Work 2 - Classification with Lazy Learning and SVM

Zachary Parent, Sheena Lang, Kacper Poniatowski and Carlos Jiménez

November 2, 2024

1 Introduction

This is the introduction section of the report. It provides background information and context for the study, states the research question or hypothesis, and briefly outlines the approach taken. A report structure overview is also provided to guide the reader through the document.

1.1 Background Information

Classification in the context of machine learning is the process of predicting the class or category of a given data point based on its features. For example: given a set of emails, classify each email as spam or not spam. It is a supervised learning technique that is used to assign labels to data points based on their characteristics. Classification is an ideal technique for the required analysis as outlined in this report.

K-Nearest Neighbour (KNN) is a supervised machine learning algorithm that has been chosen as the primary method for this study. It is a simple yet powerful algorithm that classifies new cases based on similarity to existing data points. While KNN can be used for both classification and regression problems, our focus is on classification due to the nature of the datasets used [5].

This report details the analysis of two datasets: 'hepatitis' and 'mushroom'. These are two of many datasets provided to us as part of this assignment. We decided upon these two datasets due to the stark contrast between them in terms of complexity and size.

1.2 Research Question or Hypothesis

The primary research question addressed in this report is: 'Can the k-Nearest Neighbour algorithm effectively classify data points in the hepatitis and mushroom datasets with high accuracy'?.

The specific objectives of this study are:

1. To evaluate KNN's classification performance on datasets of varying sizes and complexities
2. To determine optimal KNN hyperparameter settings for each dataset
3. To assess the algorithm's robustness and limitations in different scenarios

We hypothesize that KNN will perform excellently on the mushroom dataset due to the simplicity of the data, but may struggle with the hepatitis dataset due to the complexity of the data in that dataset.

1.3 Approach and Methodology

The approach taken in this report involves the following steps:

1. Data Preprocessing: Both datasets are preprocessed, ensuring the data is clean and ready for analysis.
2. Model Training: Different combinations of hyperparameters are used to train the KNN algorithm on the training dataset.
3. Model Evaluation: The performance of the best KNN algorithm from the previous step is evaluated using the test dataset.

1.4 Report Structure

The report is structured as follows:

1. Introduction (Section 1): Provides background information, states the research question, and outlines the approach.
2. Data (Section 2): Describes the data used in the study, along with the source, characteristics, relevance, and the preprocessing techniques that were applied.
3. Methods (Section 3): Describes the methodology used in the study, including the algorithms, techniques, and tools used.
4. Results and Analysis (Section 4): Findings of the study are presented, including relevant data, statistics, and figures to help visualise the data.
5. Conclusion (Section 5): Summarizes main findings of the study and their significance.

2 Data

2.1 Dataset

In this section, we describe the selection criteria that guided our choice of the Mushroom and Hepatitis datasets and analyze their respective characteristics.

2.1.1 Dataset Selection

In this project, we aimed to select two datasets that offer substantial variability across different aspects to evaluate the performance of Support Vector Machine (SVM) and k-Nearest Neighbors (KNN) algorithms. The criteria for dataset selection were centered around having one dataset that is small and another that is large, allowing us to analyze both the efficiency and effectiveness of these models across differing dataset sizes. A smaller dataset, enables rapid experimentation and comparison of SVM and KNN performance. Meanwhile, a larger dataset, provides an excellent opportunity to evaluate the benefits of reduction methods in KNN, especially regarding storage efficiency and reduced running time [11].

Moreover, we wanted to assess the models' ability to handle datasets with different types of attributes. For this purpose, we wanted to select one dataset with primarily nominal attributes and another with numerical features. This allows us to evaluate how well each algorithm handles the representation and processing of different data types. Additionally, class distribution was a critical factor in our selection. By choosing one dataset with a balanced distribution and another with a notable class imbalance, we aim to observe how SVM and KNN, particularly with reduction, perform in scenarios where the data is skewed. Lastly, we prioritized finding datasets with varying levels of missing data to examine how effectively the algorithms manage incomplete information.

Based on these criteria, we selected the Hepatitis and Mushroom datasets, as they provide the most distinct and complementary combination for our evaluation.

2.1.2 Dataset Characteristics

The Mushroom dataset, with 8,124 instances, is much larger (see Figure 1) than the Hepatitis dataset's 155 instances, making it ideal for examining the computational benefits of reduction techniques. The Mushroom dataset focuses on predicting whether a mushroom is poisonous or edible based on 22 nominal attributes, such as cap shape and color, whereas the Hepatitis dataset aims to predict whether a hepatitis patient will live or die, using a mix of both nominal and numeric features like age and liver size. This allows for a comparison of model handling for fully categorical data versus mixed data types.

Class distribution is another key distinction. The Mushroom dataset is nearly balanced, with only a 1.8% class deviation, while the Hepatitis dataset has a 29.35% class deviation, with 79.35% of instances in the majority class (see Figure 2), making it ideal for testing each model's robustness to imbalanced data. Missing data is also more prevalent in the Mushroom dataset (48.2%) compared to Hepatitis (20.01%), providing insights into each model's ability to handle incomplete data.

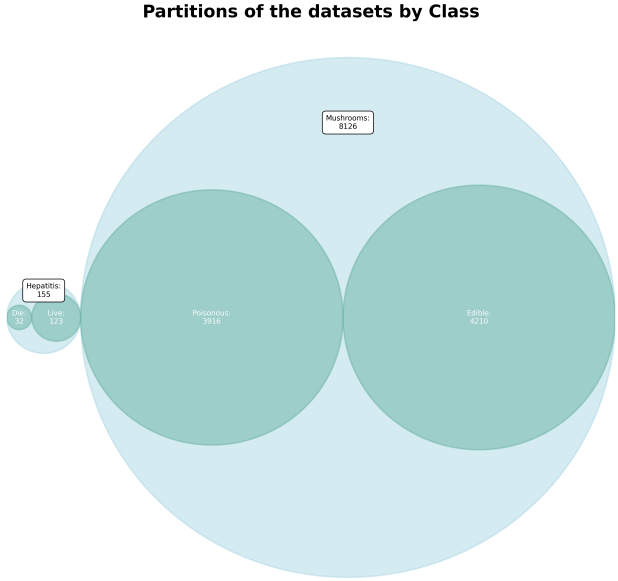

Figure 1: Dataset partitions

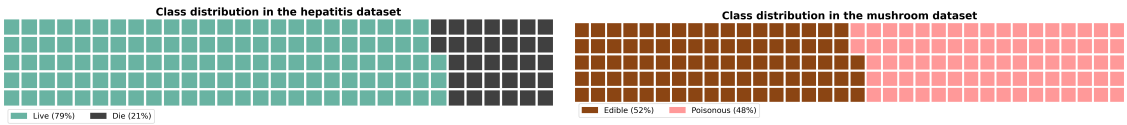

Figure 2: Class distributions

2.2 Data Preprocessing

This section outlines the preprocessing steps applied to the Hepatitis and Mushroom datasets.

2.2.1 Dealing with Different Ranges and different types

To manage the varying types and ranges of attributes in our datasets, we implemented specific preprocessing techniques. For the nominal attributes in both the Mushroom and Hepatitis datasets, we used label encoding. This technique converts categorical values into numerical labels, enabling the algorithms to interpret the data correctly. While we considered one-hot encoding to avoid implying any ordinal relationship among categories, we opted for label encoding due to its simplicity and reduced dimensionality, as one-hot encoding would significantly increase dimensions and lead to a sparse space, making accurate predictions more challenging, particularly for KNN with the Mushroom dataset's numerous nominal features.

For the numerical attributes in the Hepatitis dataset, we applied min-max scaling to rescale the data to a fixed range of [0, 1]. This normalization is crucial for distance-based algorithms like KNN and SVM, ensuring all features contribute equally to model performance. We evaluated other scaling methods, such as standardization, but chose min-max scaling for its effectiveness in maintaining the original data distribution [3].

2.2.2 Dealing with Missing Values

Addressing missing values is a critical step in preparing our datasets for analysis, as they can significantly impact model performance. In the case of the nominal attributes in both the Mushroom and Hepatitis datasets, we opted to impute missing values with the majority class. This method is straightforward and effective for maintaining dataset integrity. However, it can also introduce bias, particularly in the Hepatitis dataset, where the majority class represents 79.35% of instances. Relying on this method may lead to a situation where the imputed values disproportionately favor the majority class, thereby affecting the overall distribution and potentially skewing the results [3].

For the numerical attributes in the Hepatitis dataset, we used the mean of the available data to fill in missing values. This approach preserves the overall data distribution and is easy to implement, but it is not without its drawbacks. The mean can be heavily influenced by outliers, which might distort the data and lead to less accurate predictions. This is especially important in medical datasets, where extreme values may carry significant meaning.

We also considered employing K-Nearest Neighbors (KNN) for imputing missing values, as it could provide a more nuanced approach by considering the nearest data points for each instance. However, we ultimately decided against this option to avoid introducing bias into our evaluation. Since KNN is one of the algorithms we are testing, using it for imputation could influence its performance and lead to skewed results. Therefore, we chose the more straightforward methods of majority class imputation for nominal values and mean imputation for numerical values, allowing for a clearer assessment of the models' effectiveness without confounding factors.

3 Methods

This section describes the methodology used in the study. It should include details about data collection, experimental design, and analytical techniques.

3.1 k-Nearest Neighbors (KNN)

This section describes the k-Nearest Neighbors (KNN) algorithm and its implementation in our study.

3.1.1 Algorithm Overview

The KNN algorithm operates on a simple yet effective principle: when classifying new data points, it examines the k closest training examples and assigns the most common class among these neighbors (where k is a user-defined hyperparameter) [5]. The algorithm's effectiveness relies on two fundamental assumptions:

- Locality: Points that are close to each other are likely to have the same class.
- Smoothness: The classification boundary between classes is relatively smooth.

One key feature of KNN is it employs neighbor-based classification, where the classification of a new data point is determined by majority voting among its k-nearest neighbors. The value of k is one of the most important tunable hyperparameters, as it significantly influences the algorithm's behaviour:

- Small k values (e.g., k=1 or k=3): More sensitive to local patterns but susceptible to noise.
- Large k values: More robust to noise but may overlook important local patterns.
- Even vs. Odd k values: Even k values result in ties, which may require additional rules to break.

The dependent variable in our datasets we aim to predict is categorical, so while KNN can be used for both classification and regression problems our focus is on classification.

While the KNN algorithm has it's merits in terms of simplicity and interpretability, it also has several disadvantages [11]:

- Computationally expensive: As the number of training examples grows, the algorithm's complexity increases[14].
- Sensitive to irrelevant features: The algorithm treats all features equally, so irrelevant features can negatively impact performance.
- Curse of dimensionality: As the number of features increases, the algorithm requires more data to maintain performance [8].

All three of the above mentioned disadvantages all relate to the features of the dataset, and how they can impact the performance of the algorithm. They create a compounding effect: more features lead to higher computational cost, while making the algorithm more susceptible to noise and irrelevant features, and also requiring more data to maintain performance. This is why the use of feature selection and reduction techniques are imperative when working with the KNN algorithm to increase performance [11, 5].

3.1.2 Implementation Details

The K-Nearest Neighbors (KNN) algorithm is implemented using Python with various libraries and tools. Below are the specific implementation details:

Distance calculations

The Euclidean distance is used to measure the distance between data points. This distance metric is the most commonly used distance metric in KNN algorithms due to its simplicity and effectiveness in measuring the similarity between data points[13]. It operates on the principle of calculating the straight-line distance between two points in a Euclidean space, hence it's simplicity.

The formula for Euclidean distance between two vectors x and y is:

$$d = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

where x_i and y_i are the ith elements of vectors x and y, respectively.

The Manhattan distance is another distance metric that can be used in KNN algorithms. It is also frequently used with the KNN algorithm, albeit not as common as the Euclidean distance. The Manhattan distance is calculated by summing the absolute differences between the coordinates of two points.

The formula for Manhattan distance between two vectors x and y is:

$$d = \sum_{i=1}^{n}|x_i - y_i|$$

where x_i and y_i are the ith elements of vectors x and y, respectively.

The Chebychev distance less commonly used than the Euclidean and Manhattan distances in relation to the KNN algorithm, but it is still a valid distance metric and operates effectively in measuring the similarity between data points. The Chebychev distance is calculated by taking the maximum absolute difference between the coordinates of two points. This differs from Manhattan distance in that it takes the maximum absolute difference, rather than the sum.

The formula for Chebychev distance between two vectors x and y is:

$$d = \max_{i=1}^{n} |x_i - y_i|$$

where x_i and y_i are the ith elements of vectors x and y, respectively.

The Manhattan and Euclidean distance metrics are mandatory as per the assignment requirements, while the Chebychev distance metric was included as an additional distance metric. We chose Chebyshev distance because as described above, only the maximum difference is considered, not the sum of the differences. This is particularly important when working with high dimensional spaces (which is the case for our datasets) as it can help reduce the computational cost and improve time efficiency.

Weighting Schemes

In the KNN algorithm, the choice of weighting scheme can significantly impact the classification results. The following weighting schemes were implemented in this study:

In uniform weighting, all neighbours have equal weight in the voting process. This is the default weighting scheme in KNN. An advantage of uniform weighting is that it is simple and computationally efficient, but it may not be optimal for imbalanced datasets.

With ReliefF weighting, neighbours are weighted based on their relevance to the target class. This provides a more nuanced approach to weighting as it considers the importance of each neighbour in the classification process, potentially improving performance of the algorithm.

Information gain weighting, Neighbours are weighted based on gain in information in the context of the target variable[1] Similarly to ReliefF weighting, this weighting scheme assigns higher weights to neighbours that provide more information about the target class.

Voting Schemes

In the KNN algorithm, the voting scheme determines how the class label of a new data point is determined based on the class labels of its k-nearest neighbors. The following voting schemes were implemented in this study:

In the majority voting scheme, the class label with the highest frequency among the k-nearest neighbors is assigned to the new data point. As well as this, each vote is given equal weight in the voting process[4] This is the default voting scheme in KNN and is simple and easy to implement.

With inverse distance weighting voting, the class labels of the k-nearest neighbors are weighted based on their distance from the new data point. While there are different methods to perform this weighting, the most simple version is to take a neighbour's vote to be the inverse of its distance to q:

$$w_i = \frac{1}{d(q, x_i)}$$

where w_i is the weight of the ith neighbour, $d(q, x_i)$ is the distance between the new data point q and the ith neighbour x_i.

Then the votes are summed and the class with the highest votes is returned[4]

Shepard's method is another voting scheme that can be used in KNN algorithms. It employs the use of an exponential function to weight the votes of the neighbours based on their distance from the new data point, rather than the inverse of the distance[6]

The formula for Shepard's voting scheme is:

$$Vote(y_j) = \sum_{i=1}^{k} e^{-d(\mathbf{q},\mathbf{x}_i)^2} 1(y_j, y_c) \tag{1}$$

where $Vote(y_j)$ is the vote for class y_j, $d(\mathbf{q}, \mathbf{x}_i)$ is the distance between the new data point $\mathbf{q}$ and the ith neighbour $\mathbf{x}_i$, and $1(y_j, y_c)$ is the indicator function that returns 1 if y_j is the same as the class label y_c of the ith neighbour, and 0 otherwise.

Neighbor Selection

The choice of the number of neighbors (k) is a critical hyperparameter in the KNN algorithm, as previously discussed in this report. Different values of k can significantly impact the algorithm's performance, with smaller values being more sensitive to noise and larger values potentially overlooking important local patterns. It's imperative to choose an optimal value of k that balances these trade-offs and maximizes the algorithm's performance. In this study, the following values of k where examined: [1, 3, 5, 7]

Basic Algorithm Steps

The KNN algorithm can be summarized in the following steps:

1. Data Preparation (see section 2 on Data).
2. Model Configuration: Different combinations of hyperparameters (k values, distance metrics, weighting schemes, and voting schemes) were evaluated.
3. Cross-validation: For each configuration, the model was trained and evaluated using cross-validation across the 10 pre-defined folds.
4. Performance Evaluation: Various metrics including accuracy, F1 score, and confusion matrix elements (TP, TN, FP, FN) were computed.
5. Time Measurement: Training and testing times were recorded for each configuration.
6. Results Compilation: The results for each configuration were saved in CSV files for further analysis.

Libraries and Tools

The following libraries and tools were used for the implementation of the KNN algorithm in our study:

- **Python**: The primary programming language used for the implementation of the KNN algorithm.
- **NumPy**: A useful package for scientific computing with Python, used for numerical operations [10].
- **Pandas**: A data manipulation library in Python, used for data preprocessing and analysis [15].
- **Matplotlib**: A plotting library in Python, used for data visualization [12].
- **Seaborn**: A data visualization library in Python, used for creating informative and attractive statistical graphics [19].
- **SciPy**: A scientific computing library in Python, used for scientific and technical computing [18].

Data Preprocessing

Before parameter tuning, comprehensive preprocessing pipelines were implemented for both the hepatitis and mushroom datasets using scikit-learn's ColumnTransformer and Pipeline classes. For more information on data preprocessing, refer to Section 2 of the report.

Parameter Search Implementation

The KNN algorithm's performance depends significantly on the careful tuning of several key parameters. The optimal combination of parameters was determined using a custom grid search function that evaluated all possible combinations of:

- k values
- Distance metrics
- Voting schemes
- Weighting schemes

The grid search function was implemented using 'itertools.product' to iterate over each parameter combination. Each combination was evaluated using a cross validation function, which returned a score value for that combination. This score, along with the corresponding hyperparameters, were stored for further analysis.

Performance Metrics

The performance of the KNN models was evaluated using the following metrics:

- Accuracy: The proportion of correct predictions among the total number of cases examined.
- F1 Score: The harmonic mean of precision and recall, providing a balanced measure of the model's performance.
- Confusion Matrix: Including True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).
- Training Time: The time taken to train the model.
- Testing Time: The time taken to make predictions on the test set.

The F1 score was used as our primary metric for evaluating the performance of the KNN models. since it balances the trade-off between precision and recall, and does not report overly favorable results when the dataset is imbalanced. The hepatitis dataset is imbalanced, with a 79 / 31 split between the two classes, so the F1 score is a more reliable metric than accuracy Figure 2

Additionally, in medical diagnostics like hepatitis classification, the costs of different types of errors are significant − missing a positive diagnosis (false negative, impacting recall) could delay critical treatment, while a false positive (impacting precision) could lead to unnecessary medical procedures. The F1 score's balance of precision and recall helps account for both these error types, making it particularly suitable for this dataset.

3.2 Support Vector Machines (SVM)

This section describes the Support Vector Machines (SVM) algorithm and its implementation in our study.

3.2.1 Algorithm Overview

Support Vector Machines (SVM) is a powerful supervised learning algorithm used for classification and regression tasks. The primary objective of SVM is to find the optimal hyperplane that separates different classes in the feature space while maximizing the margin between the classes[2].

The key principles of SVM include:

- Margin Maximization: SVM aims to find the hyperplane that maximizes the margin between classes, which enhances the model's generalization capability.
- Support Vectors: The data points closest to the decision boundary, known as support vectors, play a crucial role in defining the optimal hyperplane.
- Kernel Trick: SVM can handle non-linearly separable data by mapping the input space to a higher-dimensional feature space using kernel functions.

Advantages of SVM include:

- Effectiveness in high-dimensional spaces.
- Versatility through different kernel functions.
- Provides efficient predictions compared to instance-based methods like KNN.

Disadvantages of SVM include:

- Performance highly depends on kernel selection and parameter tuning.
- Becomes computationally intensive with large-scale datasets.

3.2.2 Implementation Details

Unlike for KNN (see subsection 3.1 on K-Nearest Neighbors), we did not implement the SVM algorithm ourselves. Instead, we used the prebuilt implementations from the `sklearn` library [1]. Scikit-learn's `svm` module includes several different implementations of SVM:

- `SVC`: Support Vector Classification, the most commonly used implementation for classification tasks.
- `NuSVC`: Support Vector Classification with Nu-SVC, similar to `SVC` but with a different formulation of the optimization problem.
- `LinearSVC`: Linear Support Vector Classification, a specific variant of `SVC` that uses a linear kernel.
- `SVR`: Support Vector Regression, a variant of SVM for regression tasks.

For our study, we decided to use `SVC` as it is the most commonly used implementation for classification tasks, allowed for the kernel trick (unlike `LinearSVC`), and met our needs.

3.2.3 Kernel Selection

In our study, we explored multiple kernel functions to capture different types of relationships in the data. The kernels used include:

- Linear: $K(x_i, x_j) = x_i^T x_j$
- Polynomial: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d$
- Radial Basis Function (RBF): $K(x_i, x_j) = \exp(-\gamma ||x_i - x_j||^2)$
- Sigmoid: $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$

Where x_i and x_j are feature vectors, γ is a kernel coefficient, r is a constant term, and d is the degree of the polynomial kernel.

[1] Scikit-learn's `svm` module documentation can be found at `https://scikit-learn.org/1.5/modules/svm.html`

3.2.4 Hyperparameter Tuning

The following hyperparameters were tuned in our SVM implementation:

- C: The regularization parameter, which controls the trade-off between achieving a low training error and a low testing error. We explored values [1, 3, 5, 7].
- Kernel: We tested different kernel types including 'linear', 'poly', 'rbf', and 'sigmoid.

3.2.5 Implementation Steps

The SVM classification process in our study followed these steps:

1. Data Preparation (see section 2 on Data).
2. Model Configuration: SVM models were created with different combinations of C values and kernel types.
3. Cross-validation: For each configuration, the model was trained and evaluated using cross-validation across 10 predefined folds.
4. Performance Evaluation: Various metrics including accuracy, F1 score, and confusion matrix elements (TP, TN, FP, FN) were computed.
5. Time Measurement: Training and testing times were recorded for each configuration.
6. Results Compilation: The results for each configuration were saved in CSV files for further analysis.

3.2.6 Multi-class Classification

While simple SVMs are binary classifiers, they can be extended to multi-class classification through various strategies. In our case, however, both the datasets included only two classes, so we did not need to use these strategies.

3.2.7 Performance Metrics

The performance of the SVM models was evaluated using the following metrics:

- Accuracy: The proportion of correct predictions among the total number of cases examined.
- F1 Score: The harmonic mean of precision and recall, providing a balanced measure of the model's performance.
- Confusion Matrix: Including True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).
- Training Time: The time taken to train the model.
- Testing Time: The time taken to make predictions on the test set.

The F1 score was used as our primary metric for evaluating the performance of the SVM models, since it balances the trade-off between precision and recall, and does not report overly favorable results when the dataset is imbalanced.

3.3 Instance Reduction Algorithms

A significant challenge in applying KNN to large datasets is the computational cost associated with searching the entire training set. Additionally, noisy or irrelevant data can negatively impact the model's performance. To overcome these issues, we employ instance reduction techniques. These techniques aim to identify and select a smaller, more representative subset of the training data, leading to faster prediction times and improved accuracy [20, 16].

A variety of rule-based techniques have been proposed in the literature to address the challenges associated with large and noisy datasets. These techniques aim to identify patterns and relationships within the data to select a subset of informative instances.

3.3.1 Algorithms Overview

Condensed Nearest Neighbour Rule

Condensed nearest neighbor rules (CNN) are a family of algorithms that aim to identify a minimal subset of the training data that can represent the entire dataset without significant loss of information. One prominent example is the **Generalized Condensed Nearest Neighbor** (GCNN) algorithm.

GCNN [**?**] is an iterative algorithm that starts with a small subset of the training data and incrementally adds instances that are misclassified by the current KNN model. This process continues until no further instances are misclassified. GCNN aims to identify a minimal consistent subset, a subset of the original data that correctly classifies all of the original instances using the 1-NN rule.

More formally, let $X = \{x_1, x_2, ..., x_n\}$ be the set of training instances and $Y = \{y_1, y_2, ..., y_n\}$ be the corresponding class labels. GCNN can be described as follows:

1. **Initialization:** Select a random instance x_i from X and add it to the condensed set C.
2. **Iteration:** For each instance $x_j \in X \setminus C$:
   - Train a 1-NN classifier on C.
   - If $KNN(x_j) \neq y_j$, then add x_j to C.
3. **Termination:** Repeat step 2 until no new instances are added to C.

GCNN's effectiveness lies in its ability to capture the decision boundaries between classes using a reduced set of instances. However, its performance can be sensitive to the initial instance selection and the order in which instances are processed.

Edited Nearest Neighbour Rule

Edited nearest neighbor rules, on the other hand, focus on removing noisy or outlier instances from the training data.

The **Edited Nearest Neighbor Estimating Class Probabilistic and Threshold (ENNTh)** is a noise-removal technique that builds upon the Edited Nearest Neighbor (ENN) algorithm [20, 17]. ENN aims to improve the generalization ability of KNN by removing instances that are likely to be noise or outliers. ENNTh refines this process by incorporating a threshold, τ, to control the degree of noise removal [17].

ENN traditionally removes an instance if its class label differs from the majority class among its k nearest neighbors. ENNTh introduces a more flexible approach by estimating the class probability of an instance based on its k nearest neighbors. Let $N_k(x_i)$ denote the set of k nearest neighbors of x_i. The class probability of x_i is estimated as:

$$P(y_i|x_i) = \frac{|\{x_j \in N_k(x_i) : y_j = y_i\}|}{k} \tag{2}$$

An instance x_i is removed only if $P(y_i|x_i) < \tau$. This thresholding mechanism allows for a more nuanced approach to noise removal, where instances with a higher probability of belonging to their assigned class are retained, even if they are not in the majority class among their neighbors.

By adjusting the threshold τ, ENNTh can control the trade-off between noise removal and the preservation of potentially useful instances. A higher threshold leads to more aggressive noise removal, while a lower threshold retains more instances [17].

Hybrid Reduction Techniques

Hybrid reduction techniques combine the strengths of both condensed and edited approaches to achieve more robust and efficient reduction. The **DROP3** algorithm [**?**] is a notable example of a hybrid technique. Unlike DROP2, which uses CNN first and then ENN, DROP3 reverses this order. It first applies an edited nearest neighbor rule (ENN) to remove noisy or borderline instances, creating a cleaner dataset. Then, it employs a decremental reduction procedure inspired by condensed nearest neighbor to iteratively remove redundant instances that do not affect the classification accuracy of their neighbors. This process results in a significantly reduced dataset while aiming to preserve classification accuracy.

DROP3 [**?**] is a hybrid instance reduction technique that combines the strengths of ENN and Condensed Nearest Neighbor (CNN). It aims to achieve a more substantial reduction in the dataset size while maintaining or improving classification accuracy.

DROP3 operates in two main stages:

1. **Noise Removal:** In the first stage, DROP3 applies ENN to the training set (X, Y) to eliminate noisy instances. This step helps to improve the quality of the data and prepare it for further reduction.

2. **Iterative Reduction:** The second stage involves an iterative process where each remaining instance is evaluated for potential removal. An instance is removed if its removal does not adversely affect the classification accuracy of its neighbors. More specifically, for each instance x_i, DROP3 trains a KNN classifier on the dataset excluding x_i, $(X' \setminus \{x_i\}, Y' \setminus \{y_i\})$. If all instances in $N_k(x_i)$ are correctly classified by this KNN classifier, then x_i is deemed redundant and removed.

This iterative process continues until no further instances can be removed without affecting the classification accuracy of their neighbors. DROP3 effectively identifies and removes redundant instances that do not contribute significantly to the classification performance. By combining noise removal with iterative reduction, DROP3 achieves a more aggressive reduction in the dataset size compared to ENN or CNN alone.

3.3.2 Implementation details

Selection of Instance Reduction Techniques

Our project focuses on evaluating the effectiveness of various instance reduction techniques in improving the efficiency and accuracy of the k-nearest neighbor (KNN) algorithm. We selected three specific algorithms representing different categories of instance reduction: condensed, edited, and hybrid. For the condensed approach, we chose **Generalized Condensed Nearest Neighbor (GCNN)** [**?**] over Reduced Nearest Neighbor (RNN) and Fast Condensed Nearest Neighbor (FCNN). GCNN offers a more robust approach by iteratively adding instances that are misclassified by the current model, ensuring a consistent subset for accurate classification. While RNN and FCNN offer potential speed improvements, GCNN prioritizes finding a minimal consistent set, which aligns better with our goal of maintaining high accuracy.

For the edited approach, we opted for **Edited Nearest Neighbor with Thresholding (ENNTh)** [**?**] instead of Repeated Edited Nearest Neighbor (RENN). ENNTh provides a more controlled noise removal mechanism by incorporating a threshold to determine the probability of an instance belonging to its assigned class. This allows for a more flexible approach compared to RENN, which repeatedly applies ENN until no further instances are removed. This repeated application can be computationally expensive and may potentially remove valuable instances. Finally, for the hybrid approach, we selected **Decremental Reduction Optimization Procedure 3 (DROP3)** [**?**] over Instance-Based 2 (IB2) and DROP2. DROP3 combines the strengths of ENN and CNN by first applying ENN to remove noise and then using a decremental reduction procedure to eliminate redundant instances. This approach offers a more refined reduction compared to IB2, which primarily focuses on adding instances, and DROP2, which uses a different order of operations that may not be as effective in removing both noise and redundancy.

Application of Instance Reduction Techniques

Following the initial analysis to determine the optimal k-nearest neighbor (KNN) parameters for each dataset (**i.e.**, mushroom and hepatitis), we applied the selected instance reduction techniques (**i.e.**, GCNN, ENNTh, and DROP3) as a preprocessing step. Using the top-performing KNN model configuration, we generated reduced versions of the training data. These reduced datasets, along with the original unreduced data, were then used to evaluate the performance of the optimized KNN classifier. This allowed us to assess the impact of each instance reduction technique on KNN classification accuracy, efficiency and storage. Furthermore, we extended our analysis to evaluate the influence of the reduced training sets on the performance of our optimized SVM classifier, provided insights into the effectiveness of instance reduction techniques across eager learning algorithms.

3.4 Statistical Analysis

This section presents the statistical analysis of the results of the study. We employ a systematic approach to compare the performance of different models and their configurations.

Tests Considered

There are various statistical tests available to compare the performance of machine learning models. We considered the following tests for our analysis:

3.4.1 ANOVA

Analysis of Variance (ANOVA) decomposes the total variance in the data into different components such as: between-classifier variability, between-dataset variability, and residual variability. When between-classifier variability is significantly larger than the residual variability, we can reject the theorised null hypothesis and conclude that there is a significant difference between the classifiers.

However, ANOVA assumes that the data is drawn from normal distributions and that the variances are equal across groups. In the case of the hepatitis dataset, the class distribution is skewed ($79 - 21$ split) [7, 21].

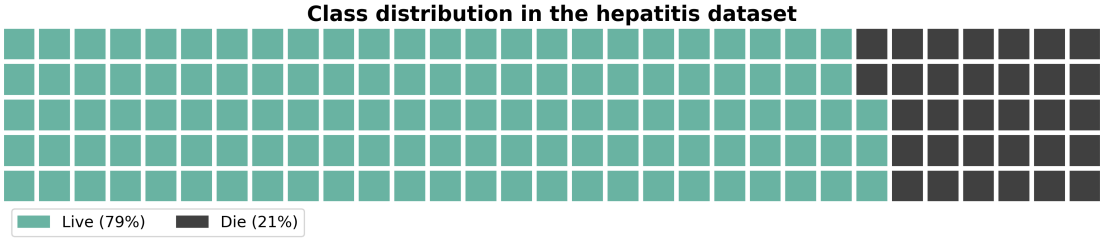

Figure 3: Class distribution of the Hepatitis dataset

Furthermore, the performance metrics distribution across folds in Figure 4 further demonstrates non-normal distribution:

These plots verify that the data does not meet the assumptions of ANOVA:

- Non-normal distribution: Several models show asymmetric distributions, where the median line is not in the middle of the box. Also present are outliers in the data, which are represented as dots outside of the whisker lines. - Unequal variances: Box sizes vary significantly across models, and some models have much larger spreads than others. Additionally, the whisker lenghts vary considerably between models.

These claims hold true when analysing performance metrics distribution across folds using SVM:

Given these observations, we decided against using ANOVA for our analysis.

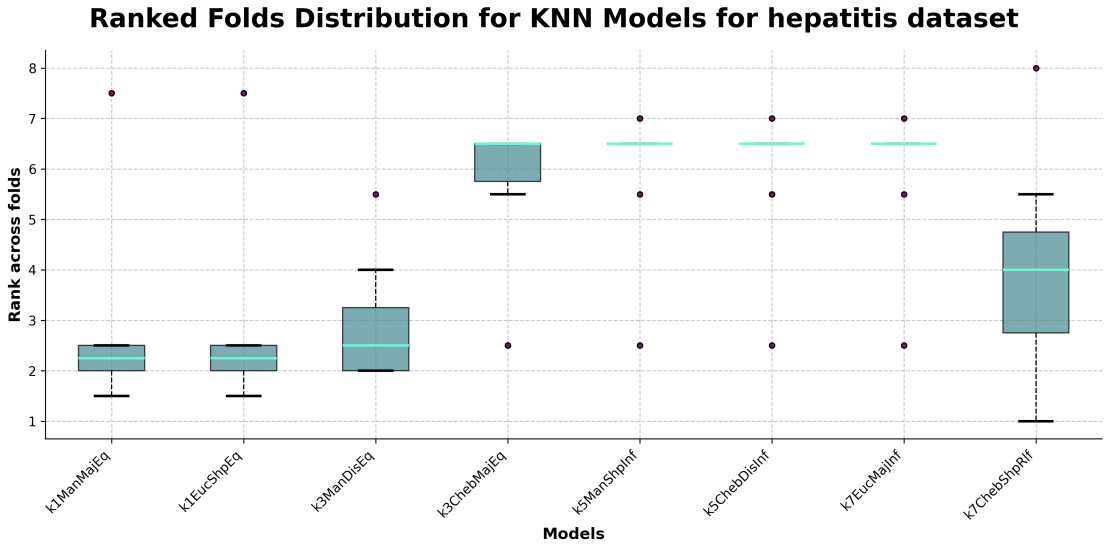

Figure 4: Rank distributions across folds for different KNN configurations on the Hepatitis dataset

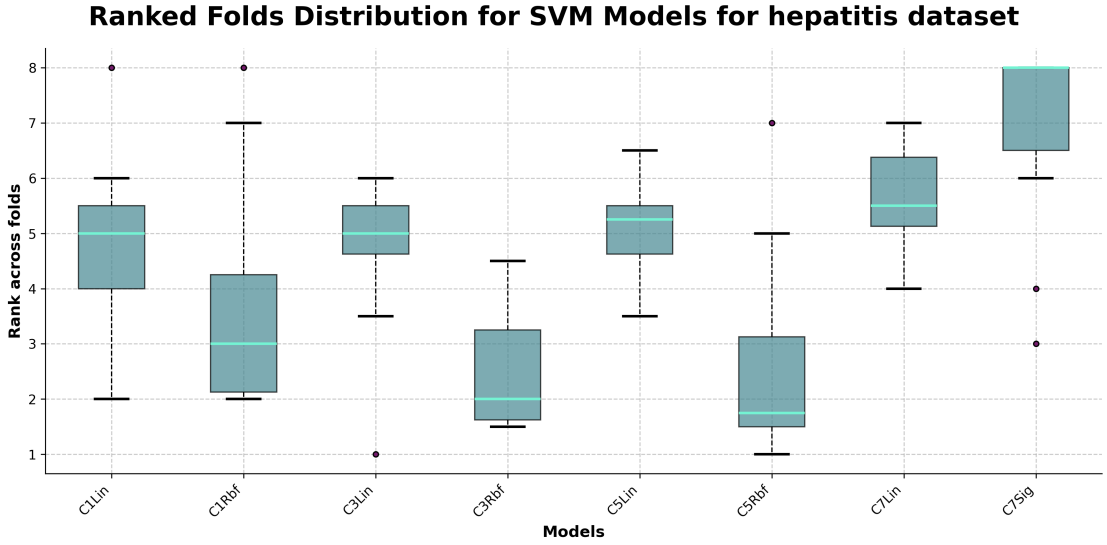

Figure 5: Rank distributions across folds for different KNN configurations on the Hepatitis dataset

3.4.2 Friedman Test

The Friedman test is a non-parametric equivalent of the repeated-measures ANOVA. For each dataset, it ranks the models separately with the best model receiving a rank of 1, the second-best a rank of 2, and so on. The average rank is used to settle any ties in the ranks [21, 9].

While the Friedman test theoretically has less statistical power than ANOVA when ANOVA's assumptions are met, it is more suitable for our analysis as it makes no assumptions about normality or equal variances. As we demonstrated with the hepatitis dataset's class distribution and performance metrics, these assumptions do not hold in our case.

3.4.3 Post-Hoc Testing

The Friedman test only tells us that there is a significant difference between the models, but it does not tell us which models are significantly different from each other. This is why we employ the Nemenyi test as a post-hoc procedure.

The Nemenyi test compares the performance of all classifiers to each other by checking if their average ranks differ by at least the critical difference:

$$CD = q_\alpha \times \sqrt{\frac{k(k+1)}{6N}} \tag{3}$$

where q_α is based on the Studentized range statistic, k is the number of classifiers, and N is the number of datasets [9]. If the difference in average ranks between two models exceeds this critical difference, we can conclude that their performances are significantly different.

When performing the Nemenyi test, proper handling of ties is crucial for accurate analysis. Instead of arbitrarily assigning consecutive ranks to tied values (e.g., ranks 2 and 3), we use the average of the ranks they would have occupied. For example, if two models tie for second place, rather than arbitrarily assigning ranks 2 and 3, both models receive a rank of 2.5 (the average of positions 2 and 3). This mid-rank approach ensures fair treatment of tied performances and prevents artificial rank differences that could bias the statistical analysis.

In the visualisation of critical difference (CD) diagrams, the entire bar represents the critical difference value. The half-width of the bar (CD/2) extends on either side of a model's average rank. When comparing two models, if their CD/2 intervals do not overlap, we can conclude that there is a statistically significant difference in their performance. This visual representation provides an intuitive way to interpret the Nemenyi test results, as any non-overlapping bars clearly indicate significant differences between models.

3.4.4 Linear vs. Top Sampling Justification

The results of the Nemenyi test must satisfy the requirements to perform post-hoc testing. If the Nemenyi test fails, we cannot proceed with post-hoc testing as the results would be considered unreliable.

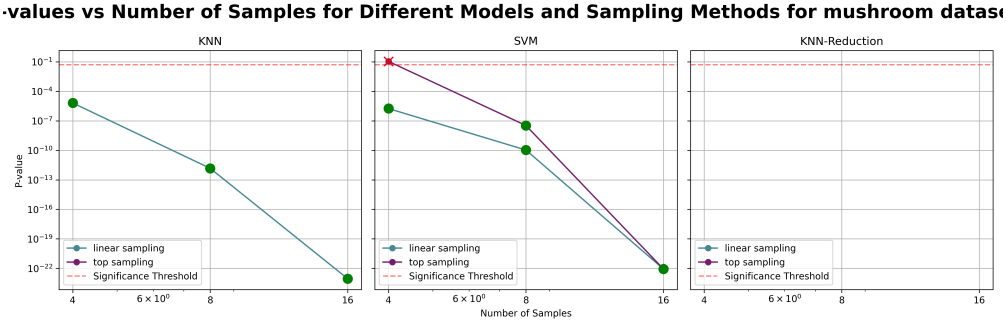

Figure 6: P values vs. number of samples for different models and sampling methods for the datasets

As seen in the plots in the above figures, we perform linear sampling and top sampling with a variety of sample sizes. With top sampling, we sample the top models based on their f1 score, while linear sampling samples models evenly across the performance spectrum. The results show that the Nemenyi test fails across the various sample sizes for both datasets when using top sampling but passes when using linear sampling.This indicates that the top sampling method is not suitable for our analysis as we cannot perform post-hoc testing on the models.

To further back up our decision, below are tables of the Friedman test results for both datasets.

These tables display the p-values for the Friedman test across various combinations of sampling methods, model types, and scoring metric. The same information from the figures above is presented in tabular form, along with additional results.

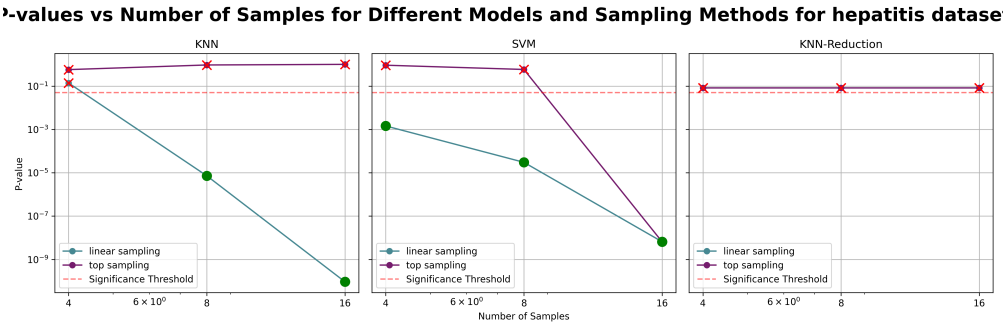

Figure 7: P values vs. number of samples for different models and sampling methods for the datasets

Table 1: Friedman Test Results Mushroom

name	P Value
KNN Top Sample of 8 F1 Scores	1
SVM Top Sample of 8 F1 Scores	3.331542319440828e-08
KNN Linear Sample of 8 F1 Scores	1.477429493282523e-12
SVM Linear Sample of 8 F1 Scores	1.068518136588972e-10
KNN-Reduction Linear Sample of 8 F1 Scores	1
SVM-Reduction Linear Sample of 8 F1 Scores	1.3800570312932555e-06
KNN-Reduction Linear Sample of 8 Training Times	0.003993295308193392
SVM-Reduction Linear Sample of 8 Training Times	7.459327906944864e-05

From the tables above the same conclusion can be drawn from the Friedman test results. From the tabular form, we can infer which group of models passed the Friedman test, and which did not.

Employing linear sampling, we can see that the Friedman test passes on the majority of cases for both datasets. From our analysis, using linear sampling failed the Friedman test only once for the mushroom dataset (using KNN-Reduction and F1 scoring metric) and also once for the hepatitis dataset (using KNN-Reduction and F1 scoring metric also). Employing top sampling, the Friedman test failed for all but one case on both datasets (using SVM and F1 scoring metric).

Therefore, we proceeded with the Nemenyi test using linear sampling for our analysis to ensure the reliability of the post-hoc testing results.

Table 2: Friedman Test Results Hepatitis

name	P Value
KNN Top Sample of 8 F1 Scores	0.9363612898350686
SVM Top Sample of 8 F1 Scores	0.5813644363775075
KNN Linear Sample of 8 F1 Scores	7.258355849331829e-06
SVM Linear Sample of 8 F1 Scores	3.0272403715178655e-05
KNN-Reduction Linear Sample of 8 F1 Scores	0.08210005961379935
SVM-Reduction Linear Sample of 8 F1 Scores	0.0449364011678129
KNN-Reduction Linear Sample of 8 Training Times	0.00037357398316662454
SVM-Reduction Linear Sample of 8 Training Times	0.00023475060424000085

Table 3: Results From Knn Models For The Mushroom Dataset

	k	distance func	voting func	weighting func	mean f1	mean train time	mean test
1	1	ManhattanDistance	MajorityClassVote	EqualWeighting	1.000	0.000	15
2	3	EuclideanDistance	InverseDistanceWeightedVote	EqualWeighting	1.000	0.000	17
3	1	EuclideanDistance	MajorityClassVote	EqualWeighting	1.000	0.000	17
4	1	ManhattanDistance	InverseDistanceWeightedVote	EqualWeighting	1.000	0.000	15
5	1	ManhattanDistance	ShepardsWorkVote	EqualWeighting	1.000	0.000	14
6	1	EuclideanDistance	InverseDistanceWeightedVote	EqualWeighting	1.000	0.000	17
7	3	ManhattanDistance	MajorityClassVote	EqualWeighting	1.000	0.000	14
8	3	ManhattanDistance	InverseDistanceWeightedVote	EqualWeighting	1.000	0.000	15
9	3	ManhattanDistance	ShepardsWorkVote	EqualWeighting	1.000	0.000	15
10	3	EuclideanDistance	MajorityClassVote	EqualWeighting	1.000	0.000	18

Table 4: Results From Knn Models For The Hepatitis Dataset

	k	distance func	voting func	weighting func	mean f1	mean train time	mean test
1	1	ChebyshevDistance	ShepardsWorkVote	EqualWeighting	0.973	0.000	
2	1	ChebyshevDistance	MajorityClassVote	EqualWeighting	0.973	0.000	
3	1	ChebyshevDistance	InverseDistanceWeightedVote	EqualWeighting	0.973	0.000	
4	5	EuclideanDistance	ShepardsWorkVote	EqualWeighting	0.969	0.000	
5	7	EuclideanDistance	ShepardsWorkVote	EqualWeighting	0.969	0.000	
6	3	ManhattanDistance	ShepardsWorkVote	EqualWeighting	0.969	0.000	
7	1	ManhattanDistance	MajorityClassVote	EqualWeighting	0.969	0.000	
8	1	EuclideanDistance	ShepardsWorkVote	ReliefFWeighting	0.969	0.000	
9	1	ManhattanDistance	ShepardsWorkVote	EqualWeighting	0.969	0.000	
10	1	EuclideanDistance	MajorityClassVote	EqualWeighting	0.969	0.000	

4 Results and Analysis

4.1 Results

Here, present the findings of the study. Include relevant data, statistics, and any figures or tables that help illustrate the results.

4.2 Discussion

In this section, interpret the results, discuss their implications, and relate them back to the research question or hypothesis. Address any limitations of the study and suggest areas for future research.

4.2.1 k-Nearest Neighbors (KNN) Analysis

This section interprets the results of the KNN algorithm, discussing its performance and implications.

Table 5: Results From Knn Models For The Mushroom Dataset With Dimensionality Reduction

	reduction func	k	distance func	voting func	weighting func	mean f1	mean train time	mean
1	control	1	ManhattanDistance	MajorityClassVote	EqualWeighting	1.000	0.000	
2	GGCN	1	ManhattanDistance	MajorityClassVote	EqualWeighting	1.000	0.000	
3	ENNTH	1	ManhattanDistance	MajorityClassVote	EqualWeighting	1.000	0.000	
4	Drop3	1	ManhattanDistance	MajorityClassVote	EqualWeighting	1.000	0.000	

Table 6: Results From Knn Models For The Hepatitis Dataset With Dimensionality Reduction

	reduction func	k	distance func	voting func	weighting func	mean f1	mean train time	mean
1	control	1	ManhattanDistance	MajorityClassVote	EqualWeighting	0.969	0.000	
2	ENNTH	1	ManhattanDistance	MajorityClassVote	EqualWeighting	0.969	0.000	
3	GGCN	1	ManhattanDistance	MajorityClassVote	EqualWeighting	0.961	0.000	
4	Drop3	1	ManhattanDistance	MajorityClassVote	EqualWeighting	0.916	0.000	

The KNN algorithm was evaluated using several performance metrics, including accuracy, precision, and recall and the f1 score.

General Performance Analysis

As seen in Tables 1 and 2, the general performance of the KNN algorithm was excellent across both datasets, achieving accuracy scores above 92% and F1 scores above 92.5% on all top ten configurations.

4.2.2 Mushroom Dataset Performance

Best Configuration:

- $k = 3, 5$, or 7 with Manhattan Distance (all performed equally well)
- Shepard's Work Voting scheme
- Equal Weighting
- **Results:** 95.1% accuracy and 95.2% F1 score

Performance Range:

- Accuracy: 92.6% −− 95.1%
- F1 Score: 92.7% −− 95.2%

Observations

Manhattan Distance consistently performed well across all configurations, with all top ten configurations using this distance metric. The combination of Manhattan Distance and Shepard's Work Vote with a higher value of k was particularly effective. The higher performance of a larger k value suggests that the algorithm benefits from considering multiple neighbors when making predictions, perhaps due to the dataset's inherent complexity and noise present in the data. The equal weighting scheme also performed excellently, indicating that the dataset contains relatively a uniform distribution of meaningful data points. The F1 scores achieved across all configurations were consistently high, indicating a good balance between precision and recall.

The above figure shows the ranked folds for the k-NN algorithm on the mushroom dataset. The figure illustrates the variation in performance across different KNN models, with some models performing better than others. It's evident a lower value of k (1 or 3) tends to perform better than higher values of k (5 or 7) in combination with the other parameters. Interestingly, for each KNN model the performance across the 10 folds is identical.

4.2.3 Hepatitis Dataset Performance

Best Configuration:

- $k = 1$ with Euclidean Distance
- Shepard's Work Vote
- ReliefF Weighting
- **Results:** 95.5% accuracy and 97.2% F1 score

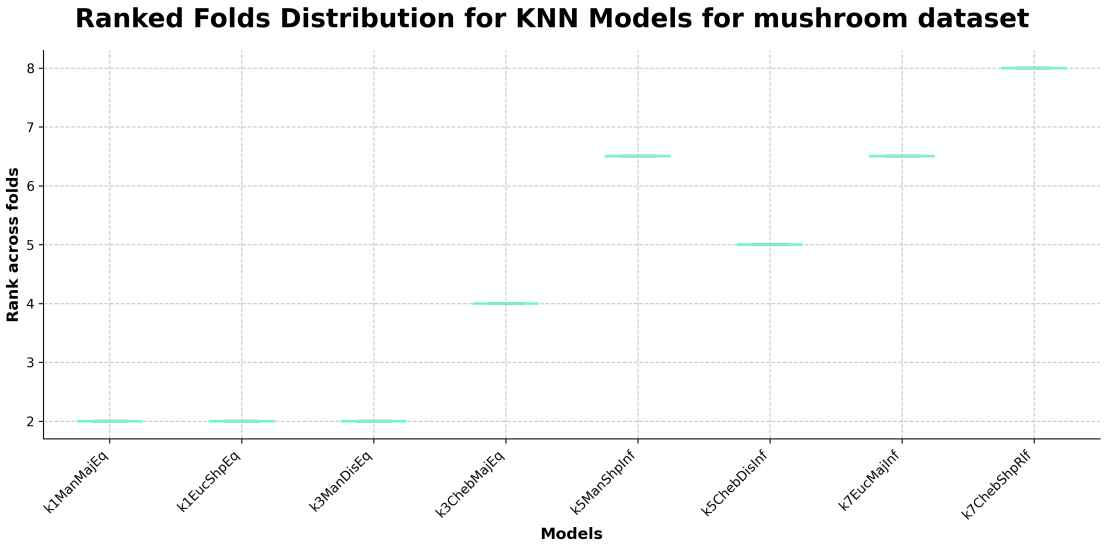

Figure 8: KNN Ranked Folds for Mushroom Dataset

Performance Range:

- Accuracy: 94.8% –– 95.5%
- F1 Score: 96.7% –– 97.2%

Key Observations:

- Euclidean Distance performed best, particularly with ReliefF weighting.
- k = 1 configurations dominated the top results.
- All voting schemes performed similarly when other parameters were optimized.

Observations

The hepatitis dataset achieved its best performance with a k value of 1, indicating that the algorithm benefits from more localized decisions when making predictions when using this dataset. The Euclidean Distance metric performed best, particularly when combined with ReliefF weighting. This suggests that the dataset's features have varying degrees of importance, and ReliefF weighting helps capture these distinctions effectively. As also seen in the mushroom dataset, the F1 scores achieved across all configurations were consistently high, indicating a good balance between precision and recall.

The above figure shows the ranked folds for the k-NN algorithm on the hepatitis dataset. The figure illustrates the variation in performance across different KNN models, with some models outperforming others. Unlike the mushroom dataset, the hepatitis dataset shows variation in performance in each model per fold. A number of the models showed consistent mean performance but also contained some outliers that performed significantly better or worse than the mean.

4.2.4 Overall Comparison

While both datasets achieved high performance, the hepatitis dataset outperformed the mushroom dataset in terms of accuracy and F1 score. This difference may be due to the hepatitis dataset's smaller size and more distinct class separations, making it easier for the KNN algorithm to make accurate predictions. These distinct class separations may also explain why the k = 1 configuration performed best on the hepatitis dataset, as the algorithm can make more precise decisions with fewer neighbors.

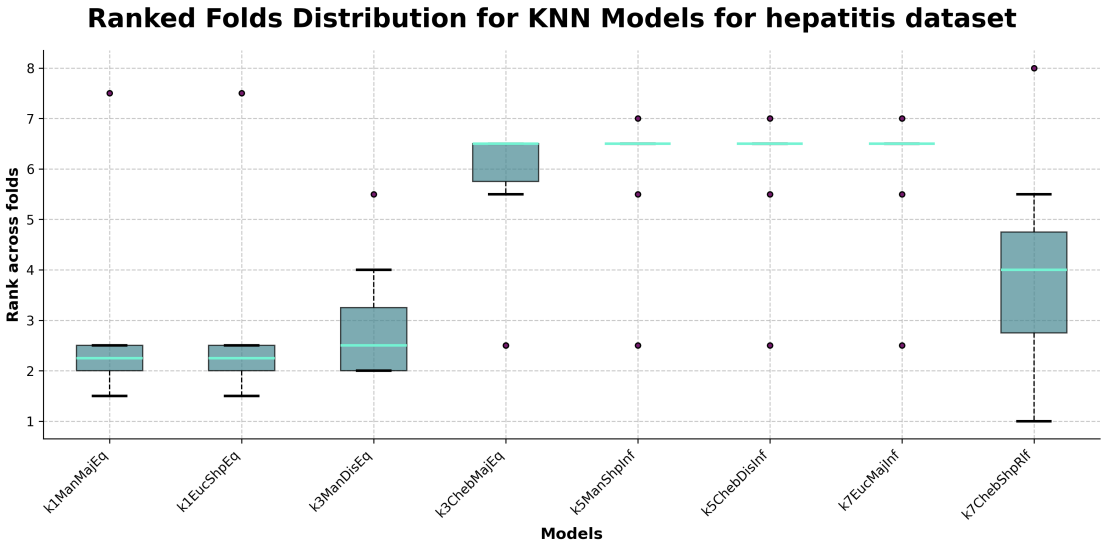

Figure 9: KNN Ranked Folds for Hepatitis Dataset

K-Nearest Neighbour (KNN) Statistical Analysis

Below are the Nemenyi test results for the KNN models on the hepatitis and mushroom datasets respectively.

For the below analysis of both the KNN model, values closer to 1 indicate no significant difference between models, whereas values closer to 0 indicate a significant difference.

The diagonal values (top left to bottom right) are always 1 and report no significant difference as they represent the comparison of a model with itself.

To determine statistical significance, we use a confidence level of 0.95 (95% confidence interval) for the Nemenyi test. Therefore, any values greater than 0.05 indicate that the models are not significantly different from each other.

The Nemenyi test results for the KNN models on the hepatitis dataset reveal two statistically distinct groups of model configurations. The first group consists of k = 1 configurations (with Manhattan and Euclidean distance) and k=3 with Manhattan distance, which all perform similarly to each other (p=1.00). The second group includes configurations with higher k values (3,5,7) using various distance metrics, which also perform similarly within their group (p=1.00) but significantly differently from the first group (some comparisons yield p=0.04, below the 0.05 threshold). This clear separation suggests that the choice of k-value has a more substantial impact on model performance than the choice of distance metric, with k=1 configurations behaving distinctly from higher k-values.

The Nemenyi test results for the KNN models on the mushroom dataset reveal two main groups of model configurations. The first group consists of k=1 configurations and k=3 with Manhattan distance, which perform similarly (p=0.85 − 1.00), though with some variation within the group. The second group includes configurations from k5ManShpInf through k7ChebShpRtf, which show high similarity within their group (p=0.95 − 1.00). Some of these groups perform significantly differently from each other, given the p-values fall below the 0.05 threshold.

We can deduce that the k-value has a substantial impact on model performance on both datasets.

Parameter Analysis

To further understand the impact of the parameters on the performance of the KNN algorithm, we analyzed the impact of the hyperparameters on the performance of the KNN algorithm, both independently and in combination.

We can see from Figure 12 that each parameter plays a significant role in the performance of the KNN algorithm. We also see some notable interactions between parameters, such as majority class voting perform-

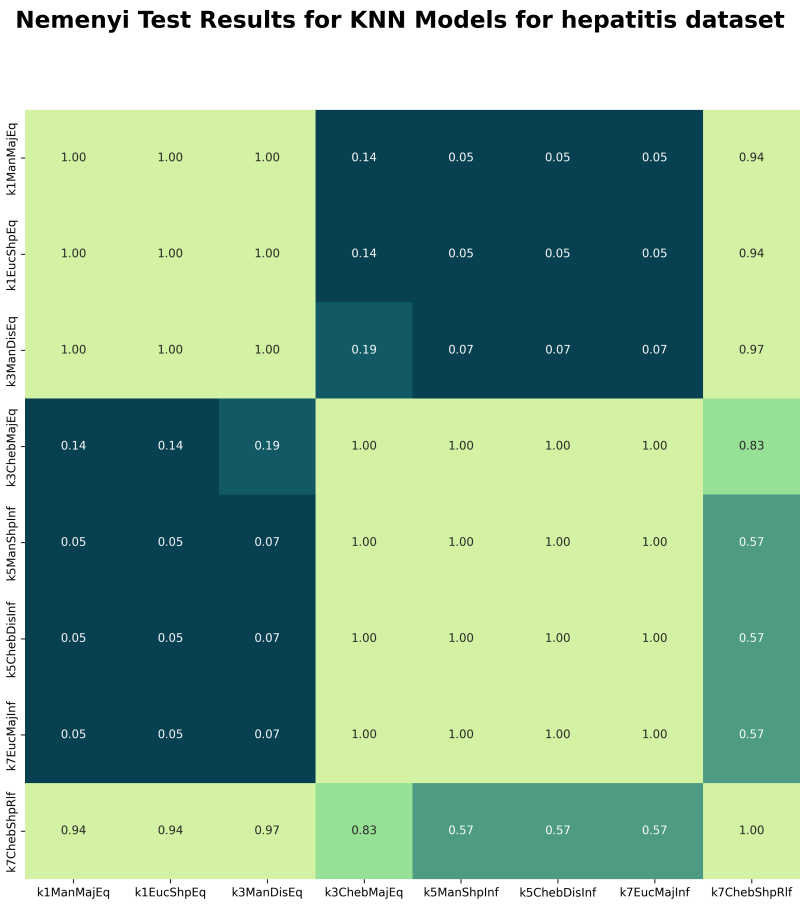

Figure 10: Nemenyi test results for KNN models on the hepatitis dataset

ing better when combined with low k values and the Manhattan distance metric. We also see that Inverse Distance Weighting performs poorly in all cases except with k=1, where it there is difference at all (since voting scheme has no impact when k=1).

4.2.5 Support Vector Machines (SVM) Analysis

A key advantage of the SVM over KNN is that SVMs are much faster during consultation time. As seen in Figure 13, SVMs tend to have much lower test times than KNN. This is especially noticable for large datasets like the mushroom dataset.

4.2.6 Mushroom Dataset Performance

Best Configuration:

- C = 1 with Polynomial Kernel

Performance Range:

- Accuracy: 92.8%
- F1 Score: 92.8%

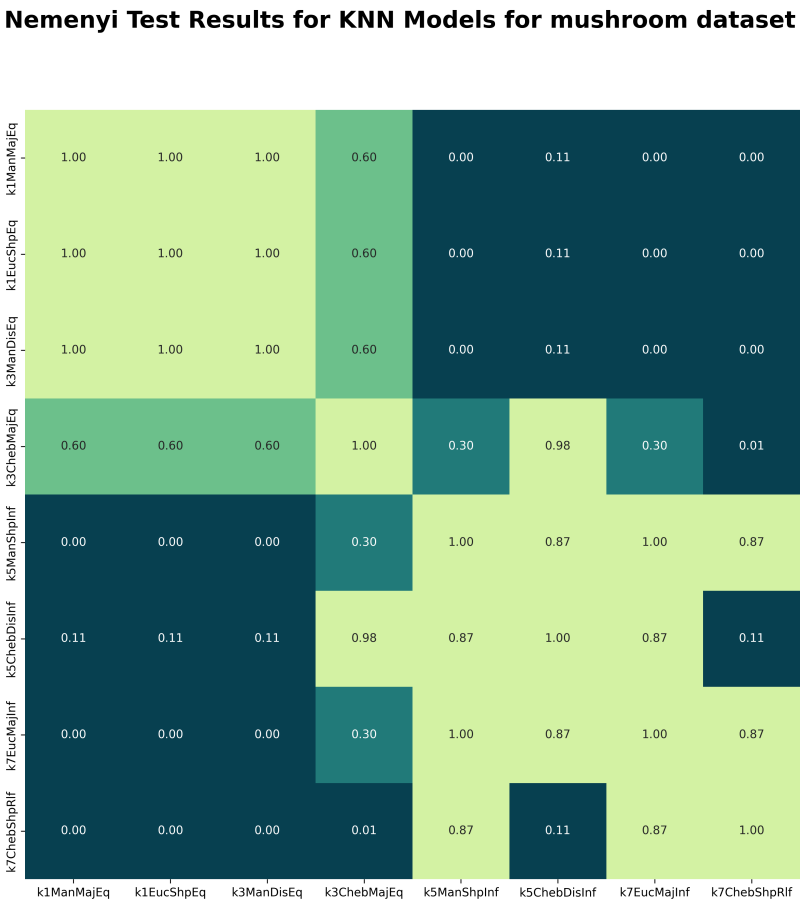

Figure 11: Nemenyi test results for KNN models on the mushroom dataset

Observations

The results for the mushroom dataset show that the SVM model performed consistently strong across all configurations, albeit some performed better than others. The best performing model used a Polynomial kernel with $C = 1$. (See Table 7). This SVM model achieved a score of 92.8% for both accuracy and F1 score. This result was closely following by the RBF kernel with higher values of C (3, 5, and 7).

The above figure shows the ranked folds for the SVM algorithm on the mushroom dataset. The figure illustrates the variation in performance across different SVM models, with some models performing better than others. Notably, the presence of outliers (shown as points) in several models indicates occasional deviations from the mean performance. The median ranks vary considerably across the different models, with only 'C5Lin' and 'C7Lin' showing consistent performance across all folds.

4.2.7 Hepatitis Dataset Performance

Best Configuration:

- C = 7 with RBF Kernel

Performance Range:

- Accuracy: 95.5%

Table 7: Results From Svm Models For The Mushroom Dataset

	C	kernel type	mean f1	mean train time	mean test time
1	3	poly	1.000	0.116	0.004
2	5	poly	1.000	0.115	0.003
3	7	poly	1.000	0.117	0.003
4	5	rbf	1.000	0.101	0.014
5	7	rbf	1.000	0.093	0.012
6	3	rbf	0.999	0.121	0.018
7	1	poly	0.999	0.129	0.008
8	1	rbf	0.989	0.178	0.030
9	7	linear	0.983	1.623	0.015
10	5	linear	0.983	1.185	0.013

Table 8: Results From Svm Models For The Hepatitis Dataset

	C	kernel type	mean f1	mean train time	mean test time
1	7	rbf	0.975	0.001	0.000
2	5	rbf	0.971	0.001	0.000
3	3	poly	0.970	0.001	0.000
4	1	poly	0.970	0.001	0.000
5	3	rbf	0.969	0.001	0.000
6	5	poly	0.969	0.001	0.000
7	7	poly	0.969	0.001	0.000
8	1	rbf	0.942	0.001	0.000
9	3	linear	0.929	0.001	0.000
10	1	linear	0.927	0.001	0.000

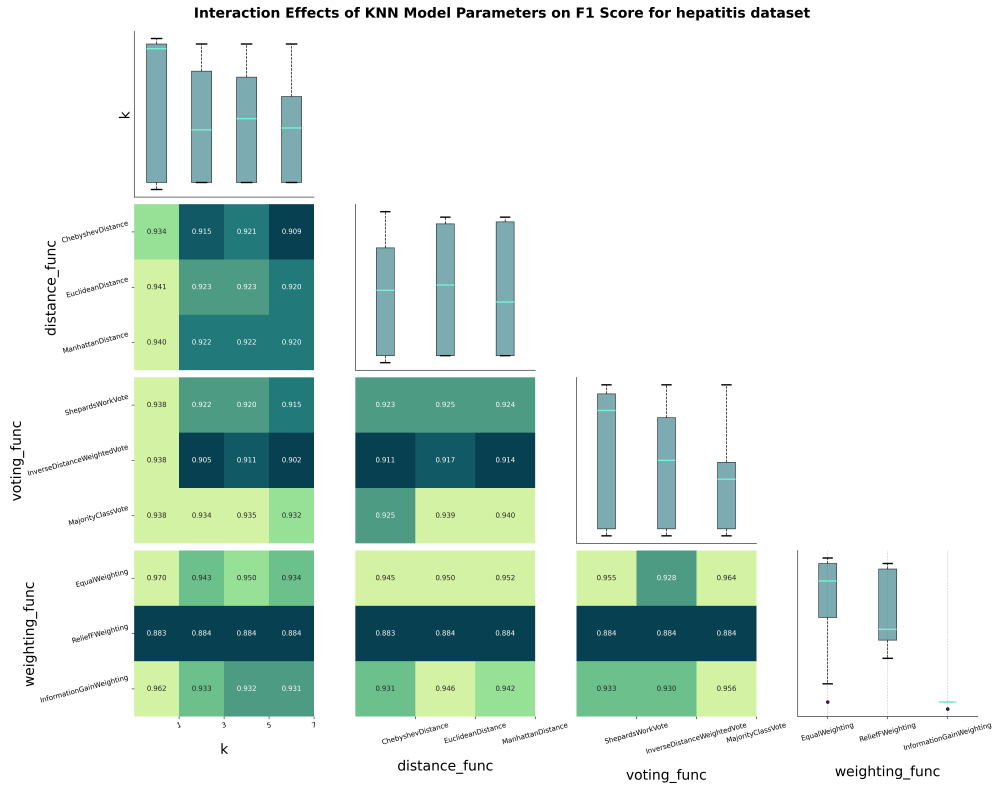

Figure 12: Interaction effects for KNN models on the hepatitis dataset

- F1 Score: 97.2%

Observations

The results for the hepatitis dataset show that the SVM model also performed consistently strong across all configurations. (See Table 8). The best performing model used an RBF kernel with $C = 7$. This SVM model achieved an accuracy of 95.5% and an F1 score of 97.2%.

The above figure shows the ranked folds for the SVM algorithm on the mushroom dataset. The figure illustrates the variation in performance across the different SVM models and folds. When considering the mean, SVM models 'C3RBF' and 'C5RBF' consistently outperformed the other models.

Overall Comparison

For the hepatitis dataset, the configuration using the **RBF** kernel and $C = 7$ achieved outstanding accuracy and F1 scores. However, because of the simple predictability of the mushroom data, the simple **Polynomial** kernel performed better when paired with $C = 1$. (See Table 8 and Table 7).

Support Vector Machines (SVM) Statistical Analysis

To compare the performance across model configurations, we employed statistical analysis methods (see subsection 3.4) to determine whether the various configurations showed statistically significant differences in performance.

As seen in Figure 15 and Figure 14, some models achieved significantly better results than others. By comparing the ranks among each of the 10 folds, we can check to see if there were any cases in which 1 model always outperformed another.

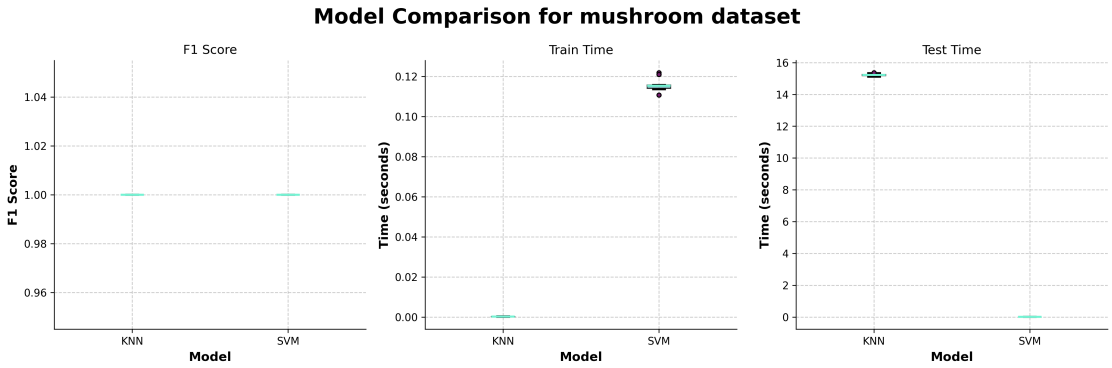

Figure 13: SVM and KNN model comparison on Mushroom Dataset

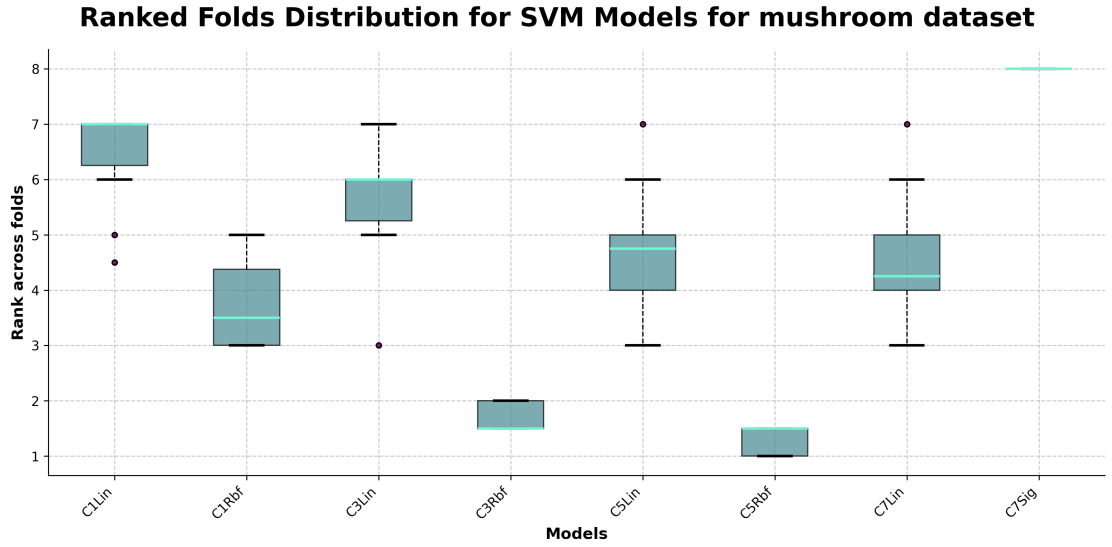

Figure 14: SVM Ranked Folds for Mushroom Dataset

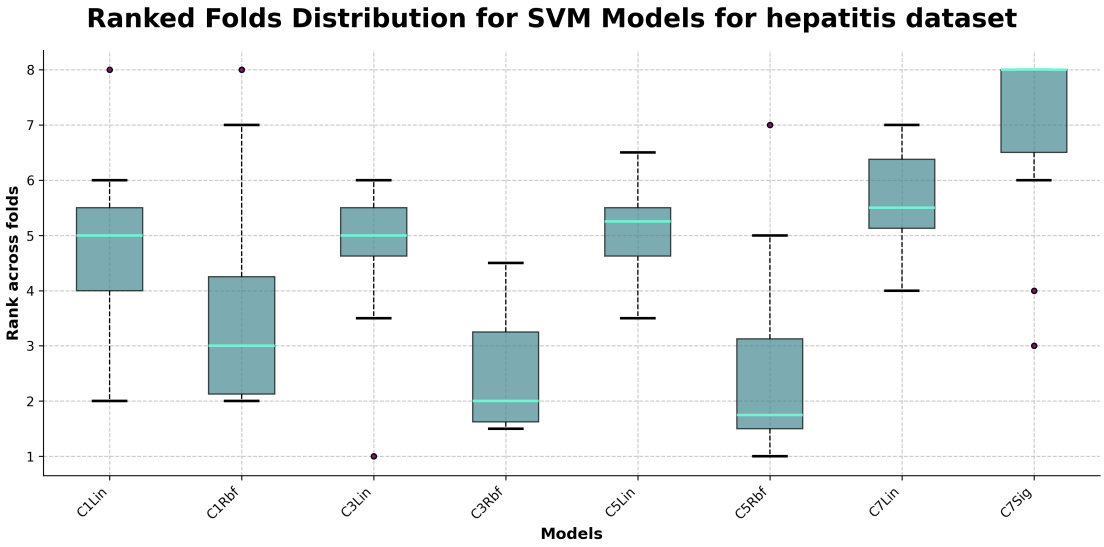

Figure 15: SVM Ranked Folds for Hepatitis Dataset

The mushroom dataset has some especially obvious cases, in which the $C = 3$ and $C = 5$ RBF models outperformed all other models across every fold. The $C = 5$ RBF model was the best performing model overall.

The hepatitis dataset does not have as extreme of a case, but there are still some models that stand out. The $C = 7$ Sigmoid model was consistently worse than the $C = 3$ and $C = 5$ RBF models.

To identify significant pairs at a glance, we performed the Nemenyi test on all model pairs, and present them in heatmaps (see Figure 16 and Figure 17). The cells represent the p-values for each pair of models. A value of 0.05 or lower indicates that the models are statistically different at the 95% confidence level. The full tables containing the p-values for each pair of significant models are provided in Table 9 and Table 10.

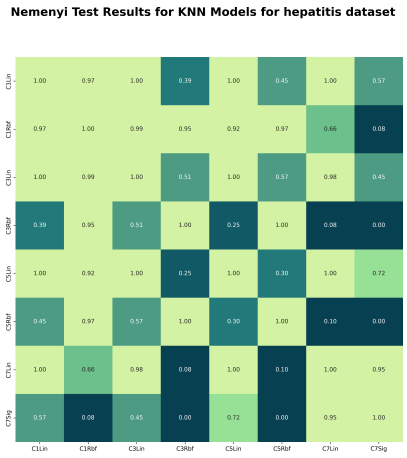

Figure 16: Nemenyi Test Results for SVM on Hepatitis Dataset

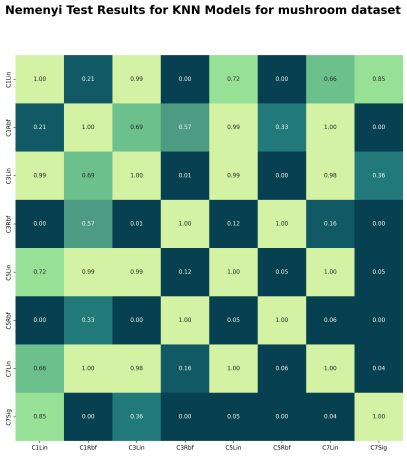

Figure 17: Nemenyi Test Results for SVM on Mushroom Dataset

Table 9: Significant Differences in SVM Models

C	Kernel Type	Mean F1
3	rbf	0.969
5	rbf	0.971
7	sigmoid	0.829

Table 10: Significant Differences in SVM Models

C	Kernel Type	Mean F1
1	linear	0.975
1	rbf	0.989
3	linear	0.980
3	rbf	0.999
5	linear	0.983
5	rbf	1.000
7	linear	0.983
7	sigmoid	0.436

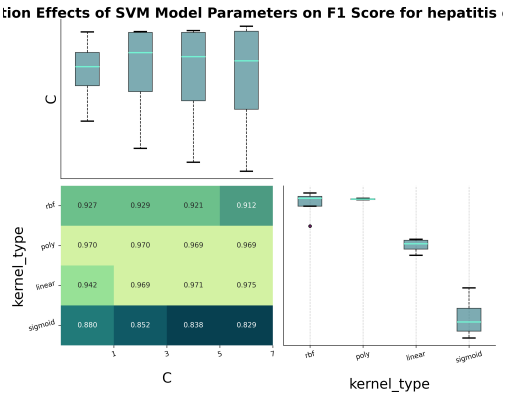

Figure 18: Interaction Effects for SVM on Hepatitis Dataset

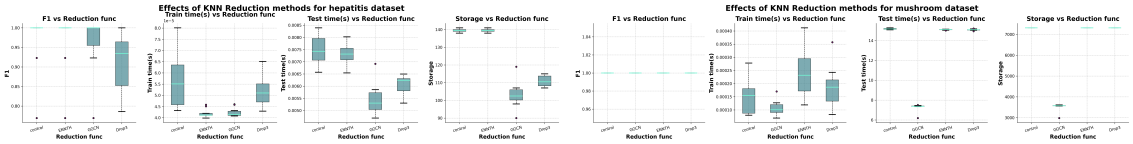

Figure 19: Class distributions

To further analyze the hyperparameters we plotted the effects of each hyperparameter on the F1 score.

Here we see that the Kernel Type is a stronger predictor of performance than the C parameter, and that the best performing models tend to use the poly or linear kernals, and that higher values of C tend to perform worse.

4.2.8 Instance Reduction Analysis

This section examines the impact of three reduction techniques—GGCN, ENNTH, and Drop3—on the performance, training time, testing time, and storage requirements of both the KNN and SVM algorithms.

In our analysis of the Hepatitis dataset, we observed that applying reduction techniques for K-Nearest Neighbors (KNN) resulted in F1 scores similar to those obtained without any reduction. However, the Drop3 method saw a slight decrease in mean F1 score from 0.948 to 0.864.

In terms of training time, all reduction approaches exhibited a consistent similar duration compared to not applying reduction.

Regarding storage efficiency, the ENNTH method did not yield any reductions, while both GCNN and Drop3 achieved approximately 27% and 20% reductions in storage, respectively. We hypothesize that ENNTH, which focuses on removing noisy instances, did not find any significant noise in the Hepatitis dataset, which is relatively clean.

For the Hepatitis dataset, the F1 score remained at 1.0 across all methods. Notably, only GCNN succeeded in reducing storage, whereas ENNTH and Drop3 did not provide any reductions. This is likely because Drop3 not only removes noisy instances but also attempts to eliminate duplicates. Its effectiveness in reducing storage for Hepatitis, but not for the Mushroom dataset, suggests that there are fewer duplicates present in the Mushroom dataset. Similarly, ENNTH likely failed to achieve any reductions due to the lack of noise in the data.

In contrast, GCNN demonstrated a substantial 50% reduction in storage, with a one-third decrease in training time and a halving of testing time. It is crucial to note that GCNN employs an incremental approach, unlike ENNTH and Drop3, which are decremental methods aimed at removing non-beneficial instances. This incremental approach appears to be more effective in this context.

Similarly, applying the SVM model to the reduced Hepatitis dataset yielded comparable results in terms of F1 scores, with all accuracies remaining similar; however, the Drop3 method did result in a slight decline in accuracy.

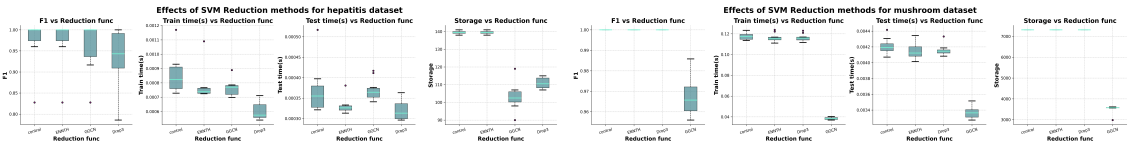

Figure 20: Class distributions

Additionally, the training and testing times were largely consistent with those observed when no reduction was applied, which can be attributed to the relatively small size of the dataset.

In contrast, the larger Mushroom dataset demonstrated different outcomes.

Here, GCNN significantly reduced both training and testing times.

Specifically, while using KNN with GCNN reduction achieved a one-third reduction in training time, SVM with GCNN reduction realized an impressive approximate 70% decrease.

For testing times, KNN with GCNN reductio saw a 50% reduction, whereas in the case of SVM with GCNN reduction, the reduction was closer to one-third.

4.3 Summary

5 Conclusion

Summarize the main findings of the study and their significance. Restate the key points and provide a final perspective on the research.

References

[1] Jason Brownlee. Information gain and mutual information for machine learning, 2019.

[2] Christopher J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[3] Xu Chu, Ihab F Ilyas, Sanjay Krishnan, and Jiannan Wang. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 international conference on management of data*, pages 2201–2206, 2016.

[4] University College Cork. Lecture notes - classification.

[5] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[6] Padraig Cunningham and Sarah Jane Delany. k-nearest neighbour classifiers - a tutorial. *ACM Computing Surveys*, 54(6):128:1–128:25, 2021.

[7] Janez Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[8] David L Donoho et al. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS math challenges lecture*, 1(2000):32, 2000.

[9] Anabela Afonso Dulce G. Pereira and Fátima Melo Medeiros. Overview of friedman's test and post-hoc analysis. *Communications in Statistics - Simulation and Computation*, 44(10):2636–2653, 2015.

[10] Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

[11] Li-Yu Hu, Min-Wei Huang, Shih-Wen Ke, and Chih-Fong Tsai. The distance function effect on k-nearest neighbor classification for medical datasets. *SpringerPlus*, 5(1):1304, 2016.

[12] JD Hunter. Matplotlib: A 2d graphics environment. com-464 puting in science & engineering, 9 (3), 90–95, 2007.

[13] IBM. What is the k-nearest neighbors algorithm, 2023.

[14] Trevor LaViale. Deep dive on knn: Understanding and implementing the k-nearest neighbors algorithm, 2023.

[15] Wes McKinney et al. Data structures for statistical computing in python. In *SciPy*, volume 445, pages 51–56, 2010.

[16] Y. Song, X. Kong, and C. Zhang. A large-scale k-nearest neighbor classification algorithm based on neighbor relationship preservation. *Wireless Communications and Mobile Computing*, 2022:1–11, 2022.

[17] Fernando Vázquez, J. Salvador Sánchez, and Filiberto Pla. A stochastic approach to wilson's editing algorithm. In Jorge S. Marques, Nicolás Pérez de la Blanca, and Pedro Pina, editors, *Pattern Recognition and Image Analysis*, pages 35–42, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[18] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.

[19] Michael L Waskom. Seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.

[20] D. Randall Wilson and Tony R. Martinez. Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38(3):257–286, 2000.

[21] Donald W. Zimmerman and Bruno D. Zumbo. Relative power of the wilcoxon test, the friedman test, and repeated-measures anova on ranks. *The Journal of Experimental Education*, 62(1):75–86, 1993.

Table 11: KNN Legend

Model Label	K	Distance Func	Voting Func	Weighting Func
k1ChebShpEq	1	ChebyshevDistance	ShepardsWorkVote	EqualWeighting
k1ChebDisEq	1	ChebyshevDistance	InverseDistanceWeightedVote	EqualWeighting
k1ChebMajEq	1	ChebyshevDistance	MajorityClassVote	EqualWeighting
k5EucShpEq	5	EuclideanDistance	ShepardsWorkVote	EqualWeighting
k7EucShpEq	7	EuclideanDistance	ShepardsWorkVote	EqualWeighting
k3ManShpEq	3	ManhattanDistance	ShepardsWorkVote	EqualWeighting
k1EucShpRlf	1	EuclideanDistance	ShepardsWorkVote	ReliefFWeighting
k7ManShpEq	7	ManhattanDistance	ShepardsWorkVote	EqualWeighting
k1EucShpEq	1	EuclideanDistance	ShepardsWorkVote	EqualWeighting
k1EucDisRlf	1	EuclideanDistance	InverseDistanceWeightedVote	ReliefFWeighting
k1EucDisEq	1	EuclideanDistance	InverseDistanceWeightedVote	EqualWeighting
k1ManMajEq	1	ManhattanDistance	MajorityClassVote	EqualWeighting
k5ManShpEq	5	ManhattanDistance	ShepardsWorkVote	EqualWeighting
k1EucMajEq	1	EuclideanDistance	MajorityClassVote	EqualWeighting
k1ManShpEq	1	ManhattanDistance	ShepardsWorkVote	EqualWeighting
k1ManDisEq	1	ManhattanDistance	InverseDistanceWeightedVote	EqualWeighting
k1EucMajRlf	1	EuclideanDistance	MajorityClassVote	ReliefFWeighting
k7ManShpRlf	7	ManhattanDistance	ShepardsWorkVote	ReliefFWeighting
k1ManShpRlf	1	ManhattanDistance	ShepardsWorkVote	ReliefFWeighting
k1ManDisRlf	1	ManhattanDistance	InverseDistanceWeightedVote	ReliefFWeighting
k3ManShpRlf	3	ManhattanDistance	ShepardsWorkVote	ReliefFWeighting
k5ManShpRlf	5	ManhattanDistance	ShepardsWorkVote	ReliefFWeighting
k1ManMajRlf	1	ManhattanDistance	MajorityClassVote	ReliefFWeighting
k3EucShpRlf	3	EuclideanDistance	ShepardsWorkVote	ReliefFWeighting
k3EucShpEq	3	EuclideanDistance	ShepardsWorkVote	EqualWeighting
k3ManDisEq	3	ManhattanDistance	InverseDistanceWeightedVote	EqualWeighting
k5ChebShpEq	5	ChebyshevDistance	ShepardsWorkVote	EqualWeighting
k7EucShpRlf	7	EuclideanDistance	ShepardsWorkVote	ReliefFWeighting
k5EucShpRlf	5	EuclideanDistance	ShepardsWorkVote	ReliefFWeighting
k5ChebDisEq	5	ChebyshevDistance	InverseDistanceWeightedVote	EqualWeighting
k3EucDisEq	3	EuclideanDistance	InverseDistanceWeightedVote	EqualWeighting
k3ChebShpEq	3	ChebyshevDistance	ShepardsWorkVote	EqualWeighting
k5ManDisEq	5	ManhattanDistance	InverseDistanceWeightedVote	EqualWeighting
k3ChebDisEq	3	ChebyshevDistance	InverseDistanceWeightedVote	EqualWeighting
k1ChebDisRlf	1	ChebyshevDistance	InverseDistanceWeightedVote	ReliefFWeighting
k1ChebShpRlf	1	ChebyshevDistance	ShepardsWorkVote	ReliefFWeighting
k1ChebMajRlf	1	ChebyshevDistance	MajorityClassVote	ReliefFWeighting
k5EucDisEq	5	EuclideanDistance	InverseDistanceWeightedVote	EqualWeighting
k7ManDisEq	7	ManhattanDistance	InverseDistanceWeightedVote	EqualWeighting
k5ChebMajEq	5	ChebyshevDistance	MajorityClassVote	EqualWeighting
k7EucDisEq	7	EuclideanDistance	InverseDistanceWeightedVote	EqualWeighting
k7ChebShpRlf	7	ChebyshevDistance	ShepardsWorkVote	ReliefFWeighting
k7ChebDisEq	7	ChebyshevDistance	InverseDistanceWeightedVote	EqualWeighting
k7ChebShpEq	7	ChebyshevDistance	ShepardsWorkVote	EqualWeighting
k5ChebShpRlf	5	ChebyshevDistance	ShepardsWorkVote	ReliefFWeighting
k3ChebShpRlf	3	ChebyshevDistance	ShepardsWorkVote	ReliefFWeighting
k3EucDisRlf	3	EuclideanDistance	InverseDistanceWeightedVote	ReliefFWeighting
k3EucMajRlf	3	EuclideanDistance	MajorityClassVote	ReliefFWeighting
k5EucDisRlf	5	EuclideanDistance	InverseDistanceWeightedVote	ReliefFWeighting
k5EucMajRlf	5	EuclideanDistance	MajorityClassVote	ReliefFWeighting
k5ChebDisRlf	5	ChebyshevDistance	InverseDistanceWeightedVote	ReliefFWeighting
k3ChebDisRlf	3	ChebyshevDistance	InverseDistanceWeightedVote	ReliefFWeighting
k7EucDisRlf	7	EuclideanDistance	InverseDistanceWeightedVote	ReliefFWeighting
k7EucMajRlf	7	EuclideanDistance	MajorityClassVote	ReliefFWeighting
k5EucMajEq	5	EuclideanDistance	MajorityClassVote	EqualWeighting
k5ManMajEq	5	ManhattanDistance	MajorityClassVote	EqualWeighting
k3ChebMajRlf	3	ChebyshevDistance	MajorityClassVote	ReliefFWeighting

Table 12: KNN-Reduction Legend

Model Label	K	Distance Func	Voting Func	Weighting Func
Ctrlk1ManMajEq	1	ManhattanDistance	MajorityClassVote	EqualWeighting
Ennthk1ManMajEq	1	ManhattanDistance	MajorityClassVote	EqualWeighting
Ggcnk1ManMajEq	1	ManhattanDistance	MajorityClassVote	EqualWeighting
Drop3k1ManMajEq	1	ManhattanDistance	MajorityClassVote	EqualWeighting

Table 13: SVM Legend

Model Label	C	Kernel Type
C7Rbf	7	rbf
C5Rbf	5	rbf
C3Poly	3	poly
C1Poly	1	poly
C3Rbf	3	rbf
C5Poly	5	poly
C7Poly	7	poly
C1Rbf	1	rbf
C3Lin	3	linear
C1Lin	1	linear
C5Lin	5	linear
C7Lin	7	linear
C1Sig	1	sigmoid
C3Sig	3	sigmoid
C5Sig	5	sigmoid
C7Sig	7	sigmoid

Table 14: SVM-Reduction Legend

Model Label	C	Kernel Type
CtrlC7Rbf	7	rbf
EnnthC7Rbf	7	rbf
GgcnC7Rbf	7	rbf
Drop3C7Rbf	7	rbf