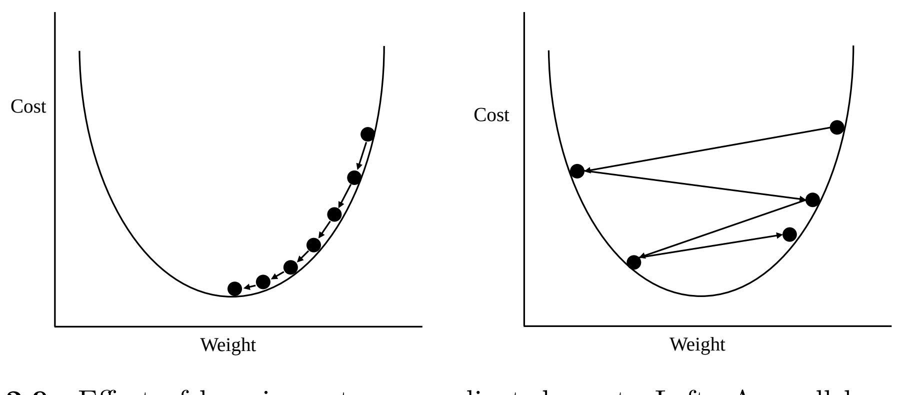


Figure 2.9: Effect of learning rate on gradient descent. Left: A small learning rate leading to a slow convergence. Right: A high learning rate resulting in large steps and divergence.

2.3 Regularization

Regularization is a technique applied to machine learning models to control their complexity. The complexity of a model is proportional to the number of features in the data and the range of values the model parameters attain during training [14]. Regularization works by introducing a penalty term to the loss function that the model is trying to minimize [9]. The penalty term ensures that the model does not assign excessively large or small values to its weights based on the requirements. The amount of regularization is controlled by a hyperparameter called the regularization strength. A higher regularization strength would prevent the model's weights from having very large values, while a lower regularization strength would allow for looser restrictions.

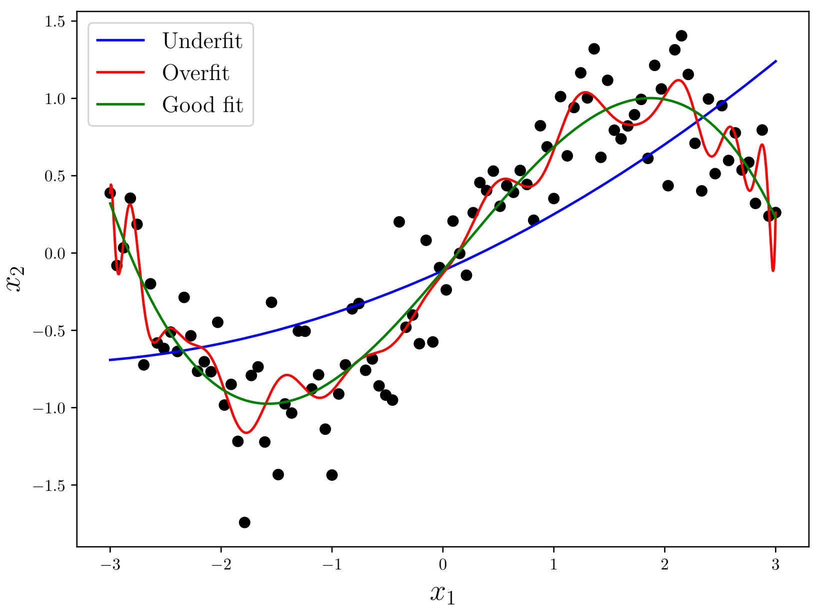


Figure 2.10: A visualization of overfitting and underfitting on data.

Regularization can prevent the overfitting or underfitting of a model on the training data. Overfitting occurs when a model is trained too closely on the training dataset and performs poorly on new, unseen data. This is referred to as the model *memorizing* the training data and not *generalizing* to the test data [15]. This happens due to model learning the noise present in the training data instead of the underlying patterns or signals. This is signified by the red line in Figure

2.10. Overfitting can be characterized by the model having high magnitudes for its weights. Underfitting occurs when a model cannot capture the underlying patterns in the training data and thus performs poorly on the training data as well as the unseen data. This is signified by the blue line in Figure 2.10. Underfitting can be observed in the model through the low variance in its weights during training.

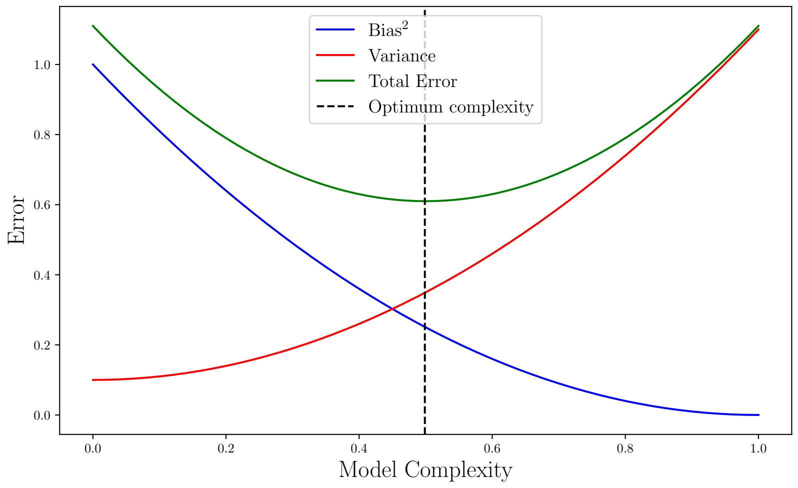


Figure 2.11: Bias-variance trade-off.

Regularization also helps balance the bias-variance trade-off. Bias refers to the error in the model's prediction. A model with high bias will not have properly captured the relationship between the input features and the output signal. This leads to underfitting. In contrast, a low bias means that a model has learned all complexities of the training data, which results in overfitting. Variance refers to how sensitive the model is to small changes to the input data. A model with high variance tends to fit the noise in the training data, leading to overfitting. A low variance means the model is too simple and has not learned enough from the training data [9]. A representation of bias and variance as function of the model complexity is given in Figure 2.11. A strong regularization term can reduce a model's bias but increase its variance, whereas a weak regularization can do the opposite.

The regularization term is typically a function of the model's parameters, such as the weights learning model. The penalty term is added to the cost function to encourage the model to choose solutions that are less likely to overfit or underfit on the training data.

Based on the penalty term used, the three most commonly used regularization techniques in machine learning are L1 (Lasso) regularization, L2 (Ridge) regularization, and elastic net regularization.

2.3.1 L2 Regularization

L2 regularization adds a penalty term proportional to the square of the magnitude of the model weights to the cost function, also known as the L2 norm of the weights [16]. A scaling parameter is multiplied by the L2 norm before adding it to the loss function. L2 regularization is also known as *Ridge regression.* Equation 2.15 shows the L2 regularization term added to the sum of squared error cost function.

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{k}\left(y^{(i)} - \sum_{j=0}^{n} w_j \cdot x_j^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=0}^{n} w_j^2 \tag{2.15}$$

The regularization strength is shown as the symbol λ (lambda) in the above formula and it is a hyperparameter (see section 4.5). As λ increases, the penalty for having large weights increases, and the model becomes more likely to have some coefficients with smaller weights. λ is multiplied by $1/2$ in the above equation for easier differentiation. Correlated features tend to have similar weights with L2 regularization [17].

Figure 2.12 shows the constraints L2 regularization places on the weights. The goal during learning is to be as close to the center of the cost function (visualized in red) as possible. The L2 norm limits the weight values within the boundary of the blue circle. The radius of the circle is controlled by the regularization strength lambda. The higher the strength, the smaller the radius of the circle.

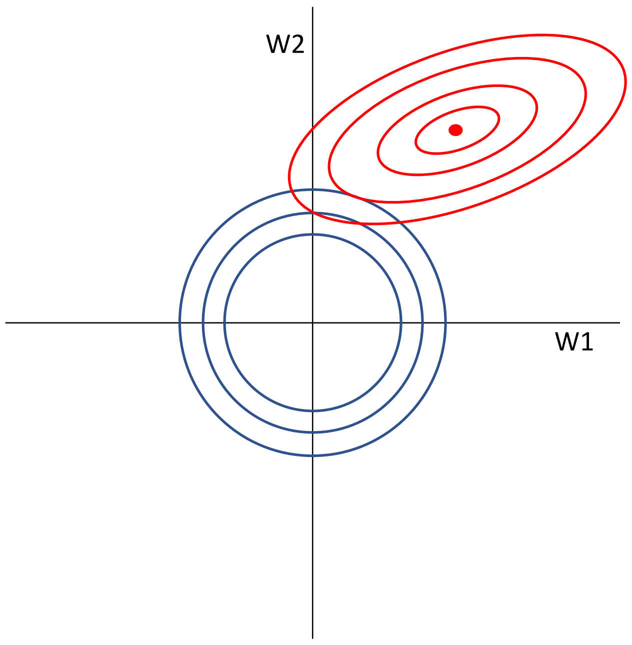


Figure 2.12: Visualisation of the regularization of the L2-norm.

2.3.2 L1 Regularization

L1 regularization, also known as Lasso regression, adds the absolute value of the model weights as a penalty term to the loss function as shown in equation 2.16. The penalty term is also called the L1-norm of the weights [16]. L1 regularization produces sparse weight vectors. It pushes some of the model's parameters to be exactly zero, resulting in them being removed from the model [17].

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{k}\left(y^{(i)} - \sum_{j=0}^{n} w_j \cdot x_j^{(i)}\right)^2 + \lambda \sum_{j=0}^{n} |w_j| \tag{2.16}$$

The larger the penalty term λ, the more likely the model parameters will shrink to zero.

Figure 2.13 shows the limitations placed on the weight values by the L1 norm. The size of the squares is inversely proportional to the regularization strength. The L1 norm ensures that the point on the square that is closest to the minima of the cost function is at the axes. Consequently, this leads to some of the weights being zeroed.

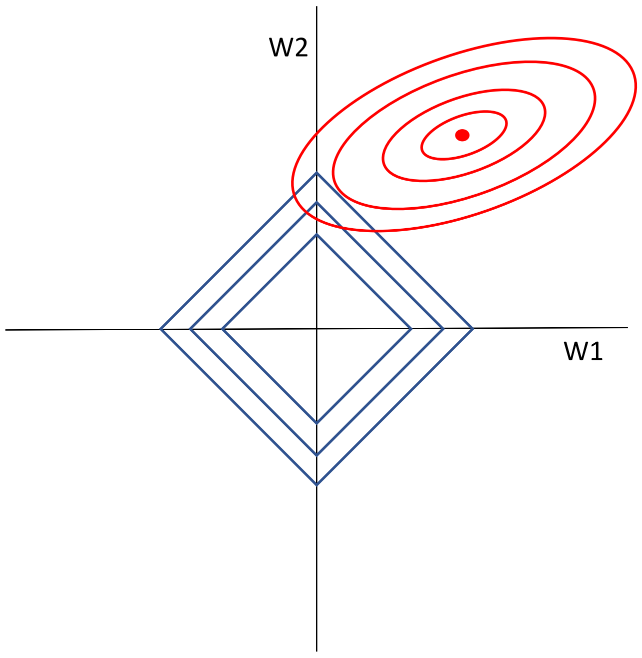


Figure 2.13: Visualisation of the regularization of the L1-norm.

L1 regularization is particularly useful when a dataset consists of many predictor variables, and only a subset is expected to be important for predicting the outcome. By introducing sparsity in the model, L1 regularization can reduce the model's complexity and improve its generalization performance on unseen data. L1 regularization shows erratic behavior when dealing with highly correlated features as their weights do not tend to achieve similar values during training [17]. This property depends on the dataset and the regularization strength applied to the model.

2.3.3 Elastic Net Regularization

Elastic net regularization is a compromise between L1 and L2 regularization. It combines the sparsity properties of Lasso with the regularization properties of Ridge, resulting in a more robust model [18]. This is done by adding both the L1 and L2 penalty terms to the loss function. The L1 term pushes the model to select a sparse set of input features by setting some weights to zero. The L2 term adds weight shrinkage to the model.

The elastic net regularization term is shown in equation 2.17.

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{k}\left(y^{(i)} - \sum_{j=0}^{n} w_j \cdot x_j^{(i)}\right)^2 + \lambda\left[\frac{\alpha}{2}\sum_{j=0}^{n} w_j^2 + (1-\alpha)\sum_{j=0}^{n}|w_j|\right] \qquad (2.17)$$

Where the symbol alpha α is the mixing parameter.

The mixing parameter α controls the amount of L1 or L2 regularization. When α increases, the L1 penalty becomes more dominant, and the model becomes more likely to have some weights equal to zero. As α decreases, the L2 penalty term becomes more dominant, and the model becomes more likely to distribute the weights evenly. When $\alpha = 1$, the elastic net becomes lasso regression; when $\alpha = 0$, it becomes ridge regression [17]. This mixing parameter allows an elastic net to be adaptable to various problems.

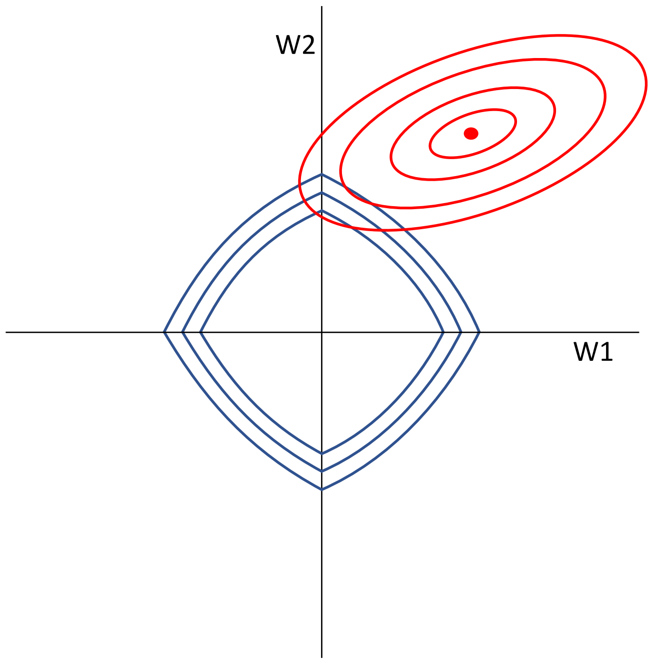


Figure 2.14: Visualisation of the elastic net regularization.

Figure 2.14 shows the constraint region for the elastic net regularization. Being a combination of L1 and L2 regularization, the boundary is in a shape somewhere between a circle and a square. This allows some of the weights to be sparse and some to be regularized.

Elastic net regularization is useful when there are many predictor variables, some of which are highly correlated [18]. It can select or discard these correlated features as a group, overcoming the weakness of the L1 regularization. However, between two penalty terms, an elastic net adds one more hyperparameter (α), and finding its optimal value, using cross-validation or other model selection techniques adds to the computation cost.

2.4 One-vs-rest Modelling Strategy

The one-vs-rest (OvR) modelling strategy, also known as the one-vs-all strategy, is used by binary classification models to tackle multiclass classification problems. This strategy creates a binary classifier for each class in the dataset. Each classifier is trained to distinguish samples of its specific class from all other classes [9].

The original multiclass dataset is divided into separate binary datasets, one for each class. Binary classifiers, such as Adaline or logistic regression, are trained on each dataset to distinguish the instance of that class from all other classes. The instances belonging to the target class are treated as positive samples, while the rest are treated as negative samples. This results in a set of binary classifiers equal to the number of classes in the training dataset. During testing or inference, the input is evaluated through each trained classifier. The class with the highest output score, determined through majority voting, is considered the predicted class.

The one-vs-rest technique is easy to implement, can be scaled to handle a large number of classes, and is compatible with a wide range of binary classifiers. Implementing it in unbalanced datasets can be problematic as it can lead to biased classifiers [9].

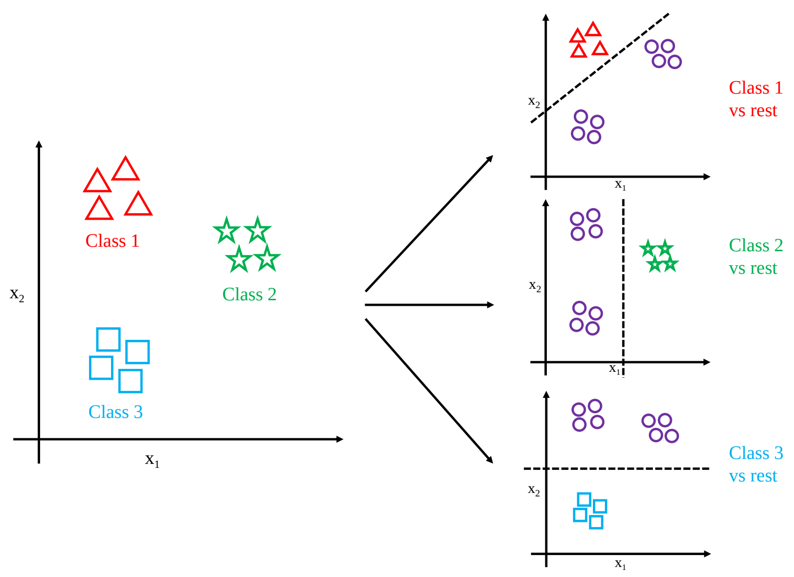


Figure 2.15: One-vs-rest technique. Three classifiers are trained to differentiate instances of a particular class, treated as positive samples, from instances of all other classes, treated as negative samples.

2.5 Feature Selection

Feature selection is the process of selecting a subset of relevant features from a dataset with the purpose of training a learning model on the reduced set. Due to the reduced dimension, feature selection lowers the complexity of the model. It can also improve the model's performance by eliminating irrelevant, redundant, or noisy features that could lead to overfitting [19]. Feature selection also improves interpretability, allowing a clearer understanding of the relationships between the input features and the model's predictions [20]. It helps identify the most important factors driving a model's performance, which can be crucial for decision-making. While feature selection has several benefits, it is often considered alongside another dimensionality reduction technique - feature extraction, which adopts a distinctly different approach.

Feature selection and feature extraction are terms often found together in literature [21][22][23]. They are both dimensionality reduction techniques; however, they differ in terms of their output features. Feature selection retains a subset of the original set of features in the data. As a result, the selected features retain their original values and interpretation [24]. Applications with high interpretability requirements would use feature selection. Feature extraction, on the other hand, generates a new set of features by applying some transformation to the original data [25]. Although this can preserve the information present in the original set of features, it may lead to a loss of interpretability [22]. Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) are popular examples of feature reduction techniques.

Feature selection techniques are generally divided into three categories: filter methods, wrapper methods, and embedded methods.

Filter methods rank features based on their relationship to the label without considering the relationships among other features [26]. The features are evaluated based on intrinsic metrics such as information gain, chi-square, or correlation coefficient. Filter methods have low computational costs as they do not employ any learning models. The features are ranked and selected based on some threshold. The threshold could be the desired number of features or a minimum metric to clear [27].

Wrapper methods search for the best-performing subset of features by training a model with them and evaluating their performance [28]. They take into account the interaction among features and their contribution to the model performance. This contrasts filter methods that assess the importance of features independently. They are computationally more expensive than filter methods due to the involvement of a learning model; however, they are more likely to produce a better subset of features than filter methods [29]. Techniques such as forward selection, backward selection, and recursive feature elimination are examples of wrapper methods.

Embedded methods have feature selection as part of a model's learning process [30]. Lasso regression, Ridge regression, and Decision trees are examples of embedded methods [31]. These techniques provide a good balance between the computational efficiency of filter methods and the performance of the wrapper methods.

This paper evaluates the feature selection capabilities of the Repeated Elastic Net Technique (RENT) with four other techniques.

2.5.1 Fisher's Score

Fisher's score selects features that maximize the separation between different classes and minimize the within-class variance [2]. The largest differences between the means of the classes determine the separation. Each feature is provided a "Fisher's Score," which is the ratio of the between-class variance and the within-class variance [32]. The features are selected in a greedy fashion, meaning those with the higher Fisher's score are determined to be more important. Being a filter method, Fisher's score is inexpensive in terms of computation costs. It is a widely-used feature selection technique for its performance and low cost. The fisher's score for a feature is calculated by the formula:

$$F = \frac{\sum_{m=1}^{M} n_m(\mu_m - \mu)^2}{\sum_{m=1}^{M} \sum_{x \in N_m} (x - \mu_m)^2} \tag{2.18}$$

where F is the Fisher's score, M is the number of classes, n_m is the number of samples in class m. μ_m is the mean of the samples in class m, μ is the overall mean and x is the sample belonging to class m.

The numerator signifies the between-class variance and is computed as the sum across all classes of the number of observations in each class multiplied by the square of the difference between the mean of the samples in the m^{th} class and the overall mean across all samples. The denominator is the within-class variance and it is computed as the sum, across all classes and all observations within each class, of the squared difference between each observation and its respective class mean [32].

2.5.2 Gini Score

The Gini score, also called the Gini index, is a metric used in decision trees for feature selection and node splitting. It measures the impurity of a feature, quantifying how poorly a feature can separate the instances between classes [3]. Hence, features with a low Gini score are selected as those would have higher class separability. The same criterion is also used for splitting the nodes of a decision tree. The Gini score is calculated as follows:

$$G(f) = \sum_{i=1}^{l} p_i * (1 - p_i) \tag{2.19}$$

Where $G(f)$ is the Gini score for feature f, l is the number of unique values for the feature f. p_i is the proportion of samples belonging to class i.

This study uses the `DecisionTreeClassifier` class from the scikit-learn library [33] with the `criterion` parameter set to `gini` to calculate the Gini scores.

2.5.3 Recursive Feature Elimination

Recursive feature elimination (RFE) works by iteratively removing features that are deemed less important until a desired number of features is reached, or a particular stopping criterion is met [4]. It is a wrapper-based method and thus requires the use of another learning algorithm to determine the relevant features. The relevancy could be derived from the model parameters and feature importance, depending on the model used. Recursive feature elimination starts with the model being trained on a set of features. The features are ranked according to the importance criterion chosen, and the least important feature is removed from the set. The process is repeated with now a subset of the feature set until a desired number of features is obtained, or some stopping criterion is met. Recursive feature elimination can get computationally expensive, as it requires training multiple models, especially for high dimensional data.

2.5.4 Random Forest Classifier

Random forest classifier (RFC) is an ensemble technique for classification tasks. It generates predictions by combining the results of multiple decision trees. Each decision tree is trained on a subset of the training data (with replacement) and a subset of the feature set [5]. The splitting criterion for the decision trees commonly used is the Gini index or entropy, both being a measure of the impurity of a node. The random forest classifier can output a feature importance ranking using this impurity metric from each decision tree. The importance is measured as the average decrease in impurity across all decision trees in the random forest. The features are then ranked according to their importance, and the desired number of features can be selected. This study uses the `RandomForestClassifier` class from the scikit-learn library for feature selection.

2.6 Repeated Elastic Net Technique

The repeated elastic net technique (RENT) aims to improve the stability and robustness of elastic net feature selection by repeating the elastic net process multiple times on different subsets of the data [1][34]. According to Jenul et al., "The feature selection is based on three criteria evaluating the weight distribution of features across all elementary models." [1] The three criteria determine: how frequently a feature gets selected across all models (τ_1), how often the feature weights alternate between positive and negative values (τ_2), and whether the feature weights differ significantly from zero (τ_3).

The RENT library is capable of performing feature selection on binary classification and regression problems. It uses the logistic regression model as the backend for binary classification problems, and linear regression for regression problems. This section shall describe the feature selection process for the classification problems.

For classification problems, RENT uses the logistic regression model with elastic net regularization. The core idea is to train k models on unique subsets of the data and investigate the statistics of the model coefficients across all models. The RENT package requires the user to define the search space in terms of the regularization strengths $\Lambda = [\lambda_1, \lambda_2, \ldots, \lambda_n]$ and the L1 ratios for the elastic net $A = [\alpha_1, \alpha_2, \ldots, \alpha_m]$. Other hyperparameters include k - the number of models to train, the range of the test size for the train-test splits, and the scoring metric for evaluating the models, all of which have default values defined in the module.

The RENT algorithm for classification problems works as follows:

1. Partition the training set k times using the train-test split to create k unique training and validation sets. The samples in each training and validation set are unique.

2. For every λ in Λ and α in A, train k logistic regression models with them as hyperparameters, one for every k^{th} subset from Step 1. Evaluate the models on their corresponding test sets.

3. For every λ and α:

   (a) Calculate the mean of the evaluation metric across all k models.

   (b) Calculate the fraction of weights that are zero across all model weights and across all k models.

   (c) Calculate the harmonic mean between the two values above. Call this the "score".

4. Find the combination of the λ and α with the highest "score" and select the set of k model weights.

5. Calculate the three feature selection criteria:

   (a) τ_1 - The fraction of non-zero weights of each feature across all k models.
   (b) τ_2 - The inverse of the rate of change of the sign of the weights of each feature across all k models.
   (c) τ_3 - The Student's t-test to determine whether the null hypothesis (H_0) can be rejected that the weights are equal to zero.

6. Select features that cross the user-defined thresholds (t_1, t_2, and t_3) for the three selection criteria from Step 5.

The total number of models to train would be $k * \lambda * \alpha$.

A feature is selected if the thresholds t_1, t_2, and t_3 of their respective hyperparameters τ_1,τ_2, and τ_3 are surpassed. The t values all range from 0 to 1. t_1 is the threshold for the fraction of non-zero weights of each feature. It refers to how often the k elastic net models select a feature. t_2 is the threshold for the fraction of weights having the same sign (either positive or negative) across the models. t_3 corresponds to the levels of statistical significance linked to the Student's t-test. For a significance level of 5%, the threshold would be 0.95. Similarly for a 1% significance level, threshold t_3 would be set at 0.99 [1].

Chapter 3

Proposed Solution

3.1 One-Vs-Rest Logistic Regression Model

The current version of RENT is limited to binary classification and regression tasks. This study aims to extend the capabilities of RENT by introducing support for multiclass classification problems. This is achieved by incorporating the one-vs-rest (OvR) technique into the logistic regression model used in the RENT algorithm. This is accomplished by explicitly configuring the `multi_class` parameter as `ovr` in the `LogisticRegression` class from the scikit-learn library.

For a given dataset with $m = 1, \ldots, M$ classes and $n = 1, \ldots, N$ features, the OvR-enhanced logistic regression model will possess (M, N) weights, as opposed to $(1, N)$ weights in the binary classification scenario. The scikit-learn implementation of the logistic regression model keeps the bias separate from the feature coefficients. Each of the M models of the OvR models will be referred to as a '*sub-model*' for the rest of the paper. For a binary classification problem, the weight matrix for K models is of shape (K, N) and for a multiclass classification problem with K models and M classes, the weight matrix is of shape (K, N, M). Consequently, this modification necessitates an update to the methods for calculating the three selection criteria τ_1, τ_2, and τ_3.

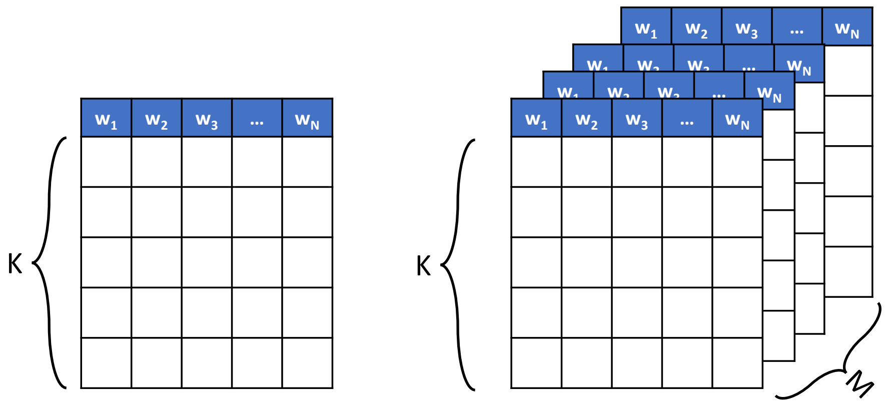


Figure 3.1: Left: N weights for K models in a binary classification problem. Right: N weights for K models and M classes for multiclass problems as a result of the one-vs-rest strategy.

3.2 Selection Criteria Updates

In order to adapt the calculations of the selection criteria to handle multiclass problems, several adjustments are required:

$\boldsymbol{\tau_1}$: The calculation of the fraction of weights that are non-zero for every n^{th} feature must consider the (K, M) weight matrix for each submodel, instead of the $(K, 1)$ weight vector for binary problems. The fraction of non-zero weights for the n^{th} feature is given as:

$$\tau_1(\mathbf{w_n}) = \frac{\sum_{k=1}^{K} \sum_{m=1}^{M} 1_{[w_{k,n,m} \neq 0]}}{K \times M} \tag{3.1}$$

Where $W_{k,n,m}$ represents the weight of the n^{th} feature of the k^{th} model and its m^{th} submodel, and $1_{[w_{k,n,m} \neq 0]}$ represents a function that is one when $W_{k,n,m}$ is not zero.

$\boldsymbol{\tau_2}$: When determining the proportion of positive or negative weights for a feature, it is necessary to consider the set of weights from each submodel separately. The weights for a given feature in one of the submodels could be all positive, while those in another could be all negative due to the nature of the learning process of the logistic regression model. Aggregating the counts into a single calculation would allow positive weights from one submodel to cancel out the negative weights from another submodel for the same n^{th} feature. Thus, separate analyses must be conducted for each submodel.

$$\tau_2(\mathbf{w_n}) = \frac{\sum_{m=1}^{M} \left| \sum_{k=1}^{K} \operatorname{sign}(w_{k,n,m}) \right|}{M \times K} \tag{3.2}$$

$$\operatorname{sign}(w_{k,n,m}) = \begin{cases} -1 & \text{if } w_{k,n,m} < 0, \\ 0 & \text{if } w_{k,n,m} = 0, \\ 1 & \text{if } w_{k,n,m} > 0. \end{cases} \tag{3.3}$$

$\boldsymbol{\tau_3}$: The Student's t-test requires the calculation of the mean and standard deviation of the weight values. The mean calculation must be updated to consider each submodel separately. As with τ_2, this is due to the potential for varying signs of weights in different submodels. Assessing the proportion of positive or negative weights and the mean weight coefficients individually for each submodel adapts the calculation for the OvR-based logistic regression models.

$$\mu(\mathbf{w_n}) = \frac{1}{M} \sum_{m=1}^{M} \left| \frac{1}{K} \sum_{k=1}^{K} w_{k,n,m} \right| \tag{3.4}$$

Similarly, the calculation of the standard deviation also needs to be updated. The standard deviation of each submodel is obtained separately, which are then averaged. Since standard deviation is always positive, there is no need to take an absolute of the values.

$$\sigma(\mathbf{w_n}) = \frac{1}{M} \sum_{m=1}^{M} \Big(\sigma(w_{1:K,n,m}) \Big) \tag{3.5}$$

With these updates, the RENT algorithm can support multiclass classification problems. The results of these changes are described in the Results section.

Chapter 4

Methods

4.1 Data Preprocessing

Data preprocessing involves the steps of cleaning, transforming, and organizing raw data into a structured format appropriate for learning algorithms. Enhancing data quality, reducing or removing noise, and resolving any inconsistencies in the data are the main goals of data preprocessing, which ultimately may lead to improved model performance.

4.1.1 Label Encoding

Machine learning algorithms require numerical data. While the scikit-learn library is able to handle string labels, there is a computation footprint associated with repeated conversions. It is more efficient to transform the labels prior to training and evaluation. Label encoding is used to transform categorical data into a numerical format. Each category value is mapped to a distinct integer value. A categorical column with n unique labels would be mapped to p unique integer values, typically ranging from 0 to $p-1$. Label encoding can add ordinal information to data that did not originally contain it [35]. This can lead to bias in a learning algorithm by misleadingly associating a higher number with higher importance. Hence, label encoding is reserved for the target variables where the ordinal property does not affect the learning process. For categorical features, using one-hot encoding is preferred.

4.1.2 One-hot Encoding

One-hot encoding, like label encoding converts categorical data into a numerical format like label encoding. It achieves this in a different way by creating binary columns for each category of a given feature [6]. It first identifies the unique values within the categorical feature, creates a new binary column for each unique category, and finally assigns a value of 1 to the binary column corresponding to the category present in the original data and 0 to the other columns.

Unlike label encoding, one-hot encoding does not suffer from the problem of ordinality, making it suitable for categorical variables where no inherent order exists. However, it can significantly increase the dimensionality of the data when dealing with a large number of categories. This can lead to increased computational complexity and the curse of dimensionality [36], potentially affecting model performance. It is also not an efficient representation for the categorical features with many categories as it creates a sparse binary vector, increasing memory requirement.

4.1.3 Standardization

Feature scaling is one of the most crucial transformations that needs to be applied to the data. It transforms the input features in the data to normalize their range and prevent features with high magnitudes from dominating the features with smaller magnitudes. Feature scaling also enhances the performance of various machine learning algorithms that rely on calculating distances or gradients [37]. This study uses standardization as the method of feature scaling.

Standardization, also known as z-score normalization, re-scales the features to have a zero mean and unit standard deviation (or variance). It is applied to each feature variable separately. The idea is to subtract the mean of the feature from each data point and divide the result by the feature's standard deviation. This centers the features around zero. The following formula achieves this:

$$x^*_{i,n} = \frac{x_{i,n} - \mu_n}{\sigma_n} \tag{4.1}$$

In this equation, $x_{i,n}$ is the i^{th} sample of the n^{th} feature, $x^*_{i,n}$ is the standardized version of the $x_{i,n}$, μ_n is the mean and σ_n is the standard deviation of all samples in feature n.

Standardization is less sensitive to outliers and does not bind the data to a specific range [38].

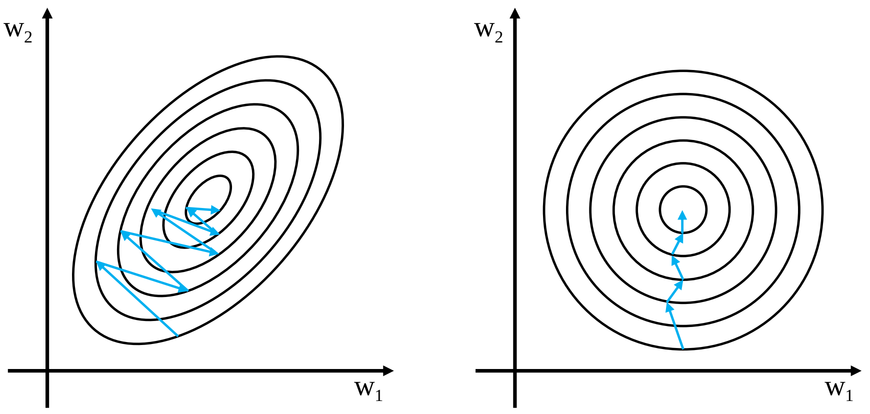


Figure 4.1: Left: Cost function of a non-standardized dataset. The arrows indicate the direction of gradient descent, describing an oscillating behavior. Right: Cost function of a standardized dataset. The arrows describe the more straight direction of gradient descent.

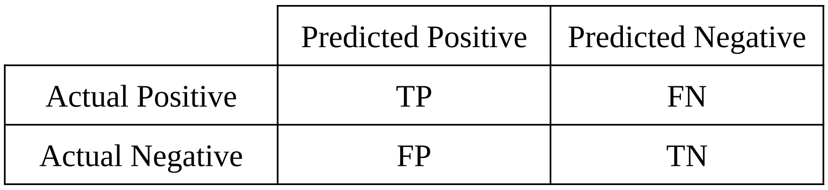
When features have different scales, the cost function will have an irregular and elongated shape (left image of Figure 4.1). The gradient descent algorithm may take longer to converge in these cases as it oscillates across steeper slopes. With standardization, all features are transferred into the same scale, which leads to a more symmetrical and balanced landscape of the cost function (right image of Figure 4.1). This allows the gradient descent algorithm to converge faster as it can go straight toward the minimum [38].

4.2 Performance Evaluation

The choice of the performance metric depends on the characteristics of the dataset. Each metric measures a different aspect of a model's performance. To gain a comprehensive understanding of a model's effectiveness, it is recommended to calculate multiple metrics. This approach provides a more nuanced perspective, allowing for a thorough analysis of the model's strengths and weaknesses.

4.2.1 Confusion Matrix

A confusion matrix is a table that shows the number of correct and incorrect predictions for each class in the dataset. It can help identify the model's strengths and weaknesses for every class and gain insights into misclassifications. Four metrics can be derived from the confusion matrix - true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN). Precision, recall, F1-score and Matthews correlation coefficient are derived from these metrics providing a more comprehensive understanding of the model performance. Figure 4.2 shows the confusion matrix for a binary classification problem. A confusion matrix can also be constructed for a multiclass problems. An M-class classification problem would have a $(M \times M)$ confusion matrix, with each cell (i, j) representing the number of samples from class i that were predicted as class j.



	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

Figure 4.2: Confusion matrix for a binary classification problem.

4.2.2 Accuracy

Accuracy is the ratio of correctly classified samples to the total number of samples. It is a commonly used metric in the field of machine learning. However, it may not be the best choice for problems where the data is imbalanced as it will favor the majority class and be less sensitive to the performance of the minority class. In a dataset where 97% of the samples belong to the majority class and a machine learning model that predicts all samples as the majority class, the accuracy would be 97%. In terms of the confusion matrix metrics, accuracy can be described by the formula:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}} \tag{4.2}$$

4.2.3 Precision

Precision measures the proportion of correctly identified positive samples among those predicted as positive by the model. It is the ratio of the true positive to the sum of the true positive and false positive instances:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4.3}$$

A high precision means the model does a good job of avoiding false positives. However, precision does not consider false negatives, so it is to be used in conjunction with other metrics.

4.2.4 Recall

Recall, also known as sensitivity, is the ratio of the true positive to the sum of the true positive and false negative. It measures the proportion of positive samples that the model correctly identified:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4.4}$$

A high recall means that the model is good at identifying positive samples. In situations where it is more important to identify all positive instances, even at the cost of including some false positives, the recall would be one of the metrics to use.

4.2.5 F1-score

F1-score is the harmonic mean of precision and recall. It provides a measure that takes both precision and recall into account, providing a trade-off between the two.

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.5}$$

F1-score is helpful when dealing with imbalanced datasets and when both false positives and false negatives are to be considered. The default `f1_score` method in the scikit-learn package only takes into account the results for the positive class when the problem is a binary classification task. For multiclass classification, the scores are generally averaged across all classes. The two most common averaging techniques are the micro and macro averages. Micro F1-score is calculated across the entire dataset, giving equal weight to each sample; however, this score can be dominated by the majority class. The macro F1-score, on the other hand, is calculated across every class individually and then averaged. In this case, the minority class is given the same weight as the majority class, which may not be desirable based on the problem [39].

4.2.6 Matthews Correlation Coefficient

The Matthews Correlation Coefficient (MCC) is less sensitive to imbalanced datasets and provides a more informative representation of the classification performance. Unlike the F1-score, it also considers all four confusion matrix metrics, including the true negative (TN), providing a balanced evaluation of a model's performance [40].

$$\text{MCC} = \frac{(TP \times TN - FP \times FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{4.6}$$

The MCC score ranges from -1 to $+1$. A score of $+1$ indicates perfect classification, 0 indicates random classification, and -1 indicates inverse classification.

4.3 Model Validation

4.3.1 Train-test Split

A model that has been trained over an entire dataset could risk overfitting if there is no separate test set for evaluation. A dataset without a predefined test set can be partitioned into two distinct subsets: a *training set* and a *test set*. The test set can be used to assess the performance of the model and ensure that it generalizes effectively to unseen data.

The ratio of the train-test split is a user-defined hyperparameter. Commonly used ratios include 70:30, 80:20, or 90:10. The proper ratio would depend on the size of the training data. If the data has, say, a thousand samples, then the 70:30 ratio may be a good fit, but if the data has a billion samples the 90:10 ratio or even a 99:1 ratio may be sufficient. To avoid bias in a classification problem, it is necessary to maintain the same distribution of the target variable across the splits. This is accomplished using *stratified sampling.*

After the split, the model is trained on the training set and evaluated on the test set while assessing its performance using appropriate evaluation metrics such as accuracy, F1-score, or the root mean squared error (RMSE) based on the nature of the problem. The model's hyperparameters are fine-tuned, or the feature selection strategy is adjusted based on the evaluation metric. Once the model achieves a desired performance, it is retrained with the full dataset while retaining the optimal hyperparameters.

Some processes use a *validation set* in addition to the training and test sets. The validation set is used to tune the hyperparameters of a model. The model is initially trained on the training set and evaluated on the validation set. Adjustments to the hyperparameters are made until the desired performance is achieved. The model is finally tested on the test set as usual. This extra set for evaluation ensures that the model is not tuned to the point of overfitting on the test data.

4.3.2 Cross-Validation

Cross-validation provides a more robust and comprehensive assessment of a model's performance than the train-test split when little data is available. The central idea behind it is to split the dataset into numerous smaller subsets and iteratively train and evaluate a model on various combinations them.

K-fold cross-validation

In K-fold cross-validation, the dataset is partitioned into k random subsets or "*folds*" that are equal in size. A model is trained and evaluated k times on the same set of folds; however, each time, the model is tested on the k^{th} fold while trained on the remaining $k-1$ folds. This is visualized in Figure 4.3. The performance metrics from each iteration are averaged to provide the final performance metric. The variance of the metrics is also measured to observe the stability of the performance. The dataset is randomly shuffled to ensure that the data points are not arranged in a particular order, possibly creating a bias in the model. K-fold cross-validation is more robust as it uses multiple train-test splits reducing the risk of overfitting on the test set [41]. However, it does add increased computational complexity to the modeling process as the model is trained and evaluated k times. This can also lead to some noise in the predictions and metrics as each model works with different subsets of data.

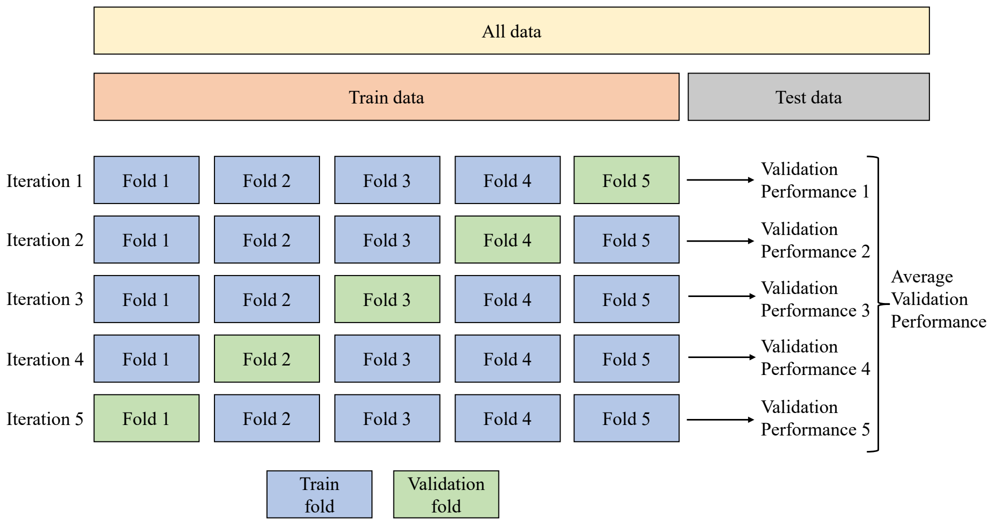


Figure 4.3: K-fold cross-validation with 5 folds.

Repeated K-fold cross-validation

Repeated K-fold cross-validation is an extension to the K-fold cross validation, where the process is repeated L times, resulting in $(L \times K)$ rounds of training and evaluation. Each repetition is done with new random splits of the data. This technique can provide an even more robust estimate of the model performance. However, it does come with an increased computation cost over its non-repeated counterpart.

Stratified k-fold cross-validation

Similar to the stratification process in stratified train-test split, stratified k-fold cross-validation ensures that the class distribution in the data is maintained for each fold during cross-validation. As a result, the performance estimate is more reliable and unbiased because each fold retains approximately the same class distribution as the original dataset.

4.4 Synthetic Dataset Generation

Synthetic dataset generation is the process of creating artificial datasets that resemble the characteristics of real-world data. Synthetic data can help augment existing datasets, obfuscate data to preserve privacy and ensure confidentiality, and test the performance of machine learning models under various conditions [42]. The sensitivity of models to various factors like noise, redundancy, missing values, outliers, or class imbalance can be evaluated by generating data with those properties resulting in more robust models. The `make_classification` function from the scikit-learn library generates binary or multiclass classification datasets with a specified number of samples, features, classes, informative features, redundant features, and more [43]. It also provides control over the degree of class separability, class weights, and scale. This paper evaluates the efficacy of the proposed multiclass RENT approach using synthetic multiclass classification datasets generated using the scikit-learn library.

4.5 Hyperparameter Tuning

Hyperparameters are model parameters that are not learned during training but set beforehand by the user. They control the overall behavior and complexity of the model and can impact its performance significantly [44]. In a logistic regression model, common hyperparameters are the learning rate, type of regularization, strength of regularization, and the optimization algorithm.

Hyperparameter tuning is the process of determining the optimal set of hyperparameters that leads to the best possible performance for a given model and dataset. This can be achieved using various techniques like grid search, random search, or Bayesian optimization algorithms [45][46]. The right set of hyperparameters strikes a balance between the model overfitting on the training set and its generalization ability on the test set. This study uses grid search to perform hyperparameter tuning.

Grid search searches through all possible combinations of a predefined set of hyperparameter values [38]. A model is trained on each set of hyperparameters and its performance is evaluated on the validation or test set using some scoring metric, such as accuracy or RMSE, depending on the type of problem. The hyperparameters of the model with the highest evaluation performance are selected as optimal.

Grid search is simple to implement but comes with a computational cost, especially with high-dimensional data. This study uses the `GridSearchCV` class from the scikit-learn library to implement grid search. As the name suggests, it can perform cross-validation during the hyperparameter search. This comes with an even higher performance cost but produces hyperparameters that are less likely to be biased.

4.6 Benchmarking

The performance of the feature selection techniques is tested on a logistic regression model with its hyperparameters tuned. To ensure a thorough analysis, this study measures the computational runtime associated with each technique. The runtime is measured using Python's `timeit` library [47]. To get a more reliable measure of the runtime, each model is trained and evaluated separately a total of ten times, and the duration of the run is tracked for every instance. Background processes running on a machine can affect the runtimes. To account for this, the minimum runtime across the ten trials is chosen to represent the runtime.

4.7 Jaccard Index

The Jaccard Index, also known as the Jaccard similarity coefficient, is a measure of the similarity across two sets of data. It is the ratio of the intersection to the union of two sets and ranges from 0 to 1, with values closer to 1 reflecting sets with more similar items. The Jaccard index has been used in several fields of machine learning, such as clustering [48] and recommender systems [49]. This study uses the Jaccard index to determine the similarity of the sets of features selected by each feature selection algorithm that was tested. For two sets, A and B, the Jaccard index is calculated as

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.7}$$

Chapter 5

Datasets and Experimental Setup

5.1 Datasets

This study uses six datasets to evaluate the performance of RENT and other feature selection techniques on multiclass classification problems. These datasets were chosen to represent several levels of complexity, dimensionality, number of classes, and real-world applications. The first three datasets are synthetic. They were generated using the scikit-learn Python library. The latter three are real-world datasets from different fields of research.

5.1.1 Synthetic Dataset 1 (SD1)

SD1 was generated using the `make_classification` function from the scikit-learn library. This dataset consists of 50 features, 2000 samples, and 3 classes. It was chosen to evaluate the performance of RENT on a relatively low-dimensional dataset with a fair number of samples and classes. 10 out of the 50 features are informative. The remaining features are random values or linear combinations of the 10 informative features. 3 classes were chosen to make it a multiclass dataset. All these configurations were set using the arguments of the `make_classification` method. During the development of the multiclass support for RENT, SD1 was used for testing and debugging due to the simplicity offered by its smaller size. The dataset is also evenly distributed to ensure no class is under or over-represented.

5.1.2 Synthetic Dataset 2 (SD2)

This dataset has 100 features, 2000 samples, 10 classes, and 10 informative features. It was also generated using the `make_classification` method. SD2 was created to evaluate the performance of the RENT on a dataset with a higher number of features and classes while still maintaining the same number of samples. The class distribution is also balanced.

5.1.3 Synthetic Dataset 3 (SD3)

SD3 consists of 1001 features, 500 samples, 3 classes, and 10 informative features. Also generated using the `make_classification` method, SD3 was created to evaluate the performance of RENT on a wide dataset, one that has more features than samples. The number of classes is set to 3 for easier testing and debugging. The class distribution is balanced.

Setting the `shuffle` flag to `False` in the `make_classification` method ensures that the features are ordered in the generated dataset. The informative features are always at the beginning. Hence, the first ten features in the SD1, SD2, and SD3 datasets are informative. This experiment sets the rest of the parameters to default. As a result, the two features in these datasets following the first ten are derived by linear combinations of the informative features. Any features after those would be random noise.

5.1.4 Modified National Institute of Standards and Technology (MNIST) Dataset

The MNIST dataset is a widely used benchmarking dataset in the field of machine learning [50]. It is a collection of handwritten digits that are 28×28 pixels in size. There are images for all 10 digits between 0 and 9. Each image is normalized with pixel intensity values in $[0, 255]$. It consists of 60,000 training samples and 10,000 testing samples. In this study, a smaller subset of 12,000 were used from the training data to save on computation time. To make the dataset compatible with RENT, the images were flattened, resulting in 784 features per sample. The MNIST dataset was chosen to evaluate the performance of the model on a real-world high-dimensional dataset with a large number of samples and classes.

5.1.5 Glass Dataset

The glass dataset comprises 10 features, 214 samples, and 6 classes. It represents the chemical composition of different types of glass in terms of their oxide content [51]. This dataset was chosen to test the performance of RENT on real-world low-dimensional data with a limited number of samples and classes.

5.1.6 ISOLET Dataset

The ISOLET dataset consists of 617 features, 7797 samples, and 26 classes. It is a speech recognition dataset consisting of spoken letters from the English alphabet, hence the 26 classes [52]. Each letter was spoken twice by every one of the 150 subjects. Three samples were dropped due to recording issues. The features consist of sound wave properties such as discrete Fourier transformation coefficients, the waveforms' pitch and amplitudes, and the speech's duration, among others. This dataset was chosen to evaluate the performance of RENT on real-world high-dimensional data.

Table 5.1 summarizes all datasets used in this paper.

Table 5.1: Summary of all datasets used in this study.

Dataset	**# Features**	**# Samples**	**# Classes**	**# Training samples**	**# Test samples**
SD1	50	2000	3	1340	660
SD2	100	2000	10	1340	660
SD3	1001	500	3	335	165
MNIST	784	22000	10	12000	10000
Glass	10	214	6	143	71
ISOLET	617	7797	26	6238	1559

Using these six datasets, this paper aims to assess the effectiveness and robustness of RENT for multiclass classification problems on various degrees of dimensionality, complexity, and number of classes. The diverse datasets provide a better understanding of the strengths and limitations of the RENT and ensure the generalizability of the technique.

5.2 Experimental Setup

All datasets were standardized prior to training. Categorical columns were one-hot encoded, and class labels were label encoded where applicable.

RENT requires the user to provide the three t threshold values as hyperparameters. After a number of experiments, it was determined that t_1 and t_2 could be set to the inverse of the number of classes in the dataset $(1/M)$, as those were observed to be promising thresholds. t_3 was set to the default value of 0.975, as defined in [1].

Two machines were used for this study. One was the author's personal laptop, a machine with an AMD Ryzen 9 5900HS 3.2 GHz CPU and 2x8GB of 3200 MHz DDR4 RAM running in dual-channel. The second machine was the NMBU Orion cluster. A total of 64 CPU cores and 32 GB of RAM were utilized for the experiments. All experiments pertaining to feature selection were run on the Orion cluster, while the baseline and final evaluation of the selected features were run on the laptop.

The feature selection through RENT was done for the following set of hyperparameters:

- The elastic net regularization strength, $C = [0.001, 0.01, 0.1, 1, 10, 100]$
- The elastic net alpha parameter, $l1\_ratios = [0, 0.25, 0.5, 0.75, 1]$
- The number of train-test splits, $K = 100$

The same hyperparameters were applied for all datasets.

5.3 Reproducibility

To ensure that the results of this paper are reproducible, the random seed was fixed for the Python `random` library, the `numpy` library, and all processes and models used from the `scikit-learn` library.

Chapter 6

Results

6.1 Data Presentation

All datasets except the MNIST and ISOLET were split into training and test sets using a 67/33 split ratio with stratification. The MNIST and ISOLET data came with a test set already present. Feature selection was only performed on the training set to prevent information leakage, and the selected features were applied to the test set for the final evaluation. The evaluation metrics micro-averaged F1-score (F1 micro), macro-averaged F1-score (F1 macro) and Matthew's correlation coefficient (MCC) are used to compare the selected features. The class distributions of all datasets can be found in Appendix B.

6.2 Baseline

In this section the baseline numbers for model performance and runtime are presented.

6.2.1 Baseline Performance

The baseline scores for the datasets were obtained using the logistic regression model implemented in the scikit-learn library with the one-vs-all approach and trained on the entire feature set. The hyperparameters for the logistic regression method were tuned using the *GridSearchCV* method from the scikit-learn library for each dataset. The tuning was done on the parameters:

- *C*: The inverse of the regularization strength,
- *penalty* : The regularization technique to apply to the model
- *solver*: The optimization algorithm.

The grid search was performed using stratified K-fold cross-validation with five folds. The logistic regression models with the best hyperparameters were trained again on each dataset and evaluated to get the scores in Table 6.1.

Table 6.1: Summary of all datasets used in this study.

Dataset	**F1 micro**	**F1 Macro**	**MCC**
SD1	0.8045	0.8041	0.7077
SD2	0.3273	0.3184	0.2531
SD3	0.6970	0.6961	0.5492
MNIST	0.9153	0.9141	0.9059
Glass	0.5915	0.4617	0.4396
ISOLET	0.9532	0.9530	0.9513

6.2.2 Baseline Runtime

Table 6.2 shows the training and inference runtimes of the datasets with the complete feature set. The numbers were obtained through benchmarks on the logistic regression model.

Table 6.2: Baseline training and inference runtimes.

Dataset	Training Runtime (sec)	Inference Runtime (ms)
SD1	0.021	0.105
SD2	0.633	0.2499
SD3	0.22	0.2544
MNIST	9.24	14.6104
Glass	0.015	0.0476
ISOLET	4.25	1.3833

6.3 Selected Feature Counts

All the other techniques tested in this paper: the Fisher score, Gini score, recursive feature elimination, and random forest classifiers require the user to specify the number of features to be selected. This is not the case with RENT since it uses the elastic net regularization . RENT selects features whose selection criteria (τ) cross the three t thresholds. Hence, the feature selection is performed using RENT first, and the same number of features are selected from the other techniques for a fair comparison of their relative performance. Table 6.3 shows the number of features selected by RENT.

Table 6.3: Feature reduction from the multiclass RENT algorithm. Columns t_1 and t_2 refer to the threshold values used for each dataset. t_3 was fixed to 0.975 as mentioned in Section 5.2.

Dataset	Original Features	Selected Features	Percentage Reduction	t_1	t_2
SD1	50	9	82%	0.3333	0.3333
SD2	100	13	87%	0.1000	0.1000
SD3	1001	7	99.3%	0.3333	0.3333
MNIST	784	263	66.45%	0.1000	0.1000
Glass	10	4	60%	0.1667	0.1667
ISOLET	617	375	39.22%	0.0385	0.0385

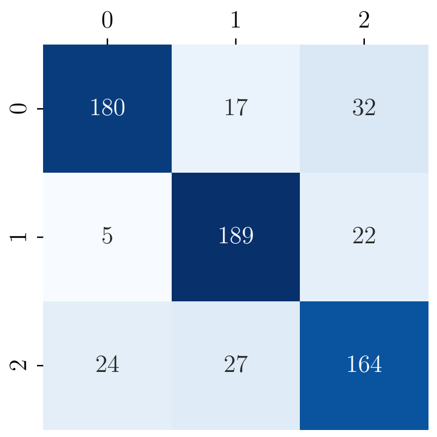
6.4 Model Performance

Table 6.4 displays the performance scores of the multiclass RENT feature selection method and compares it with the other techniques. The performance was evaluated for each dataset on logistic regression models whose hyperparameters were tuned for the subset of features selected by each feature selection technique. RENT is often the best feature selector or ties with the next best selection method.

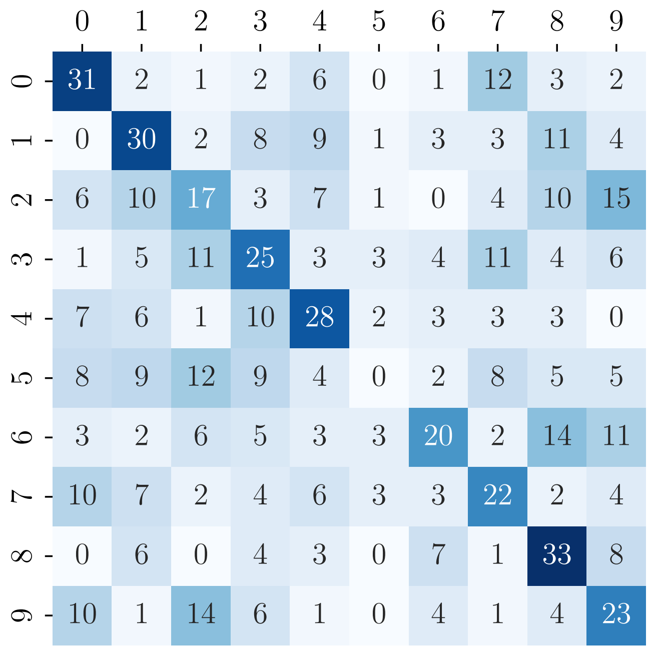
The confusion matrices for the best performing models are shown in Figure 6.1. The remainder are present in Appendix C.

Table 6.4: Comparison of performance metrics of different feature selectors.

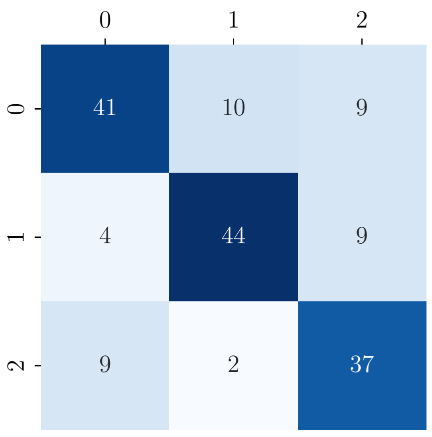
Dataset	**Feature Selector**	**F1 Micro**	**F1 Macro**	**MCC**
SD1	None (Baseline)	0.8045	0.8041	0.7077
	RENT	**0.8076**	**0.8071**	**0.7124**
	Fisher's score	**0.8076**	**0.8071**	**0.7124**
	Gini Score	0.803	0.8022	0.7054
	RFE	**0.8076**	**0.8071**	**0.7124**
	RFC	**0.8076**	**0.8071**	**0.7124**
SD2	None (Baseline)	0.3273	0.3184	0.2531
	RENT	**0.347**	**0.3304**	**0.2757**
	Fisher's score	0.3439	0.3367	0.2714
	Gini Score	0.3424	0.3327	0.2698
	RFE	0.3439	0.3367	0.2714
	RFC	0.3439	0.3367	0.2714
SD3	None (Baseline)	0.697	0.6961	0.5492
	RENT	**0.7697**	**0.7696**	**0.657**
	Fisher's score	**0.7697**	**0.7696**	**0.657**
	Gini Score	0.7394	0.7401	0.6106
	RFE	0.7273	0.7282	0.5955
	RFC	**0.7697**	**0.7696**	**0.657**
MNIST	None (Baseline)	**0.9153**	**0.9141**	**0.9059**
	RENT	0.892	0.8904	0.88
	Fisher's score	0.8729	0.8711	0.8587
	Gini Score	0.869	0.8673	0.8544
	RFE	0.8527	0.8505	0.8363
	RFC	0.8612	0.8593	0.8457
Glass	None (Baseline)	**0.5915**	**0.4617**	**0.4396**
	RENT	0.5775	0.4635	0.4155
	Fisher's score	0.5775	0.4635	0.4155
	Gini Score	0.5352	0.378	0.3522
	RFE	0.5352	0.3758	0.3495
	RFC	0.5493	0.386	0.375
ISOLET	None (Baseline)	**0.9532**	**0.953**	**0.9513**
	RENT	0.9513	0.951	0.9494
	Fisher's score	0.9359	0.9356	0.9333
	Gini Score	0.9442	0.944	0.942
	RFE	0.9442	0.9438	0.942
	RFC	0.9403	0.94	0.938



(a) Confusion matrix for the SD1 dataset with features selected by the RENT package.

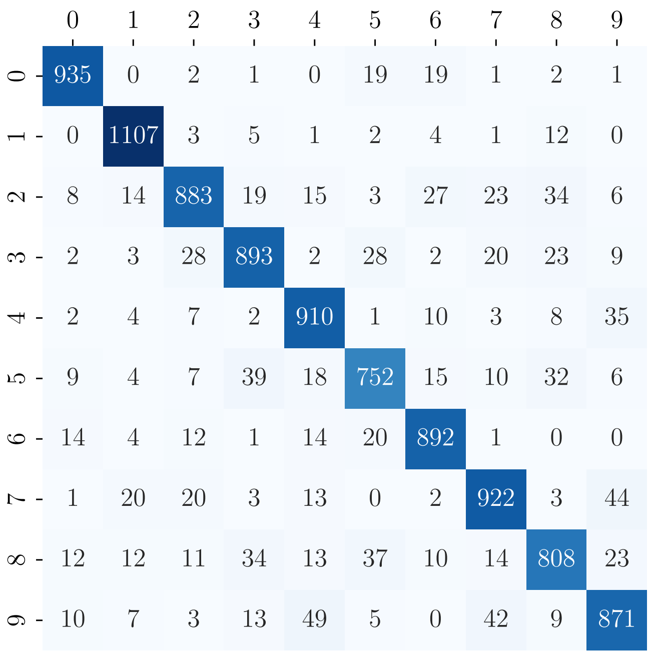


(b) Confusion matrix for the SD2 dataset with features selected by the RENT package.



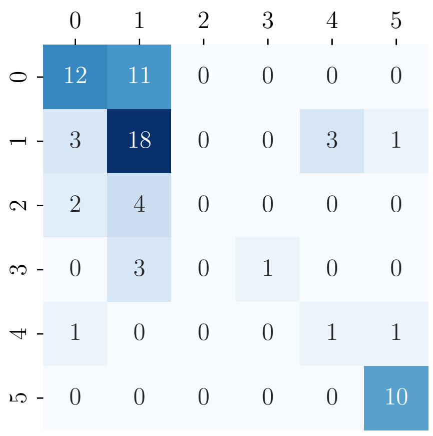
	0	1	2
0	41	10	9
1	4	44	9
2	9	2	37

(c) Confusion matrix for the SD3 dataset with features selected by the RENT package.



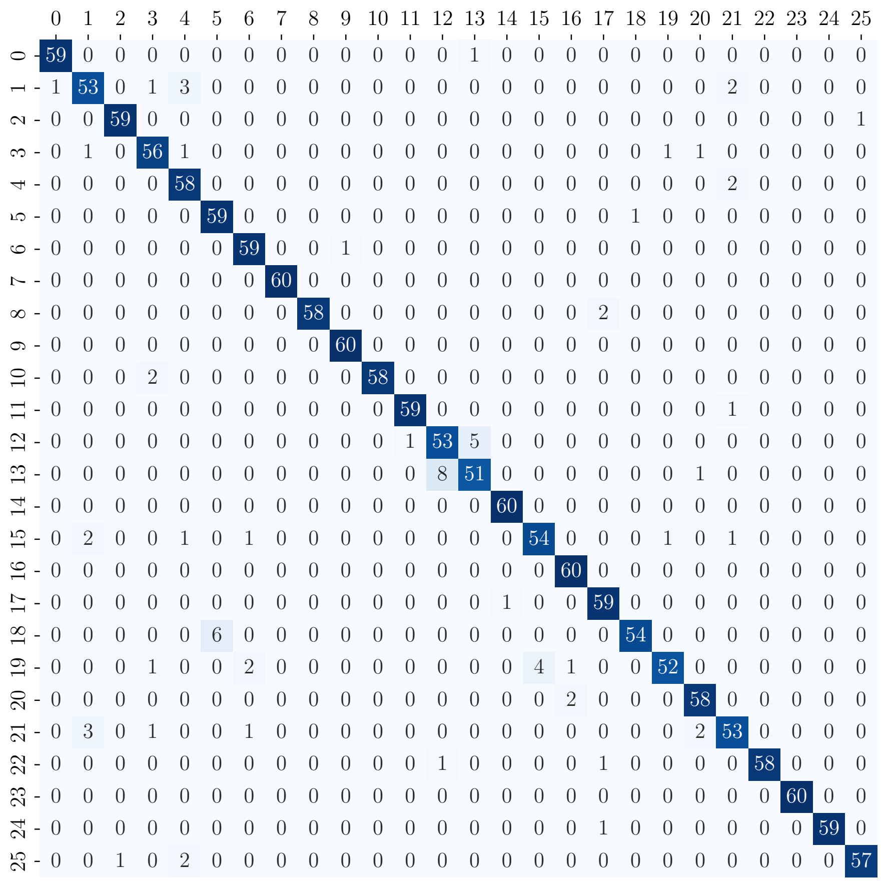
	0	1	2	3	4	5	6	7	8	9
0	935	0	2	1	0	19	19	1	2	1
1	0	1107	3	5	1	2	4	1	12	0
2	8	14	883	19	15	3	27	23	34	6
3	2	3	28	893	2	28	2	20	23	9
4	2	4	7	2	910	1	10	3	8	35
5	9	4	7	39	18	752	15	10	32	6
6	14	4	12	1	14	20	892	1	0	0
7	1	20	20	3	13	0	2	922	3	44
8	12	12	11	34	13	37	10	14	808	23
9	10	7	3	13	49	5	0	42	9	871

(d) Confusion matrix for the MNIST dataset with all features.



	0	1	2	3	4	5
0	12	11	0	0	0	0
1	3	18	0	0	3	1
2	2	4	0	0	0	0
3	0	3	0	1	0	0
4	1	0	0	0	1	1
5	0	0	0	0	0	10

(e) Confusion matrix for the glass dataset with all features.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	59	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	53	0	1	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
2	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
3	0	1	0	56	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
4	0	0	0	0	58	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
5	0	0	0	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
6	0	0	0	0	0	0	59	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	58	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	2	0	0	0	0	0	0	58	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	59	0	0	0	0	0	0	0	0	0	1	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	1	53	5	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	8	51	0	0	0	0	0	0	1	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0
15	0	2	0	0	1	0	1	0	0	0	0	0	0	0	0	54	0	0	0	1	0	1	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	59	0	0	0	0	0	0	0	0
18	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	54	0	0	0	0	0	0	0
19	0	0	0	1	0	0	2	0	0	0	0	0	0	0	0	4	1	0	0	52	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	58	0	0	0	0	0
21	0	3	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2	53	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	58	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	59	0
25	0	0	1	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	57

(f) Confusion matrix for the ISOLET dataset with all features.

Figure 6.1: Confusion matrix for all datasets trained on the feature subset with the highest performance. The columns indicate the actual class, and the rows indicate the predicted class.

6.5 Runtimes

Table 6.5 shows the runtimes of each feature selection technique on the datasets.

Table 6.5: Runtimes for feature selection techniques.

	Selection Runtime (seconds)					
Method	**SD1**	**SD2**	**SD3**	**MNIST**	**Glass**	**ISOLET**
RENT	115.28	173.36	277.08	98634.51	89.96	58779.5
Fisher's score	0.36	0.2	0.09	350.59	0	3.63
Gini score	0.06	0.12	0.19	18.55	0	3.87
RFE	0.1	0.6	4.1	50949.71	0.17	57.44
RFC	2.61	2.89	2.62	17.95	1.66	5.42

Table 6.6 shows the the training and inference runtimes of the logistic regression models on the selected features. The hyperparameters were tuned separately for each dataset. Features selected from the multiclass RENT technique were used for the benchmark.

Table 6.6: Final training and inference times on selected feature subset.

Dataset	**Training Runtime (sec)**	**Inference Runtime (ms)**
SD1	0.019	0.0681
SD2	0.02	0.0824
SD3	0.01	0.0572
MNIST	4.02	6.109
Glass	0.011	0.0474
ISOLET	2.78	0.9123

6.6 Selected Features

For the synthetic datasets SD1, SD2, and SD3, it is possible to evaluate the feature selection quality at a glance as shown in Table 6.7. As mentioned in the section 5.1 the first ten features of the datasets are informative, while the rest are either derived from the first ten or random noise.

The similarity of the features selected by each method can be calculated using the Jaccard index and plotted as shown in the figures 6.2 to 6.7.

Table 6.7: Indices of selected features. MNIST and ISOLET datasets were skipped here due to their much higher number of features. See Appendix A.

Dataset	**Feature Selector**	**Selected Features**
SD1	RENT	1, 2, 3, 4, 5, 7, 8, 10, 11
	Fisher's score	1, 2, 3, 4, 5, 7, 8, 10, 11
	Gini score	1, 2, 3, 4, 5, 8, 9, 10, 11
	RFE	1, 2, 3, 4, 5, 7, 8, 10, 11
	RFC	1, 2, 3, 4, 5, 7, 8, 10, 11
SD2	RENT	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 47
	Fisher's score	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 47
	Gini score	1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 22, 81
	RFE	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 47
	RFC	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 47
SD3	RENT	1, 2, 3, 4, 5, 10, 11
	Fisher's score	1, 2, 3, 4, 5, 10, 11
	Gini score	2, 3, 5, 10, 11, 200, 611
	RFE	2, 3, 4, 5, 10, 679, 752
	RFC	1, 2, 3, 4, 5, 10, 11
Glass	RENT	2, 3, 4, 8
	Fisher's score	2, 3, 4, 8
	Gini score	1, 3, 4, 8
	RFE	1, 2, 3, 5
	RFC	1, 3, 4, 7

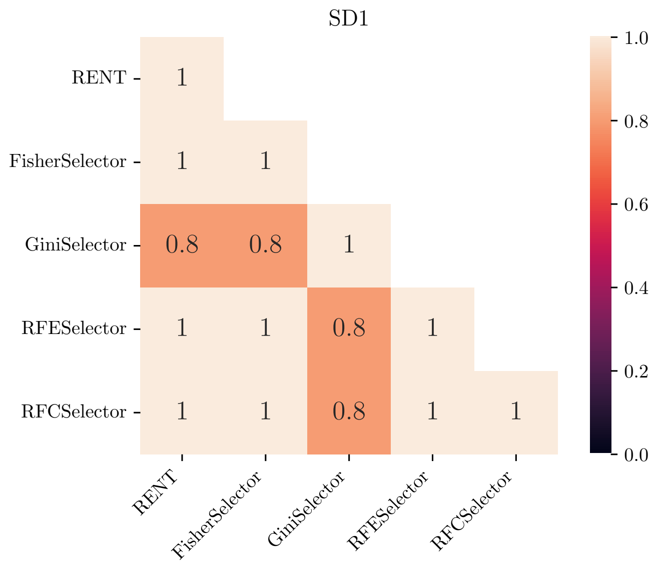


Figure 6.2: Jaccard index for features selected in the SD1 dataset.

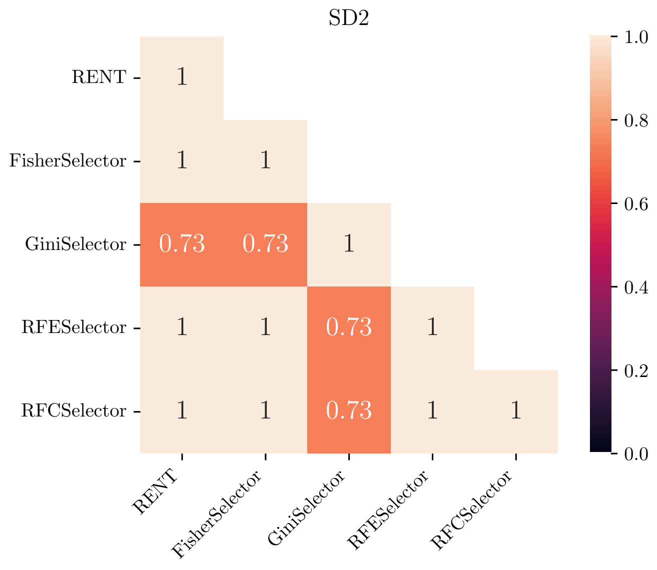


Figure 6.3: Jaccard index for features selected in the SD2 dataset.

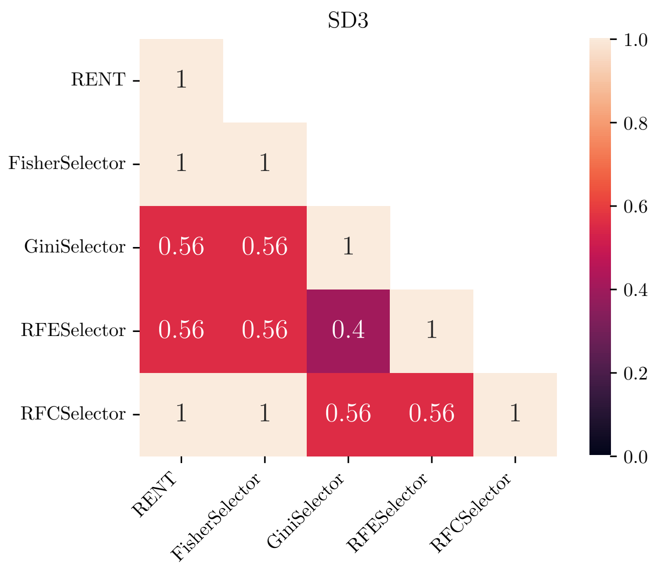


Figure 6.4: Jaccard index for features selected in the SD3 dataset.

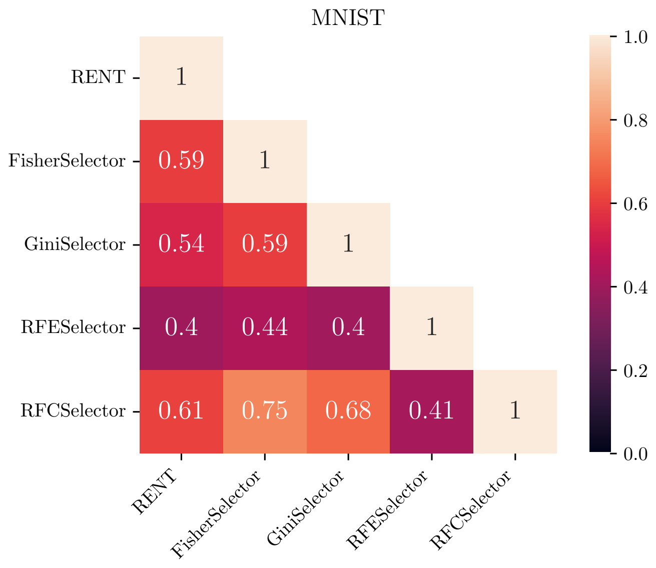


Figure 6.5: Jaccard index for features selected in the MNIST dataset.

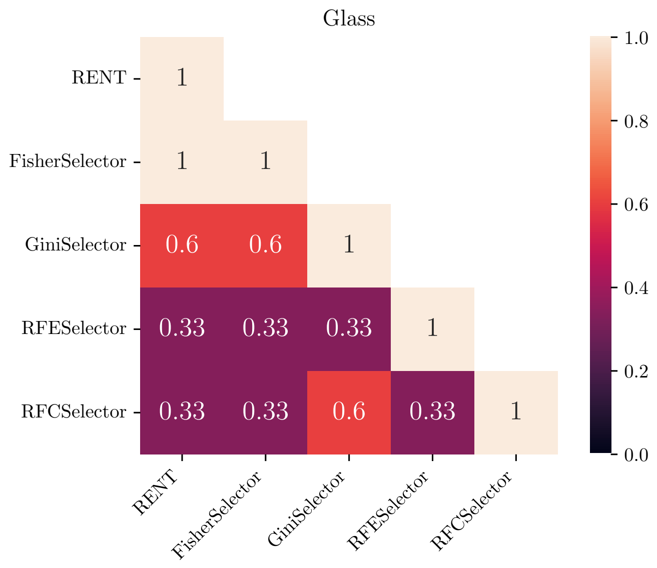


Figure 6.6: Jaccard index for features selected in the glass dataset.

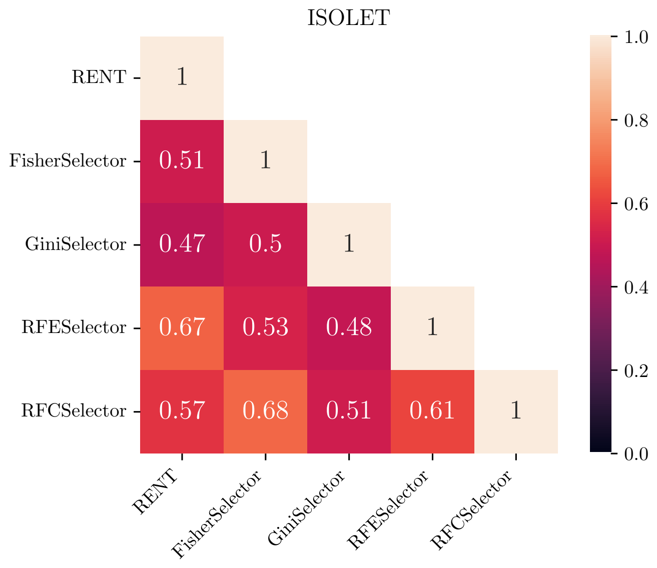


Figure 6.7: Jaccard index for features selected in the ISOLET dataset.

Chapter 7

Discussion

This thesis aimed to add support for multiclass classification to the RENT package and compare its performance with other feature selection techniques. The modified RENT package demonstrates competitive performance when compared to other widely used feature selection methods. This section discusses the performance gains of the RENT technique, the reduction in features, and the speedup obtained as a result compared to the baseline.

7.1 RENT Performance Gains

Comparing tables 6.4 and 6.1, with one exception, the feature selection techniques experienced a noticeable increase in performance on the synthetic dataset. Compared to the baseline, the Gini score method lost some performance on the SD1 dataset and there are performance losses across all metrics and feature selection techniques for the MNIST, glass and ISOLET datasets.

Table 7.1 compares the MCC scores of RENT with the baseline. A significant boost in performance can be observed for datasets SD2 and SD3. Among the three synthetic datasets, SD2 and SD3 are more complex in terms of the number of features and classes. The performance of multiclass RENT on the MNIST, glass, and ISOLET datasets shows a decline in MCC by 2.94%, 5.80%, and 0.20%, respectively. These datasets, however, still exhibit a significant reduction in features. While the MCC values for these datasets may not be optimal, the feature reduction can lead to a significant reduction in model training and inference times.

Table 7.1: Comparison of MCC and feature reduction percentages between the baseline and RENT methods.

Dataset	**Baseline MCC**	**RENT MCC**	**% Change in MCC**	**% Reduction in features**
SD1	0.7077	0.7124	0.66%	82%
SD2	0.2531	0.2757	8.20%	87%
SD3	0.5492	0.6570	16.41%	99.30%
MNIST	0.9059	0.8800	-2.94%	66.45%
Glass	0.4396	0.4155	-5.80%	60%
ISOLET	0.9513	0.9494	-0.20%	39.22%

This is explored further in Tables 7.2 and 7.3.

The low performance on the SD2 dataset is also apparent on the confusion matrix in Figure 6.1b. There are no true positives for class 5, and many of the samples in the test set have been labelled incorrectly as the class 2, 7, 8 or 9. The confusion matrix for the glass dataset in Figure 6.1e also paints an interesting picture. None of the samples for class 2 were predicted correctly, and classes 3 and 4 only had one correct prediction. Given the higher number of samples for the rest of the classes, this is a good indicator of a model being biased towards the majority class.

7.2 Comparisons to Other Methods

Evaluating the performance metrics of multiclass RENT compared to the remaining feature selection techniques from Table 6.4, RENT shares the best performance in three of the six datasets and ties with the rest.

RENT draws with Fisher's score, recursive feature elimination, and random forest classifier in the SD1 dataset. RENT also draws with Fisher's score and random forest classifier selectors for dataset SD3, while it ties to the Fisher's score selector in the glass dataset. It is worth noting that these datasets have relatively fewer classes and features than the rest, which could account for the observed outcomes.

RENT is the best-performing feature selection technique in datasets SD2, MNIST, and ISOLET. All three are datasets with a higher number of classes and features compared to the rest.

Looking at Table 6.5, RENT is shown to take significantly longer to select features compared to the rest of the techniques. This can be attributed to the larger number of model that RENT trains. For each regularization strength and L1 ratio provided as hyperparameters, K models are trained. For each dataset, the number of trained models is multiplied by the number of classes due to the implementation of the one-vs-rest technique.

It is worth noting that the feature selection process is a one-time investment and depending on the application a longer computation may not be critical. The number of models trained in RENT can be controlled through the elastic net search space and the K hyperparameter. While the performance difference between the features selected by RENT and the other techniques might not be significant, the three thresholds t_1, t_2, and t_3 could be fine-tuned to achieve to get higher performance.

7.3 Runtimes Reductions

Table 7.2 shows a notable decrease in the model training times across all datasets as a result of the feature selection process. The datasets SD1 and glass exhibit the most modest runtime gains, which can be attributed to their relatively small number of features and samples compared to the other datasets. The RENT and ISOLET datasets see a 66.45% and 39.22% reduction in runtime respectively.

Table 7.2: Comparison of training runtimes between the full feature set and selected feature subset.

Dataset	**Baseline training runtime (sec)**	**Final training runtime (sec)**	**% reduction**
SD1	0.021	0.019	9.52%
SD2	0.633	0.02	96.84%
SD3	0.22	0.01	95.45%
MNIST	9.24	4.02	56.49%
Glass	0.015	0.011	26.67%
ISOLET	4.25	2.78	34.59%

Table 7.3 shows the decrease in the model inference times across all datasets. All datasets except glass show a modest reduction in inference time. The low reduction for the glass dataset could be attributed to its already minuscule dataset size compared to the rest.

Table 7.3: Comparison of inference runtimes between the full feature set and selected feature subset.

Dataset	**Baseline inference runtime (ms)**	**Final inference runtime (ms)**	**% reduction**
SD1	0.105	0.0681	35.14%
SD2	0.2499	0.0824	67.03%
SD3	0.2544	0.0572	77.52%
MNIST	14.6104	6.109	58.19%
Glass	0.0476	0.0474	0.42%
ISOLET	1.3833	0.9123	34.05%

This highlights another advantage of using feature selection techniques - a reduction in runtime on real-world datasets with similar performance to the full feature set. A reduction in features also comes with a potential increase in interpretability. This serves as a compelling reason to use RENT in scenarios where costs are a concern. Applications that require frequent model re-training and serve a high volume inference traffic would benefit from feature selection. One would need to take into account *data drifting*, which would require re-running the feature selection process at regular intervals.

7.4 Selected Features

Table 6.7 portrays the features selected by each method examined in this study for four datasets. Analyzing dataset SD1, the sixth feature is not considered important by any of the methods despite its inclusion in the informative feature set. Furthermore, the ninth feature is excluded by four out of the five methods, with the Gini score method being the sole exception. The Jaccard index plot in Figure 6.2 also shows how the Gini score is the only method that has a different set of selected features. Interestingly, all methods selected the eleventh feature even though it is a redundant feature derived from the ten previous informative features. One could speculate that the eleventh feature encapsulates information from the sixth and ninth features, giving it higher importance than the individual features. This is investigated by calculating the correlation between the informative and derived features as shown in Figure 7.1.

It can be observed that both features eleven and twelve are highly correlated with most of the informative features. However, there are no distinct relationships with

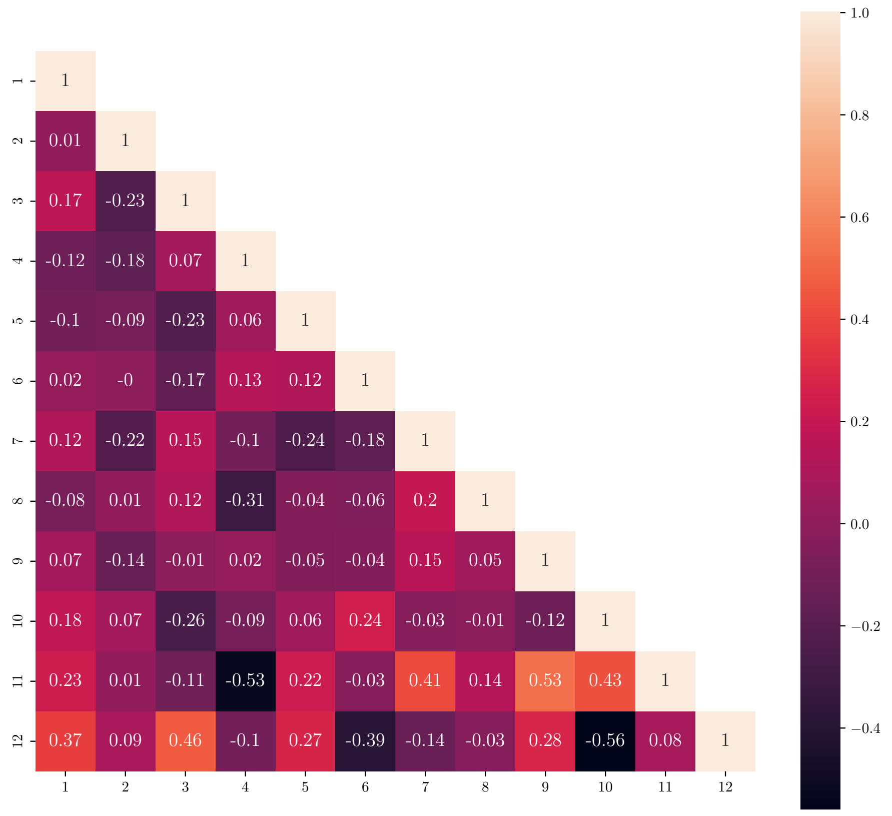


Figure 7.1: Correlation plot of the informative and derived features from the SD1 dataset.

the sixth and ninth feature that could account for their absence from the selected features list. If a high correlation was the factor the absence then feature ten would be a prime candidate due to its high correlation with both features eleven and twelve. However, that is not the case, hence the correlation test is inconclusive in explaining the selected features.

In the case of dataset SD2, four of the five methods accurately selected all ten informative features, with the Gini method failing to select the eighth feature. The Gini score method, again, is the only method with a different set of features selected as evidenced by the Jaccard index plot in Figure 6.3. Notably, all methods also selected the two derived features, eleven and twelve. All methods also selected features containing random noise. Removing those features could improve system performance.

For the SD3 dataset, RENT, Fisher's score, and random forest classifier techniques selected the same set of features as seen in Figure 6.4. Informative features six to nine were not selected by any method. Four out of the five methods did however select the derived feature eleven. The Gini and recursive feature elimination methods each selected two random noise features. The number of features to be returned by those techniques were set to the same number of features selected by RENT. Fine-tuning this number could have prevented the selection of the noise. In real-world applications, noise in the data could go unnoticed when proper analysis (like the correlation test) is not done. One would also not be aware of the proper number of features to be selected. In such circumstances, RENT offers an advantage as it does not require user-defined feature counts.

The Jaccard index shows that, for the MNIST and ISOLET datasets, none of the methods agreed on the feature sets to select, as seen in Figures 6.5 and 6.7. These two are also the only datasets where the RENT method disagrees with the Fisher's score method on the selected feature subsets. For the MNIST dataset, the RENT selected features are more similar to the features selected by the random forest classifier method, and for the ISOLET dataset, the feature are similar to the ones from recursive feature elimination. For the glass dataset, both RENT and Fisher's score methods agree on the selected feature subset. All other techniques selected a unique subset and showed a decline in their final performance, as seen in Table 6.4.

The difference in performance from the selected feature subsets across all datasets and techniques were not significantly different and a low Jaccard index for the MNIST, glass and ISOLET datasets indicate that the features might be correlated.

The selected features for the MNIST dataset have been visualized in the form of an image in Figure 7.2. The 784-dimensional feature set is reshaped back to a 28x28 image. The white pixels indicate the selected features.

Considering that the MNIST dataset consists of images of handwritten digits, the features selected by the Fisher and random forest classifier methods appear to describe the region where the digit is likely to be present, towards the center. The overall circular shape is suggestive of the number zero. In contrast, features selected by the RENT, Gini score, and recursive feature elimination methods are spread throughout the image. This is further investigated by studying regions in MNIST data where the digits are present.

Figure 7.3 is an image created by coalescing all MNIST training data. As expected, it reveals a high concentration of pixels around the center of the image, signifying that most digits have pixels clustered around this region. This image aligns with the features selected by the Fisher and random forest classifier methods. To

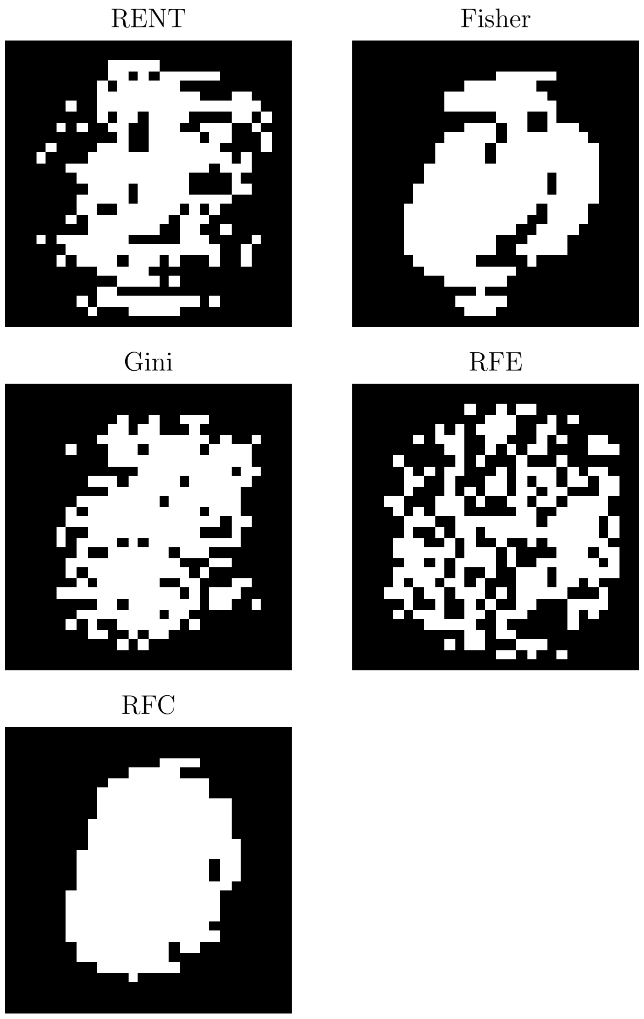


Figure 7.2: MNIST features selected by all techniques visualized as 28x28 pixel images.

understand the features selected by the remaining methods, one can examine the binarized version of this image.

In Figure 7.4, the pixels from Figure 7.3 are set to a value of one if their original value is greater than zero. This shows that, across all the training data in the MNIST dataset, the digits fall under the yellow region, which is much larger than the region visible in Figure 7.3. The RENT, Gini score, and recursive feature elimination methods selected the features along the edges. Since RENT exhibits a higher test performance than Fisher's score or random forest classifier methods, this could indicate information in the outer regions that are relevant for classification, overlooked by the two techniques. This can be investigated further

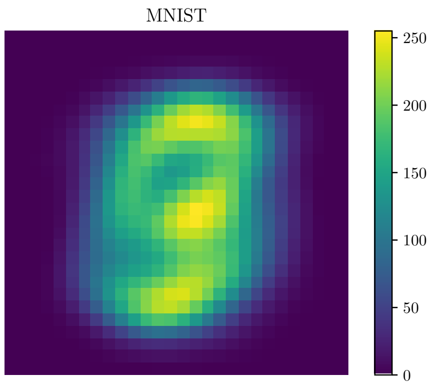


Figure 7.3: Composite image of all MNIST training data.

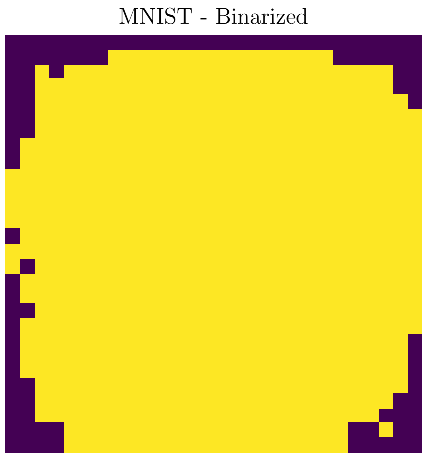


Figure 7.4: Binarized composite image of all MNIST training data.

by subtracting the feature map from the RENT method from each of the other methods yielding results shown in Figure 7.5.

Figure 7.5 visualizes the features that were selected by RENT but not by the other four techniques. It was obtained through a set difference of the selected features by RENT with the other methods. With the recursive feature elimination, as was apparent from Figure 7.2, there are regions around the center of the image that it fails to capture. And compared to the rest of the techniques, RENT captures regions that extend out further from the center, aligning with our assumptions.

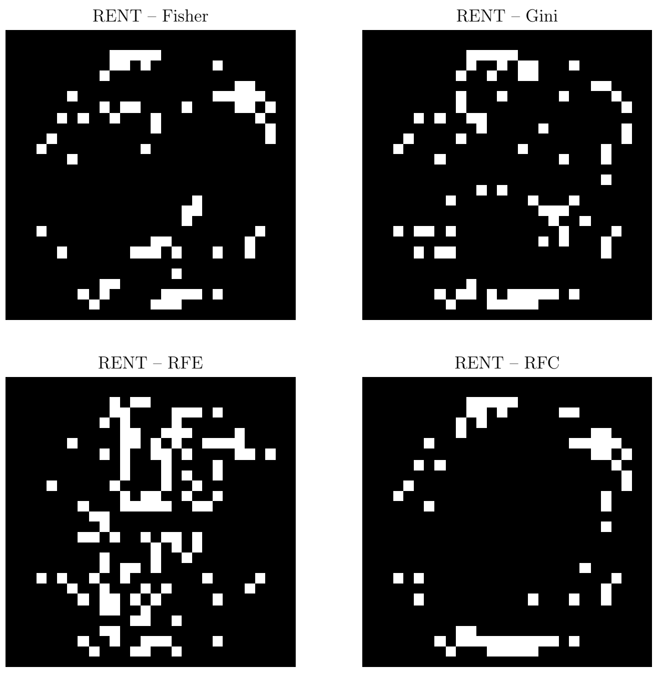


Figure 7.5: Comparison of features selected by RENT with other methods.

Chapter 8

Conclusion

This study successfully modified the RENT algorithm to support multiclass classification problems, expanding its applications to more diverse and complex scenarios. The modifications were rigorously tested on a variety of datasets and benchmarked against several established feature selection techniques providing valuable insights into its performance and potential areas of improvement.

While RENT demonstrated a higher performance in certain datasets, particularly those featuring a relatively high number of classes, it only sometimes outperformed other feature selection techniques or the baseline across all datasets. Regardless, even in cases where RENT's performance was not clearly superior, the algorithm significantly reduced the number of features and the overall runtime. This study demonstrated its potential as a valuable tool in the feature selection domain.

The insights gained from this study contribute to our understanding of the RENT algorithm's strength and limitations and serve as a platform for future research to enhance its performance and applications across a broader spectrum of real-world classification problems.

Chapter 9

Future Work

In light of the findings from this study, there are some areas for future research. Our analysis revealed that RENT performs well, particularly on datasets with a high number of classes. It would be beneficial to explore further the underlying factors contributing to this phenomenon and determine potential improvements to the RENT algorithm that capitalize on its strengths. Also, the current RENT implementation requires the user to select the three t thresholds manually. Future work could automate this process by exhaustively examining all possible values for each t threshold and selecting the optimal values based on the model performance. This would streamline the application of RENT and also enable more efficient and accurate feature selection, further supporting its application in real-world scenarios. Nested k-fold cross-validation is another avenue of research. Running the feature selection process multiple times on different folds of the data increases the time and cost complexity, but it would give us another indicator of the stability of the selected features and model performance.

Bibliography

[1] A. Jenul, S. Schrunner, K. H. Liland, U. G. Indahl, C. M. Futsæther and O. Tomic, 'Rent—repeated elastic net technique for feature selection,' *IEEE Access*, vol. 9, pp. 152 333–152 346, 2021.

[2] Q. Gu, Z. Li and J. Han, 'Generalized fisher score for feature selection,' *arXiv preprint arXiv:1202.3725*, 2012.

[3] J. Li, K. Cheng, S. Wang *et al.*, 'Feature selection: A data perspective,' *ACM computing surveys (CSUR)*, vol. 50, no. 6, pp. 1–45, 2017.

[4] X. Zeng, Y.-W. Chen and C. Tao, 'Feature selection using recursive feature elimination for handwritten digit recognition,' in *2009 Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, IEEE, 2009, pp. 1205–1208.

[5] L. Breiman, 'Random forests,' *Machine learning*, vol. 45, pp. 5–32, 2001.

[6] S. Raschka, V. Mirjalili and a. O. M. C. Safari, *Python Machine Learning - Third Edition*. Packt Publishing, 2019. [Online]. Available: https://books.google.no/books?id=SYs-zQEACAAJ.

[7] G. Thimm and E. Fiesler, 'High-order and multilayer perceptron initialization,' *IEEE Transactions on Neural Networks*, vol. 8, no. 2, pp. 349–359, 1997.

[8] F. C. Rodrigues, M. Espadoto, R. Hirata Jr and A. C. Telea, 'Constructing and visualizing high-quality classifier decision boundary maps,' *Information*, vol. 10, no. 9, p. 280, 2019.

[9] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4.

[10] S. Desmedt, *The math behind neural networks: Part 2 - the adaline perceptron*, Aug. 2020. [Online]. Available: https://www.codeproject.com/Articles/5276948/The-Math-behind-Neural-Networks-Part-2-The-ADALINE#sum-of-squared-errors (visited on 20/04/2023).

[11] T. Hastie, R. Tibshirani, J. H. Friedman and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009, vol. 2.

[12] M. Durmus, *A Primer to the 42 Most commonly used Machine Learning Algorithms (With Code Samples)*. Murat Durmus, 2023. [Online]. Available: https://books.google.no/books?id=VemrEAAAQBAJ.

[13] M. D. Zeiler, 'Adadelta: An adaptive learning rate method,' *arXiv preprint arXiv:1212.5701*, 2012.

[14] A. Kumar, *Model complexity & overfitting in machine learning*, May 2022. [Online]. Available: https://vitalflux.com/model-complexity-overfitting-in-machine-learning/ (visited on 25/04/2023).

[15] C. Stephenson, S. Padhy, A. Ganesh, Y. Hui, H. Tang and S. Chung, 'On the geometry of generalization and memorization in deep neural networks,' *arXiv preprint arXiv:2105.14602*, 2021.

[16] M. Schmidt, 'Least squares optimization with l1-norm regularization,' *CS542B Project Report*, vol. 504, pp. 195–221, 2005.

[17] T. Hastie, R. Tibshirani and M. Wainwright, *Statistical learning with sparsity: the lasso and generalizations*. CRC press, 2015.

[18] H. Zou and T. Hastie, 'Regularization and variable selection via the elastic net,' *Journal of the royal statistical society: series B (statistical methodology)*, vol. 67, no. 2, pp. 301–320, 2005.

[19] I. Guyon and A. Elisseeff, 'An introduction to variable and feature selection,' *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[20] Y. Saeys, I. Inza and P. Larranaga, 'A review of feature selection techniques in bioinformatics,' *bioinformatics*, vol. 23, no. 19, pp. 2507–2517, 2007.

[21] H. Motoda and H. Liu, 'Feature selection, extraction and construction,' *Communication of IICM (Institute of Information and Computing Machinery, Taiwan)*, vol. 5, no. 67-72, p. 2, 2002.

[22] S. Khalid, T. Khalil and S. Nasreen, 'A survey of feature selection and feature extraction techniques in machine learning,' in *2014 science and information conference*, IEEE, 2014, pp. 372–378.

[23] Z. M. Hira and D. F. Gillies, 'A review of feature selection and feature extraction methods applied on microarray data,' *Advances in bioinformatics*, vol. 2015, 2015.

[24] E. A. K. Zaman, A. Mohamed and A. Ahmad, 'Feature selection for online streaming high-dimensional data: A state-of-the-art review,' *Applied Soft Computing*, p. 109 355, 2022.

[25] K. Torkkola, ‘Feature extraction by non-parametric mutual information maximization,’ *Journal of machine learning research*, vol. 3, no. Mar, pp. 1415–1438, 2003.

[26] G. Manikandan and S. Abirami, ‘Feature selection is important: State-of-the-art methods and application domains of feature selection on high-dimensional data,’ *Applications in Ubiquitous Computing*, pp. 177–196, 2021.

[27] M. I. Prasetiyowati, N. U. Maulidevi and K. Surendro, ‘Determining threshold value on information gain feature selection to increase speed and prediction accuracy of random forest,’ *Journal of Big Data*, vol. 8, no. 1, p. 84, 2021.

[28] C. Khammassi and S. Krichen, ‘A ga-lr wrapper approach for feature selection in network intrusion detection,’ *computers & security*, vol. 70, pp. 255–277, 2017.

[29] R. Kohavi and G. H. John, ‘Wrappers for feature subset selection,’ *Artificial intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[30] S. Ma and J. Huang, ‘Penalized feature selection and classification in bioinformatics,’ *Briefings in bioinformatics*, vol. 9, no. 5, pp. 392–403, 2008.

[31] W. A. Zgallai, *Biomedical signal processing and artificial intelligence in healthcare*. Academic Press, 2020.

[32] C. Li and B. Wang, ‘Fisher linear discriminant analysis,’ *CCIS Northeastern University*, p. 6, 2014.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort *et al.*, ‘Scikit-learn: Machine learning in Python,’ *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[34] A. Jenul, S. Schrunner, B. N. Huynh and O. Tomic, ‘Rent: A python package for repeated elastic net feature selection,’ *Journal of Open Source Software*, vol. 6, no. 63, p. 3323, 2021.

[35] G. Zeng, ‘On the analytical properties of category encodings in logistic regression,’ *Communications in Statistics-Theory and Methods*, vol. 52, no. 6, pp. 1870–1887, 2023.

[36] R. Bellman, ‘Dynamic programming, princeton univ,’ *Press Princeton, New Jersey*, 1957.

[37] S. Misra, H. Li and J. He, *Machine learning for subsurface characterization*. Gulf Professional Publishing, 2019.

[38] A. Géron, ‘Hands-on machine learning with scikit-learn and tensorflow: Concepts,’ *Tools, and Techniques to build intelligent systems*, 2017.

[39] D. Zhang, J. Wang and X. Zhao, 'Estimating the uncertainty of average f1 scores,' in *Proceedings of the 2015 International conference on the theory of information retrieval*, 2015, pp. 317–320.

[40] D. Chicco and G. Jurman, 'The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation,' *BMC genomics*, vol. 21, pp. 1–13, 2020.

[41] S. Raschka, 'Model evaluation, model selection, and algorithm selection in machine learning,' *arXiv preprint arXiv:1811.12808*, 2018.

[42] A. Dandekar, R. A. Zen and S. Bressan, 'A comparative study of synthetic dataset generation techniques,' in *Database and Expert Systems Applications: 29th International Conference, DEXA 2018, Regensburg, Germany, September 3–6, 2018, Proceedings, Part II 29*, Springer, 2018, pp. 387–395.

[43] *Sklearn.datasets.make_classification — scikit-learn 1.2.2 documentation.* [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make%5C_classification.html (visited on 20/04/2023).

[44] J. Snoek, H. Larochelle and R. P. Adams, 'Practical bayesian optimization of machine learning algorithms,' *Advances in neural information processing systems*, vol. 25, 2012.

[45] J. Bergstra and Y. Bengio, 'Random search for hyper-parameter optimization.,' *Journal of machine learning research*, vol. 13, no. 2, 2012.

[46] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams and N. De Freitas, 'Taking the human out of the loop: A review of bayesian optimization,' *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[47] *Timeit — measure execution time of small code snippets — python 3.11.3 documentation.* [Online]. Available: https://docs.python.org/3/library/timeit.html (visited on 20/04/2023).

[48] C.-M. Hwang, M.-S. Yang and W.-L. Hung, 'New similarity measures of intuitionistic fuzzy sets based on the jaccard index with its application to clustering,' *International Journal of Intelligent Systems*, vol. 33, no. 8, pp. 1672–1688, 2018.

[49] S. Lee, 'Improving jaccard index for measuring similarity in collaborative filtering,' in *Information Science and Applications 2017: ICISA 2017 8*, Springer, 2017, pp. 799–806.

[50] L. Deng, 'The mnist database of handwritten digit images for machine learning research [best of the web],' *IEEE signal processing magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[51] I. W. Evett and E. J. Spiehler, 'Rule induction in forensic science,' in *Knowledge Based Systems*, 1989, pp. 152–160.

[52] M. Fanty and R. Cole, 'Spoken letter recognition,' *Advances in neural information processing systems*, vol. 3, 1990.

Appendix A

Features Selected for the MNIST and ISOLET datasets

A.1 MNIST

A.1.1 MNIST with RENT

67, 68, 69, 70, 71, 95, 96, 98, 100, 101, 102, 103, 104, 105, 122, 124, 125, 126, 127, 128, 129, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 163, 164, 175, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 206, 208, 209, 211, 212, 213, 214, 219, 220, 222, 230, 232, 235, 236, 239, 240, 241, 244, 245, 246, 249, 264, 267, 268, 269, 270, 271, 273, 274, 278, 285, 290, 291, 292, 295, 296, 297, 298, 299, 300, 301, 306, 312, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 332, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 358, 359, 360, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 387, 401, 402, 403, 404, 406, 407, 408, 409, 410, 415, 416, 428, 429, 430, 431, 432, 434, 435, 436, 437, 438, 439, 440, 441, 442, 456, 457, 460, 461, 462, 463, 464, 465, 467, 469, 483, 485, 486, 487, 488, 489, 490, 491, 492, 494, 495, 496, 513, 514, 515, 516, 517, 518, 519, 522, 523, 526, 536, 538, 539, 540, 541, 542, 543, 544, 550, 551, 552, 557, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 584, 594, 596, 597, 598, 599, 601, 602, 603, 605, 609, 612, 626, 627, 628, 629, 630, 656, 657, 658, 659, 661, 682, 683, 708, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 721, 737, 741, 742, 743, 744, 745

A.1.2 MNIST with Fisher's score

99, 100, 101, 102, 103, 104, 124, 125, 126, 127, 128, 129, 130, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 211, 212, 213, 216, 236, 237, 238, 240, 241, 244, 245, 246, 262, 263, 264, 265, 266, 268, 269, 270, 271, 272, 273, 274, 275, 289, 290, 291, 292, 293, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 317, 318, 319, 320, 321, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 385, 386, 387, 388, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 413, 414, 415, 416, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 468, 469, 470, 471, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 496, 497, 498, 499, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 523, 524, 525, 526, 538, 539, 540, 541, 542, 543, 544, 545, 546, 550, 551, 552, 553, 566, 567, 568, 569, 570, 571, 572, 573, 574, 577, 578, 579, 580, 581, 595, 596, 597, 598, 599, 600, 606, 624, 625, 626, 627, 628, 629, 630, 631, 632, 654, 655, 656, 657, 658, 659, 685, 711, 712, 713, 714, 715, 740, 741, 742

A.1.3 MNIST with Gini score

96, 99, 102, 103, 104, 123, 124, 126, 127, 131, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 162, 165, 175, 179, 180, 181, 183, 184, 185, 186, 187, 189, 190, 191, 192, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 238, 239, 240, 241, 242, 243, 244, 245, 246, 248, 249, 261, 262, 263, 265, 266, 267, 268, 269, 271, 272, 273, 274, 275, 276, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 316, 317, 318, 319, 320, 321, 322, 323, 325, 326, 327, 328, 329, 330, 331, 343, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 357, 358, 359, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 387, 388, 398, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 415, 426, 428, 429, 430, 431, 433, 435, 436, 437, 438, 439, 440, 441, 442, 443, 456, 459, 460, 461, 462, 463, 464, 466, 467, 468, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 497, 498, 511, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 524, 540, 542, 543, 544, 545, 546, 547, 550, 551, 555, 556, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 579, 581, 582, 583, 598, 599, 601, 602, 603, 604, 605, 606, 607, 609, 610, 613, 624, 625, 626, 627, 628, 629, 630, 631, 633, 634, 651, 653, 656, 657, 658, 659, 660, 661, 663, 681, 682, 685, 687, 688, 712, 714

A.1.4 MNIST with RFE

68, 71, 73, 74, 98, 99, 100, 104, 106, 121, 123, 125, 126, 127, 128, 131, 132, 147, 149, 150, 151, 154, 155, 159, 161, 164, 165, 178, 179, 182, 184, 186, 187, 192, 193, 194, 201, 207, 209, 213, 217, 230, 232, 234, 235, 239, 241, 242, 243, 244, 246, 248, 249, 258, 259, 262, 266, 267, 269, 271, 272, 274, 275, 277, 278, 286, 290, 292, 294, 295, 297, 298, 300, 301, 305, 306, 312, 313, 317, 318, 319, 321, 324, 325, 327, 328, 330, 331, 332, 333, 341, 343, 345, 346, 347, 353, 354, 357, 358, 359, 360, 361, 370, 371, 372, 375, 376, 377, 378, 379, 380, 381, 382, 384, 385, 386, 387, 388, 390, 398, 401, 403, 404, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 418, 425, 426, 427, 430, 432, 435, 438, 440, 441, 442, 443, 444, 445, 453, 454, 455, 456, 457, 458, 460, 461, 462, 464, 469, 470, 471, 473, 474, 482, 483, 485, 487, 488, 489, 490, 492, 495, 496, 497, 498, 499, 500, 502, 509, 510, 511, 512, 513, 515, 518, 521, 522, 523, 525, 526, 527, 528, 529, 530, 537, 539, 540, 542, 543, 545, 549, 550, 551, 553, 554, 556, 564, 565, 566, 568, 569, 571, 572, 573, 575, 577, 578, 579, 580, 581, 585, 594, 597, 600, 602, 605, 606, 610, 611, 612, 613, 621, 623, 624, 625, 628, 629, 631, 632, 633, 638, 640, 652, 653, 654, 655, 656, 659, 666, 684, 685, 687, 688, 710, 712, 713, 717, 718, 719, 743, 744, 746, 749

A.1.5 MNIST with RFC

100, 101, 102, 103, 125, 126, 127, 128, 129, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 386, 387, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 414, 415, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 596, 597, 598, 599, 600, 601, 602, 603, 604, 606, 607, 624, 625, 626, 627, 628, 629, 630, 631, 632, 654, 655, 656, 657, 658, 659, 660, 661, 685

A.2 ISOLET

A.2.1 ISOLET with RENT

1, 2, 3, 4, 5, 6, 9, 10, 12, 14, 15, 16, 17, 18, 19, 21, 34, 38, 45, 46, 48, 51, 53, 61, 65, 66, 67, 68, 69, 70, 73, 76, 78, 82, 84, 85, 98, 99, 100, 101, 102, 104, 105, 106, 107, 115, 116, 118, 129, 130, 131, 132, 133, 134, 136, 138, 139, 142, 143, 144, 145, 146, 147, 151, 162, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 177, 178, 179, 180, 181, 182, 183, 184, 186, 187, 189, 191, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 208, 209, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 237, 238, 239, 240, 241, 242, 243, 244, 249, 250, 252, 253, 255, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 273, 275, 276, 277, 288, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 308, 309, 310, 312, 320, 321, 322, 324, 325, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 342, 343, 344, 352, 360, 361, 362, 363, 364, 367, 368, 372, 373, 376, 381, 383, 384, 385, 387, 388, 389, 390, 391, 394, 395, 396, 397, 398, 399, 400, 408, 411, 412, 413, 414, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 457, 458, 461, 462, 463, 471, 472, 473, 474, 475, 476, 477, 479, 480, 481, 482, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 504, 505, 508, 510, 511, 512, 513, 514, 515, 519, 520, 521, 522, 524, 525, 526, 527, 528, 529, 530, 531, 533, 537, 538, 539, 540, 541, 545, 546, 548, 549, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 562, 565, 566, 567, 569, 572, 574, 576, 577, 578, 579, 581, 582, 583, 584, 585, 587, 588, 590, 591, 592, 595, 596, 597, 598, 599, 600, 601, 602, 604, 606, 607, 608, 609, 610, 611, 612, 614, 616

A.2.2 ISOLET with Fisher's score

2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 34, 35, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 66, 67, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 87, 88, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 117, 118, 119, 120, 121, 122, 123, 124, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 162, 163, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 226, 227, 229, 230, 231, 232, 233, 234, 235, 236, 258, 259, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 276, 290, 291, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 309, 310, 311, 312, 313, 332, 333, 356, 357, 358, 359, 360, 361, 362, 363, 371, 372, 373, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 433, 436, 440, 441, 442, 443, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 466, 467, 468, 469, 470, 471, 472, 474, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 505, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 579, 581, 584, 585, 586, 587, 588, 589, 590, 591, 592, 596, 610, 614, 615, 616, 617

A.2.3 ISOLET with Gini score

2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 23, 24, 25, 29, 33, 34, 35, 36, 39, 40, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 55, 60, 61, 62, 63, 64, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 88, 91, 95, 97, 98, 100, 101, 102, 103, 104, 105, 106, 107, 109, 110, 111, 112, 115, 116, 117, 118, 119, 120, 122, 124, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 150, 154, 155, 157, 160, 164, 166, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 185, 186, 187, 188, 189, 190, 193, 195, 198, 199, 200, 201, 203, 204, 205, 207, 209, 212, 213, 214, 216, 217, 220, 221, 222, 225, 228, 229, 231, 234, 237, 238, 239, 241, 242, 243, 244, 246, 247, 248, 252, 253, 254, 255, 257, 260, 263, 264, 265, 268, 271, 272, 275, 276, 277, 278, 285, 286, 290, 293, 294, 297, 298, 306, 308, 310, 311, 312, 315, 319, 320, 321, 323, 326, 329, 330, 331, 332, 333, 334, 335, 337, 339, 340, 345, 347, 348, 350, 352, 357, 360, 361, 362, 363, 364, 365, 367, 372, 374, 376, 377, 380, 383, 385, 386, 387, 389, 390, 392, 393, 394, 395, 396, 397, 398, 399, 400, 402, 404, 407, 410, 411, 412, 416, 417, 418, 422, 424, 425, 426, 427, 431, 432, 433, 434, 436, 438, 439, 442, 443, 449, 455, 456, 457, 458, 460, 461, 464, 465, 466, 467, 469, 470, 472, 474, 475, 476, 479, 480, 481, 484, 485, 486, 487, 488, 489, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 502, 503, 504, 505, 506, 510, 512, 515, 516, 518, 519, 520, 522, 523, 527, 528, 529, 534, 535, 537, 538, 540, 542, 543, 545, 547, 548, 549, 550, 552, 553, 554, 555, 557, 558, 560, 561, 563, 564, 565, 566, 568, 571, 572, 574, 575, 577, 581, 582, 584, 586, 587, 588, 589, 591, 593, 594, 596, 599, 603, 607, 610, 611, 612, 613, 616, 617

A.2.4 ISOLET with RFE

2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 23, 24, 25, 29, 33, 34, 35, 36, 39, 40, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 55, 60, 61, 62, 63, 64, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 88, 91, 95, 97, 98, 100, 101, 102, 103, 104, 105, 106, 107, 109, 110, 111, 112, 115, 116, 117, 118, 119, 120, 122, 124, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 150, 154, 155, 157, 160, 164, 166, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 185, 186, 187, 188, 189, 190, 193, 195, 198, 199, 200, 201, 203, 204, 205, 207, 209, 212, 213, 214, 216, 217, 220, 221, 222, 225, 228, 229, 231, 234, 237, 238, 239, 241, 242, 243, 244, 246, 247, 248, 252, 253, 254, 255, 257, 260, 263, 264, 265, 268, 271, 272, 275, 276, 277, 278, 285, 286, 290, 293, 294, 297, 298, 306, 308, 310, 311, 312, 315, 319, 320, 321, 323, 326, 329, 330, 331, 332, 333, 334, 335, 337, 339, 340, 345, 347, 348, 350, 352, 357, 360, 361, 362, 363, 364, 365, 367, 372, 374, 376, 377, 380, 383, 385, 386, 387, 389, 390, 392, 393, 394, 395, 396, 397, 398, 399, 400, 402, 404, 407, 410, 411, 412, 416, 417, 418, 422, 424, 425, 426, 427, 431, 432, 433, 434, 436, 438, 439, 442, 443, 449, 455, 456, 457, 458, 460, 461, 464, 465, 466, 467, 469, 470, 472, 474, 475, 476, 479, 480, 481, 484, 485, 486, 487, 488, 489, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 502, 503, 504, 505, 506, 510, 512, 515, 516, 518, 519, 520, 522, 523, 527, 528, 529, 534, 535, 537, 538, 540, 542, 543, 545, 547, 548, 549, 550, 552, 553, 554, 555, 557, 558, 560, 561, 563, 564, 565, 566, 568, 571, 572, 574, 575, 577, 581, 582, 584, 586, 587, 588, 589, 591, 593, 594, 596, 599, 603, 607, 610, 611, 612, 613, 616, 617

A.2.5 ISOLET with RFC

3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 227, 228, 232, 233, 234, 235, 239, 240, 241, 259, 260, 262, 264, 265, 266, 267, 268, 271, 272, 273, 291, 292, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 343, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 368, 371, 372, 373, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 424, 425, 426, 427, 428, 429, 430, 433, 434, 436, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 472, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 503, 505, 506, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 566, 567, 568, 569, 570, 571, 572, 574, 575, 577, 581, 582, 584, 585, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599

Appendix B

Class distribution of datasets

This section shows the class distribution of all the datasets used in this study.

Table B.1: SD1 dataset class distribution.

Class	Percentage
1	33.35%
2	33.20%
3	33.45%

Table B.2: SD2 dataset class distribution.

Class	Percentage
1	10.05%
2	10.05%
3	10.00%
4	10.05%
5	9.85%
6	10.00%
7	10.05%
8	9.90%
9	10.05%
10	10.00%

Table B.3: SD3 dataset class distribution.

Class	**Percentage**
1	33.2%
2	33.6%
3	33.2%

Table B.4: MNIST dataset class distribution.

Class	**Percentage**
1	9.86 %
2	11.25 %
3	9.98 %
4	10.20 %
5	9.74 %
6	9.01 %
7	9.82 %
8	10.41 %
9	9.75 %
10	9.94 %

Table B.5: Glass dataset class distribution.

Class	**Percentage**
1	32.71%
2	35.51%
3	7.94%
5	6.07%
6	4.20%
7	13.55%

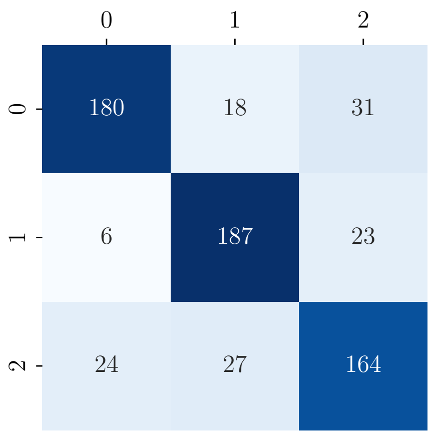
Table B.6: ISOLET dataset class distribution.

Class	Percentage
1	3.84%
2	3.84%
3	3.84%
4	3.84%
5	3.84%
6	3.82%
7	3.84%
8	3.84%
9	3.84%
10	3.84%
11	3.84%
12	3.84%
13	3.83%
14	3.84%
15	3.84%
16	3.84%
17	3.84%
18	3.84%
19	3.84%
20	3.84%
21	3.84%
22	3.84%
23	3.84%
24	3.84%
25	3.84%
26	3.84%

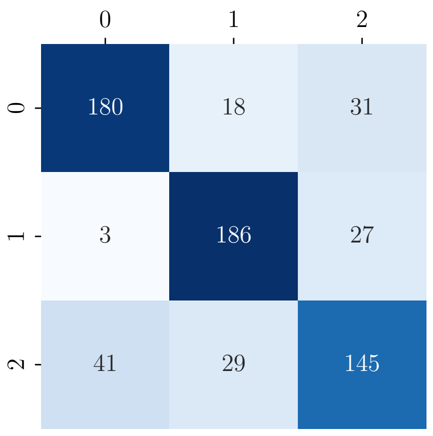
Appendix C

Confusion Matrix

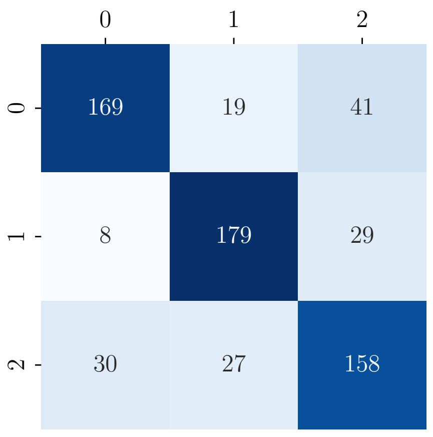
This section contains the confusion matrix not included in section 6.4



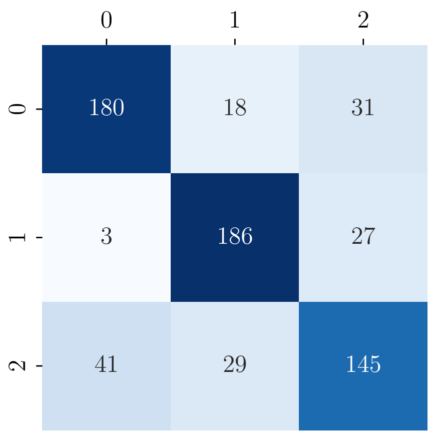
(a) SD1 dataset with all features.



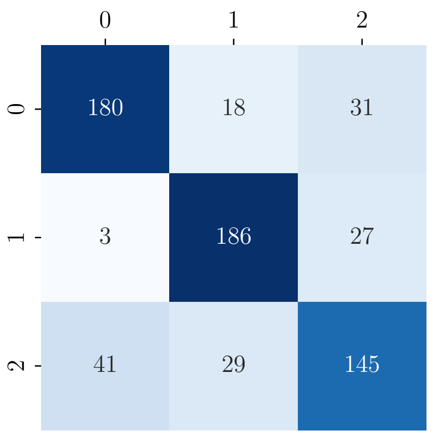
(b) SD1 dataset with Fisher's score method.



(c) SD1 dataset with Gini score method.

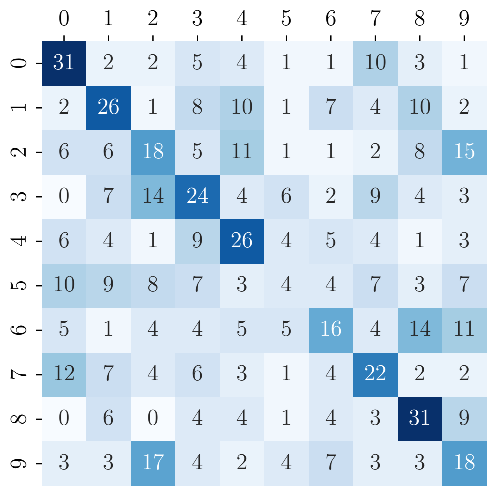


(d) SD1 dataset with RFE method.



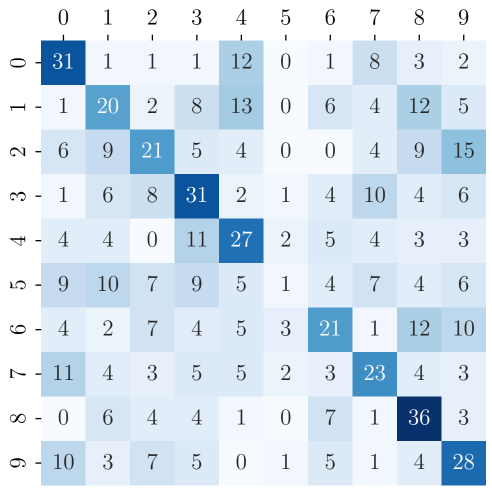
(e) SD1 dataset with RFC method.

Figure C.1: Confusion matrix for the SD1 dataset



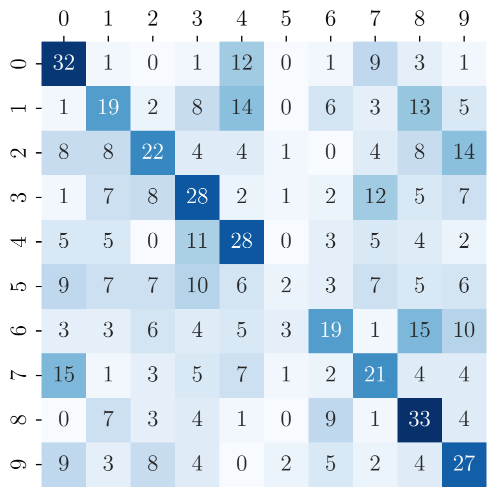
	0	1	2	3	4	5	6	7	8	9
0	31	2	2	5	4	1	1	10	3	1
1	2	26	1	8	10	1	7	4	10	2
2	6	6	18	5	11	1	1	2	8	15
3	0	7	14	24	4	6	2	9	4	3
4	6	4	1	9	26	4	5	4	1	3
5	10	9	8	7	3	4	4	7	3	7
6	5	1	4	4	5	5	16	4	14	11
7	12	7	4	6	3	1	4	22	2	2
8	0	6	0	4	4	1	4	3	31	9
9	3	3	17	4	2	4	7	3	3	18

(a) SD2 dataset with all features.



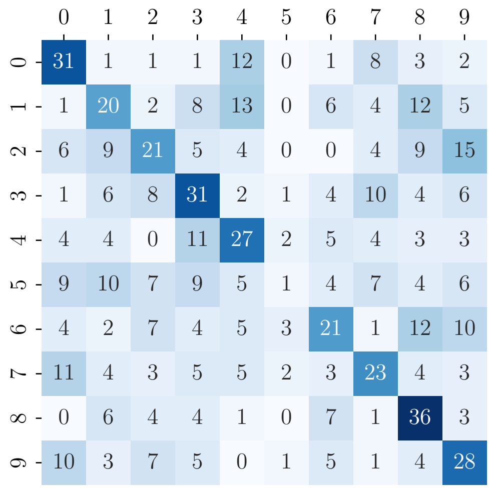
	0	1	2	3	4	5	6	7	8	9
0	31	1	1	1	12	0	1	8	3	2
1	1	20	2	8	13	0	6	4	12	5
2	6	9	21	5	4	0	0	4	9	15
3	1	6	8	31	2	1	4	10	4	6
4	4	4	0	11	27	2	5	4	3	3
5	9	10	7	9	5	1	4	7	4	6
6	4	2	7	4	5	3	21	1	12	10
7	11	4	3	5	5	2	3	23	4	3
8	0	6	4	4	1	0	7	1	36	3
9	10	3	7	5	0	1	5	1	4	28

(b) SD2 dataset with Fisher's score method.



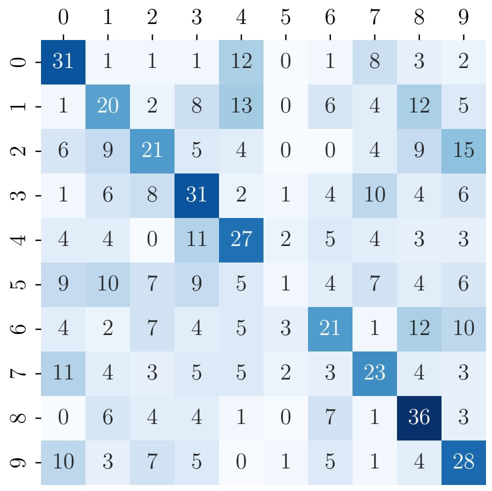
	0	1	2	3	4	5	6	7	8	9
0	32	1	0	1	12	0	1	9	3	1
1	1	19	2	8	14	0	6	3	13	5
2	8	8	22	4	4	1	0	4	8	14
3	1	7	8	28	2	1	2	12	5	7
4	5	5	0	11	28	0	3	5	4	2
5	9	7	7	10	6	2	3	7	5	6
6	3	3	6	4	5	3	19	1	15	10
7	15	1	3	5	7	1	2	21	4	4
8	0	7	3	4	1	0	9	1	33	4
9	9	3	8	4	0	2	5	2	4	27

(c) SD2 dataset with Gini score method.



	0	1	2	3	4	5	6	7	8	9
0	31	1	1	1	12	0	1	8	3	2
1	1	20	2	8	13	0	6	4	12	5
2	6	9	21	5	4	0	0	4	9	15
3	1	6	8	31	2	1	4	10	4	6
4	4	4	0	11	27	2	5	4	3	3
5	9	10	7	9	5	1	4	7	4	6
6	4	2	7	4	5	3	21	1	12	10
7	11	4	3	5	5	2	3	23	4	3
8	0	6	4	4	1	0	7	1	36	3
9	10	3	7	5	0	1	5	1	4	28

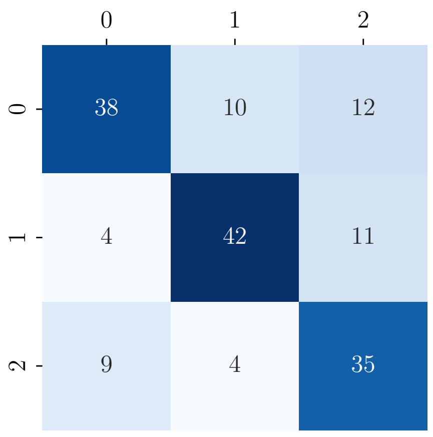
(d) SD2 dataset with RFE method.



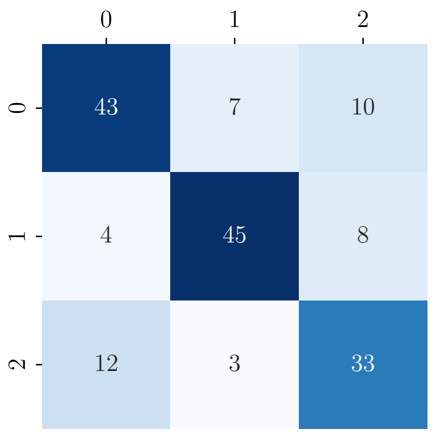
	0	1	2	3	4	5	6	7	8	9
0	31	1	1	1	12	0	1	8	3	2
1	1	20	2	8	13	0	6	4	12	5
2	6	9	21	5	4	0	0	4	9	15
3	1	6	8	31	2	1	4	10	4	6
4	4	4	0	11	27	2	5	4	3	3
5	9	10	7	9	5	1	4	7	4	6
6	4	2	7	4	5	3	21	1	12	10
7	11	4	3	5	5	2	3	23	4	3
8	0	6	4	4	1	0	7	1	36	3
9	10	3	7	5	0	1	5	1	4	28

(e) SD2 dataset with RFC method.

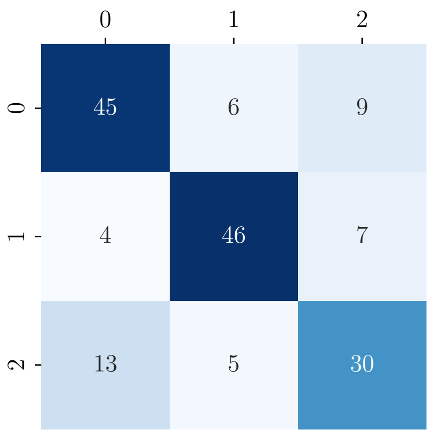
Figure C.2: Confusion matrix for the SD2 dataset



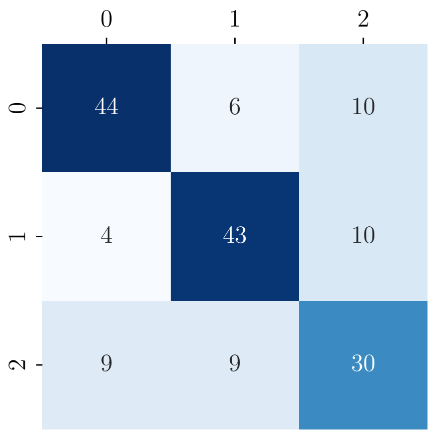
(a) SD3 dataset with all features.



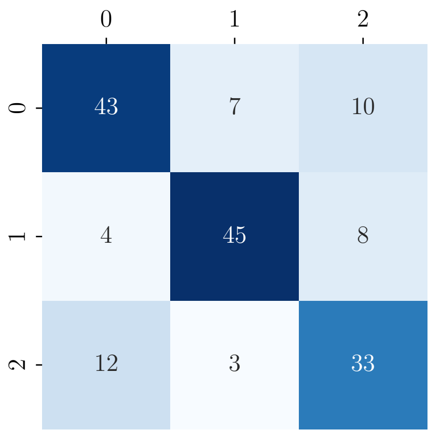
(b) SD3 dataset with Fisher's score method.



(c) SD3 dataset with Gini score method.

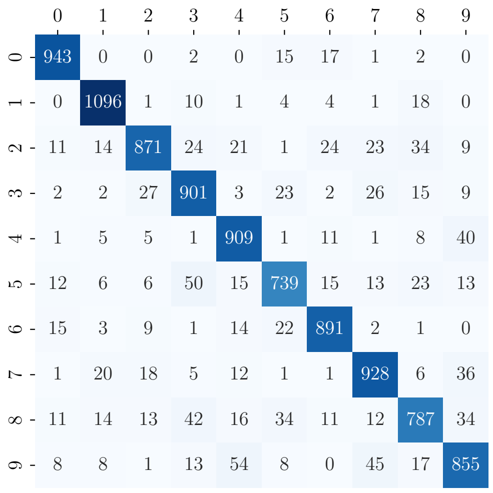


(d) SD3 dataset with RFE method.



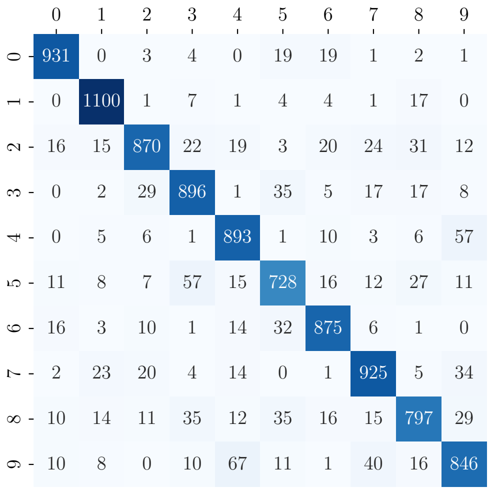
(e) SD3 dataset with RFC method.

Figure C.3: Confusion matrix for the SD3 dataset



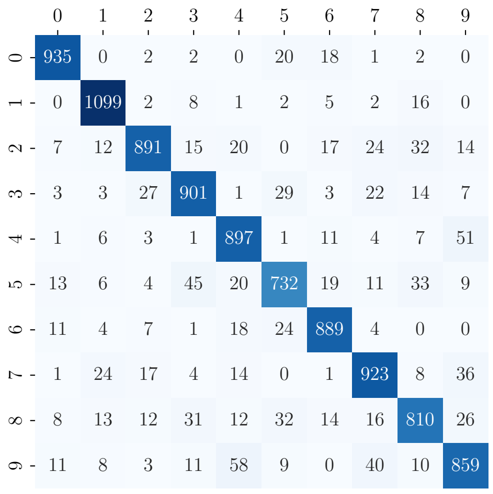
	0	1	2	3	4	5	6	7	8	9
0	943	0	0	2	0	15	17	1	2	0
1	0	1096	1	10	1	4	4	1	18	0
2	11	14	871	24	21	1	24	23	34	9
3	2	2	27	901	3	23	2	26	15	9
4	1	5	5	1	909	1	11	1	8	40
5	12	6	6	50	15	739	15	13	23	13
6	15	3	9	1	14	22	891	2	1	0
7	1	20	18	5	12	1	1	928	6	36
8	11	14	13	42	16	34	11	12	787	34
9	8	8	1	13	54	8	0	45	17	855

(a) MNIST dataset with RENT method.



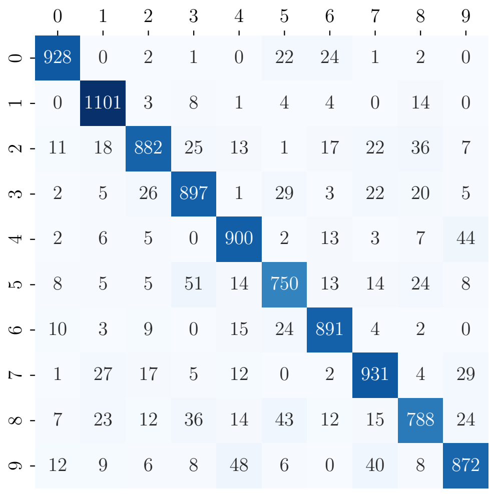
	0	1	2	3	4	5	6	7	8	9
0	931	0	3	4	0	19	19	1	2	1
1	0	1100	1	7	1	4	4	1	17	0
2	16	15	870	22	19	3	20	24	31	12
3	0	2	29	896	1	35	5	17	17	8
4	0	5	6	1	893	1	10	3	6	57
5	11	8	7	57	15	728	16	12	27	11
6	16	3	10	1	14	32	875	6	1	0
7	2	23	20	4	14	0	1	925	5	34
8	10	14	11	35	12	35	16	15	797	29
9	10	8	0	10	67	11	1	40	16	846

(b) MNIST dataset with Fisher's score method.

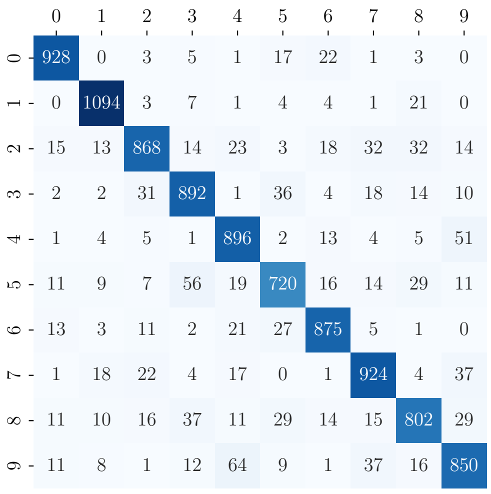


	0	1	2	3	4	5	6	7	8	9
0	935	0	2	2	0	20	18	1	2	0
1	0	1099	2	8	1	2	5	2	16	0
2	7	12	891	15	20	0	17	24	32	14
3	3	3	27	901	1	29	3	22	14	7
4	1	6	3	1	897	1	11	4	7	51
5	13	6	4	45	20	732	19	11	33	9
6	11	4	7	1	18	24	889	4	0	0
7	1	24	17	4	14	0	1	923	8	36
8	8	13	12	31	12	32	14	16	810	26
9	11	8	3	11	58	9	0	40	10	859

(c) MNIST dataset with Gini score method.



	0	1	2	3	4	5	6	7	8	9
0	928	0	2	1	0	22	24	1	2	0
1	0	1101	3	8	1	4	4	0	14	0
2	11	18	882	25	13	1	17	22	36	7
3	2	5	26	897	1	29	3	22	20	5
4	2	6	5	0	900	2	13	3	7	44
5	8	5	5	51	14	750	13	14	24	8
6	10	3	9	0	15	24	891	4	2	0
7	1	27	17	5	12	0	2	931	4	29
8	7	23	12	36	14	43	12	15	788	24
9	12	9	6	8	48	6	0	40	8	872

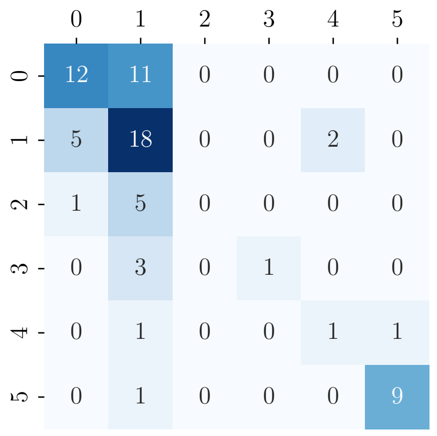


	0	1	2	3	4	5	6	7	8	9
0	928	0	3	5	1	17	22	1	3	0
1	0	1094	3	7	1	4	4	1	21	0
2	15	13	868	14	23	3	18	32	32	14
3	2	2	31	892	1	36	4	18	14	10
4	1	4	5	1	896	2	13	4	5	51
5	11	9	7	56	19	720	16	14	29	11
6	13	3	11	2	21	27	875	5	1	0
7	1	18	22	4	17	0	1	924	4	37
8	11	10	16	37	11	29	14	15	802	29
9	11	8	1	12	64	9	1	37	16	850

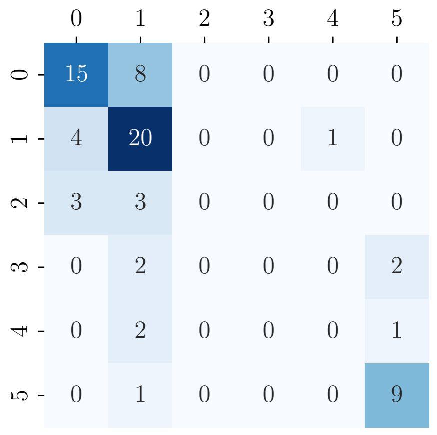
(d) MNIST dataset with RFE method.

(e) MNIST dataset with RFC method.

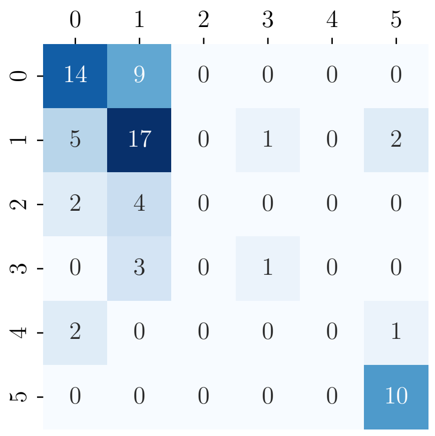
Figure C.4: Confusion matrix for the MNIST dataset



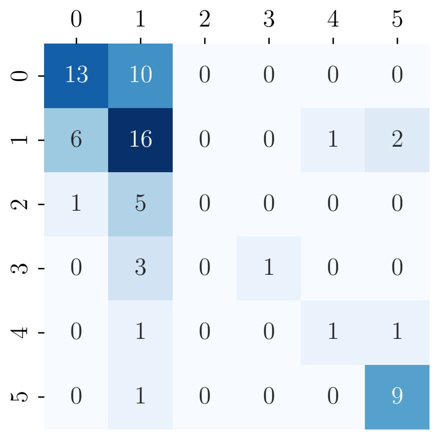
(a) glass dataset with RENT method.



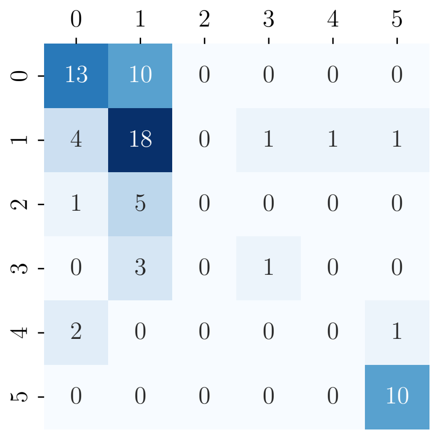
(b) glass dataset with Fisher's score method.



(c) glass dataset with Gini score method.

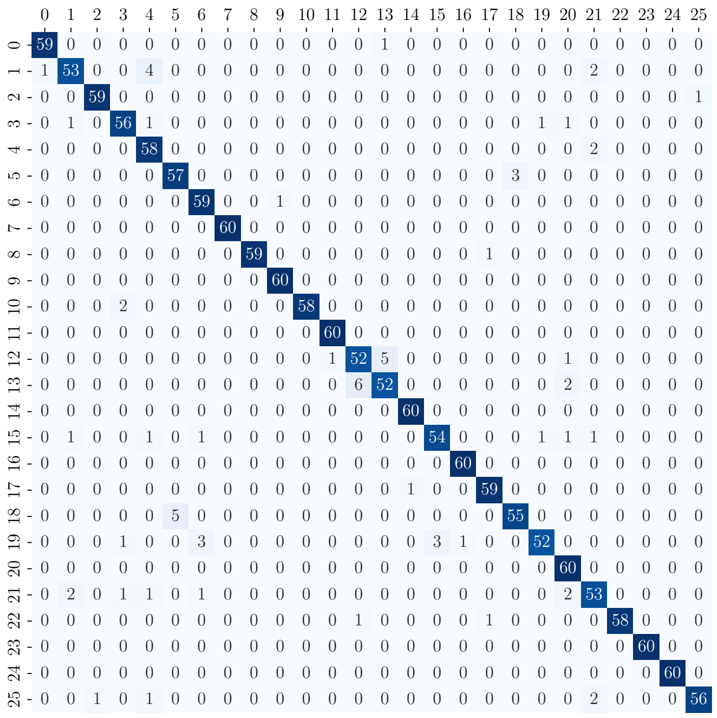


(d) glass dataset with RFE method.



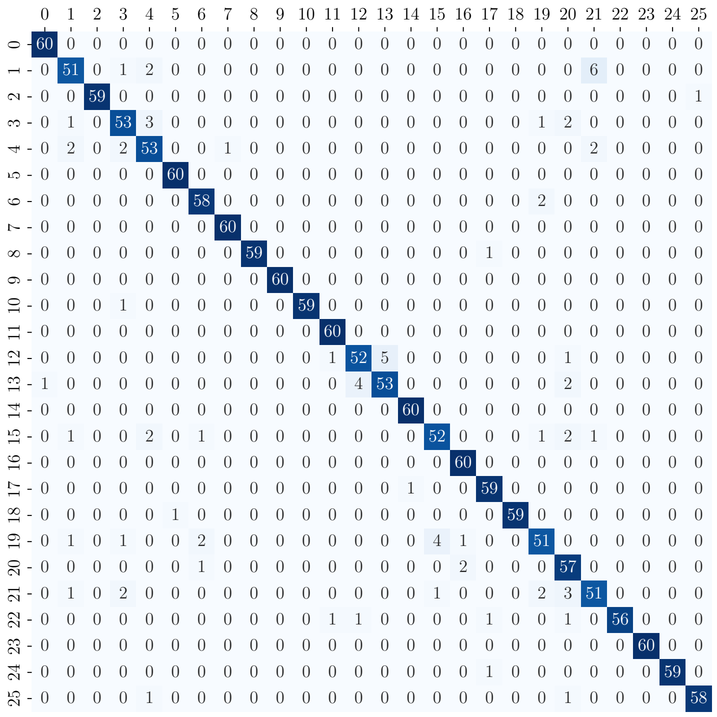
(e) glass dataset with RFC method.

Figure C.5: Confusion matrix for the glass dataset



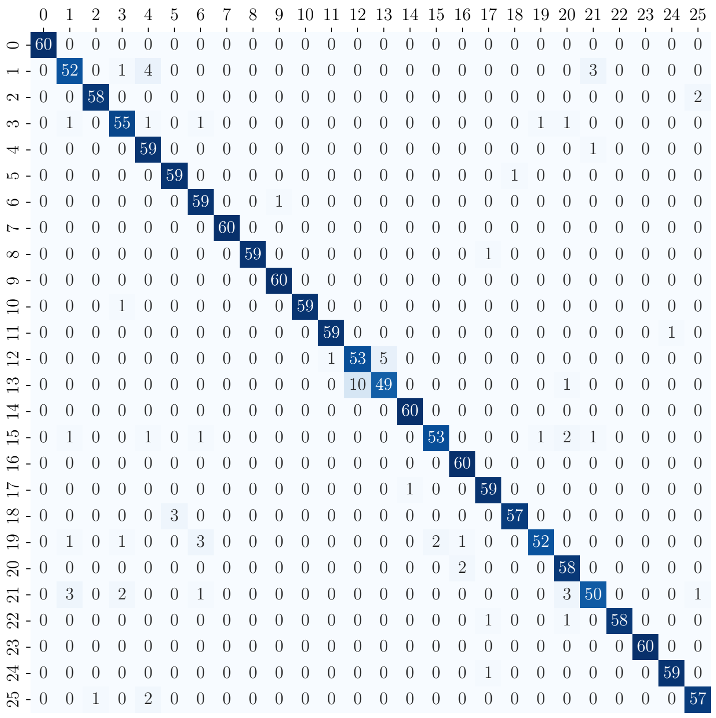
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	59	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	53	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
2	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
3	0	1	0	56	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
4	0	0	0	0	58	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
5	0	0	0	0	0	57	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0
6	0	0	0	0	0	0	59	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	59	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	2	0	0	0	0	0	0	58	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	1	52	5	0	0	0	0	0	0	1	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	6	52	0	0	0	0	0	0	2	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0
15	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	54	0	0	0	1	1	1	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	59	0	0	0	0	0	0	0	0
18	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	55	0	0	0	0	0	0	0
19	0	0	0	1	0	0	3	0	0	0	0	0	0	0	0	3	1	0	0	52	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0
21	0	2	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2	53	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	58	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0
25	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	56

(a) ISOLET dataset with RENT method.



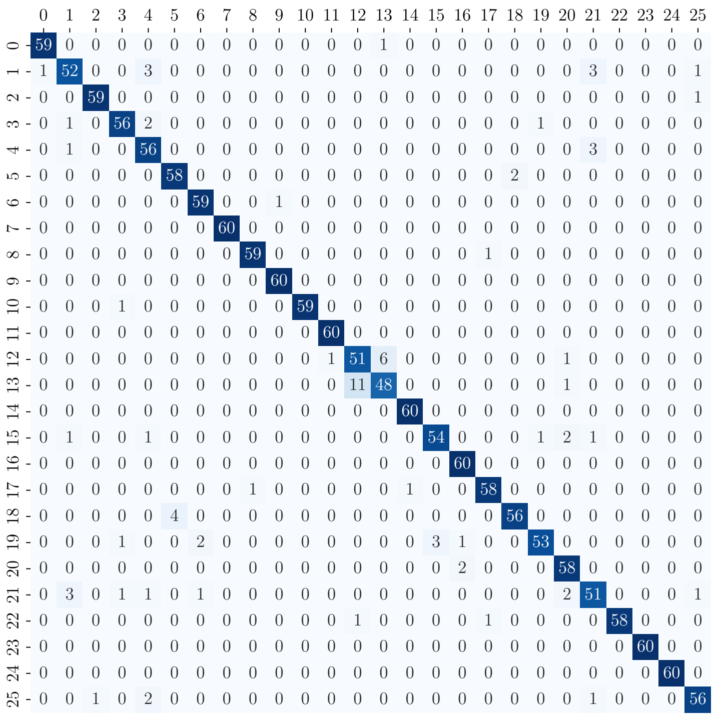
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	51	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0
2	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
3	0	1	0	53	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	0
4	0	2	0	2	53	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
5	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	58	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0
7	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	59	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	1	0	0	0	0	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	1	52	5	0	0	0	0	0	0	1	0	0	0	0	0
13	1	0	0	0	0	0	0	0	0	0	0	0	4	53	0	0	0	0	0	0	2	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0
15	0	1	0	0	2	0	1	0	0	0	0	0	0	0	0	52	0	0	0	1	2	1	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	59	0	0	0	0	0	0	0	0
18	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	59	0	0	0	0	0	0	0
19	0	1	0	1	0	0	2	0	0	0	0	0	0	0	0	4	1	0	0	51	0	0	0	0	0	0
20	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	2	0	0	0	57	0	0	0	0	0
21	0	1	0	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2	3	51	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	1	0	56	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	59	0
25	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	58

(b) ISOLET dataset with Fisher's score method.



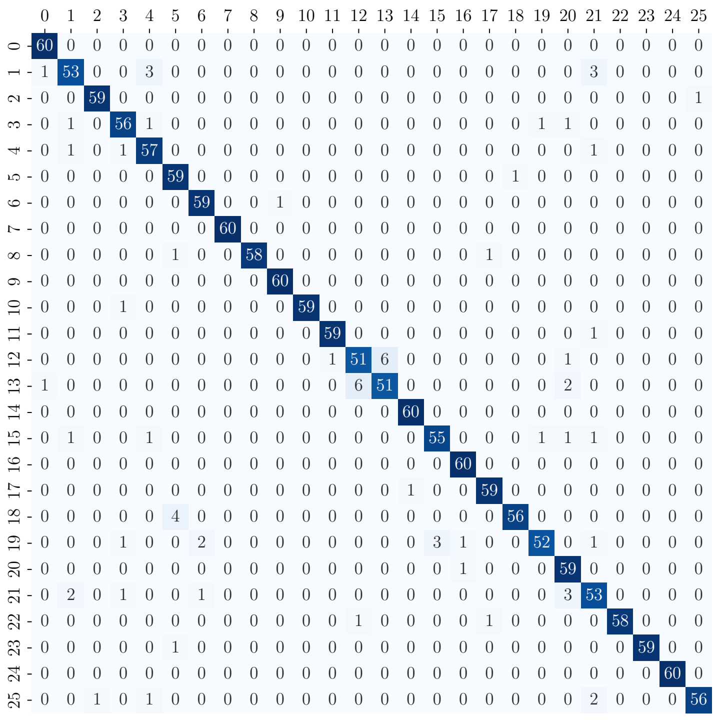
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	52	0	1	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0
2	0	0	58	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
3	0	1	0	55	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
4	0	0	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
5	0	0	0	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
6	0	0	0	0	0	0	59	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	59	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	1	0	0	0	0	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	1	0
12	0	0	0	0	0	0	0	0	0	0	0	1	53	5	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	10	49	0	0	0	0	0	0	1	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0
15	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	53	0	0	0	1	2	1	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	59	0	0	0	0	0	0	0	0
18	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	57	0	0	0	0	0	0	0
19	0	1	0	1	0	0	3	0	0	0	0	0	0	0	0	2	1	0	0	52	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	58	0	0	0	0	0
21	0	3	0	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	3	50	0	0	0	1
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	58	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	59	0
25	0	0	1	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	57

(c) ISOLET dataset with Gini score method.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	59	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	52	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	1
2	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
3	0	1	0	56	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
4	0	1	0	0	56	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0
5	0	0	0	0	0	58	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
6	0	0	0	0	0	0	59	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	59	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	1	0	0	0	0	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	1	51	6	0	0	0	0	0	0	1	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	11	48	0	0	0	0	0	0	1	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0
15	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	54	0	0	0	1	2	1	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	58	0	0	0	0	0	0	0	0
18	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	56	0	0	0	0	0	0	0
19	0	0	0	1	0	0	2	0	0	0	0	0	0	0	0	3	1	0	0	53	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	58	0	0	0	0	0
21	0	3	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2	51	0	0	0	1
22	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	58	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0
25	0	0	1	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	56

(d) ISOLET dataset with RFE method.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	53	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0
2	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
3	0	1	0	56	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
4	0	1	0	1	57	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
5	0	0	0	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
6	0	0	0	0	0	0	59	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	1	0	0	58	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	1	0	0	0	0	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	59	0	0	0	0	0	0	0	0	0	1	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	1	51	6	0	0	0	0	0	0	1	0	0	0	0	0
13	1	0	0	0	0	0	0	0	0	0	0	0	6	51	0	0	0	0	0	0	2	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0
15	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	55	0	0	0	1	1	1	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	59	0	0	0	0	0	0	0	0
18	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	56	0	0	0	0	0	0	0
19	0	0	0	1	0	0	2	0	0	0	0	0	0	0	0	3	1	0	0	52	0	1	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	59	0	0	0	0	0
21	0	2	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	3	53	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	58	0	0	0
23	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	59	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0
25	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	56

(e) ISOLET dataset with RFC method.

Figure C.6: Confusion matrix for the ISOLET dataset