Work 2 - Classification with Lazy Learning and SVM

Zachary Parent, Sheena Lang, Kacper Poniatowski and Carlos Jiménez

November 1, 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 (k-NN) 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 k-NN can be used for both classification and regression problems, our focus is on classification due to the nature of the datasets used.

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

We hypothesize that k-NN 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 k-NN algorithm on the training dataset.
3. Model Evaluation: The performance of the best k-NN 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.

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.

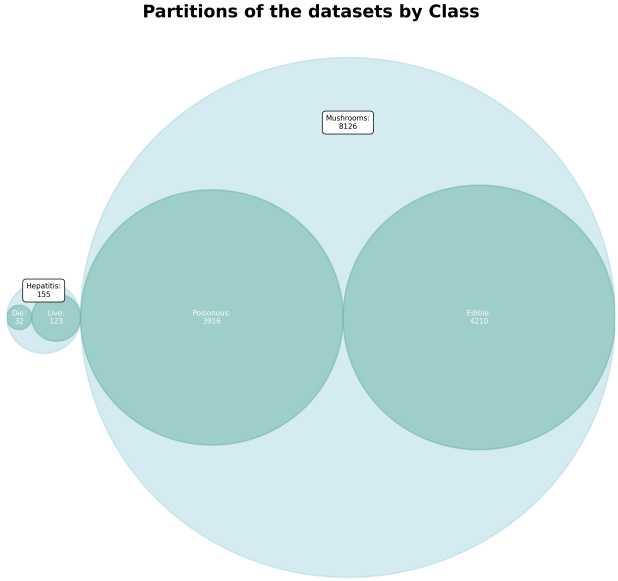


Figure 1: Dataset partitions

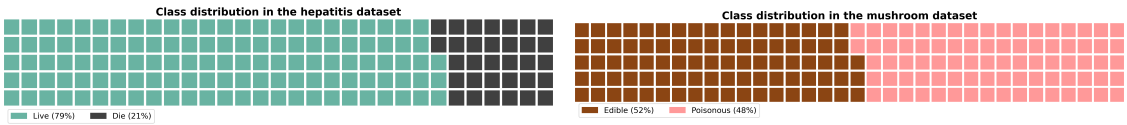


Figure 2: Class distributions

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 also considered one-hot encoding, which avoids implying any ordinal relationship among categories, we preferred label encoding for its simplicity and reduced dimensionality, especially given 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.

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.

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 k-NN 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). 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 k-NN 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 k-NN can be used for both classification and regression problems our focus is on classification.

While the k-NN algorithm has it's merits in terms of simplicity and interpretability, it also has several disadvantages:

- Computationally expensive: As the number of training examples grows, the algorithm's complexity increases[6].
- 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.

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 k-NN algorithm to increase performance.

3.1.2 Implementation Details

The K-Nearest Neighbors (k-NN) 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 k-NN algorithms due to its simplicity and effectiveness in measuring the similarity between data points[5]. 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 k-NN algorithms. It is also frequently used with the k-NN 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 k-NN 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.

Weighting Schemes

In the k-NN 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 k-NN. 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 k-NN 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[3] This is the default voting scheme in k-NN 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[3]

Shepard's method is another voting scheme that can be used in k-NN 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[4]

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 k-NN 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 k-NN 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 k-NN algorithm in our study:

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

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 k-NN 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 k-NN 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 k-NN 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) could delay critical treatment, while a false positive 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
- Faster consultation times than KNN, thanks to training step

Disadvantages of SVM include:

- Sensitivity to the choice of kernel function and hyperparameters
- Computational complexity for large 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.

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".

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

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 [7].

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 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.

3.3.2 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 [**?**]. 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. [CITE]

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. [CITE]

3.3.3 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.

Table 1: Results from KNN models for the mushroom dataset

	k	distance func	voting func	weighting func	accuracy	f1
1	7	ManhattanDistance	ShepardsWorkVote	EqualWeighting	0.951	0.952
2	5	ManhattanDistance	ShepardsWorkVote	EqualWeighting	0.951	0.952
3	3	ManhattanDistance	ShepardsWorkVote	EqualWeighting	0.951	0.952
4	1	ManhattanDistance	MajorityClassVote	EqualWeighting	0.950	0.951
5	1	ManhattanDistance	InverseDistanceWeightedVote	EqualWeighting	0.950	0.951
6	1	ManhattanDistance	ShepardsWorkVote	EqualWeighting	0.950	0.951
7	5	ManhattanDistance	InverseDistanceWeightedVote	EqualWeighting	0.933	0.933
8	7	ManhattanDistance	InverseDistanceWeightedVote	EqualWeighting	0.930	0.929
9	3	ManhattanDistance	InverseDistanceWeightedVote	EqualWeighting	0.928	0.929
10	3	ManhattanDistance	MajorityClassVote	EqualWeighting	0.926	0.927

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.

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.

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

Table 2: Results from KNN models for the hepatitis dataset

	k	distance func	voting func	weighting func	accuracy	f1
1	1	EuclideanDistance	ShepardsWorkVote	ReliefFWeighting	0.955	0.972
2	1	EuclideanDistance	InverseDistanceWeightedVote	ReliefFWeighting	0.955	0.972
3	1	EuclideanDistance	MajorityClassVote	ReliefFWeighting	0.955	0.972
4	1	ChebyshevDistance	ShepardsWorkVote	EqualWeighting	0.948	0.969
5	1	ChebyshevDistance	InverseDistanceWeightedVote	EqualWeighting	0.948	0.969
6	1	ChebyshevDistance	MajorityClassVote	EqualWeighting	0.948	0.969
7	7	EuclideanDistance	ShepardsWorkVote	EqualWeighting	0.948	0.968
8	5	EuclideanDistance	ShepardsWorkVote	EqualWeighting	0.948	0.968
9	1	ManhattanDistance	MajorityClassVote	EqualWeighting	0.948	0.967
10	7	ManhattanDistance	ShepardsWorkVote	EqualWeighting	0.948	0.967

Table 3: Results from SVM models for the mushroom dataset

	C	kernel type	accuracy	f1
1	1	poly	0.928	0.928
2	7	rbf	0.926	0.926
3	5	rbf	0.925	0.925
4	3	rbf	0.924	0.924
5	7	poly	0.920	0.920
6	5	poly	0.916	0.916
7	3	poly	0.916	0.915
8	3	linear	0.914	0.912
9	1	linear	0.911	0.910
10	1	rbf	0.906	0.902

Table 4: Results from SVM models for the hepatitis dataset

	C	kernel type	accuracy	f1
1	7	rbf	0.955	0.972
2	3	rbf	0.948	0.968
3	5	rbf	0.948	0.968
4	1	poly	0.948	0.968
5	3	poly	0.948	0.967
6	5	poly	0.948	0.967
7	7	poly	0.948	0.967
8	1	rbf	0.903	0.941
9	3	linear	0.890	0.929
10	1	linear	0.884	0.927

Table 5: Results from KNN models for the mushroom dataset with dimensionality reduction

	k	reduction func	accuracy	f1	train time	test time	storage
1	1	control	0.950	0.951	0.000	0.255	1000
2	1	enn	0.950	0.951	0.000	0.255	1000
3	1	drop2	0.950	0.951	0.000	0.257	1000
4	1	cnn	0.925	0.929	0.000	0.065	189

Table 6: Results from KNN models for the hepatitis dataset with dimensionality reduction

	k	reduction func	accuracy	f1	train time	test time	storage
1	1	control	0.948	0.967	0.001	0.070	1395
2	1	enn	0.948	0.967	0.000	0.072	1395
3	1	drop2	0.948	0.967	0.000	0.073	1395
4	1	cnn	0.877	0.921	0.000	0.021	409

General Performance Analysis

As seen in Tables 1 and 2, the general performance of the k-NN 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.

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

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.

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 k-NN 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.

4.2.5 Support Vector Machines (SVM) Analysis

As seen in Table 3 and Table 4, SVMs performed well on both the hepatitis and the mushroom datasets, achieving peak accuracy and F1 scores which match the best results from KNN.

A key advantage of the SVM over KNN is that SVMs are much faster during consultation time

For both the hepatitis and mushroom datasets, the configuration using 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 just as well when used with $C = 1$.

4.2.6 Instance Reduction Analysis

This section interprets the results of the dimensionality reduction techniques, discussing their impact on the analysis and visualization.

Figure [INSERT FIGURE REF] shows the results of our analyzis in terms of accuracy, training speed and storage reduction capbilities of the instance reduction techniques implemented (*i.e.*, GCNN, ENNTH and DROP3)

...

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.

For KNN applied to the Hepatitis dataset, applying no redcution yields a high F1 score of 0.9483, with moderate storage requirements of 139.5 units and a testing time of 0.0747 seconds. The GGCN reduction technique slightly lowers the F1 score to 0.9354 but significantly improves testing time to 0.0545 seconds and reduces storage to 103.2 units. GGCN achieves this by selectively condensing the dataset, retaining instances that best represent the overall data structure and class boundaries. By removing redundant points that do not impact classification accuracy, it effectively reduces storage requirements and enhances processing speed without sacrificing much performance.

The F1 score for ENNTH is the same as applying no reduction, and achieves the fastest training time of 0.00042 seconds, but results in almost no reduction in storage since its approach primarily focuses on removing noisy data points at the boundaries. Given the relatively low noise in the Hepatitis dataset, ENNTH performs similarly to applying not reduction in terms of storage efficiency. Conversely, Drop3 reduces storage to 110.9 units but leads to a more significant drop in the F1 score to 0.9157. This reduction in performance can be attributed to Drop3's aggressive removal of instances based on noise and redundancy filtering, which can inadvertently discard valuable data points and degrade predictive accuracy when the dataset is sparse.

When analyzing the Mushroom dataset with KNN, all techniques achieve a perfect F1 score of 1.0, indicating no loss in predictive accuracy. GGCN stands out by significantly reducing storage from 7311.6 units to 3519.8 units while also decreasing testing time from 72.78 seconds to 33.70 seconds. This effectiveness arises

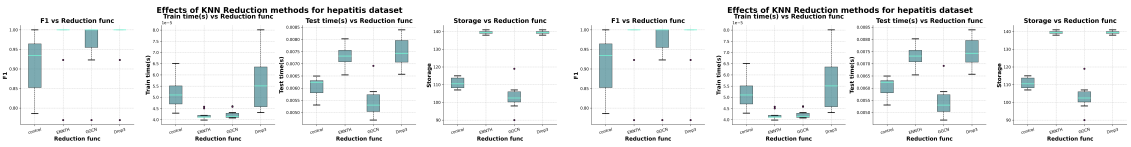


Figure 3: Class distributions

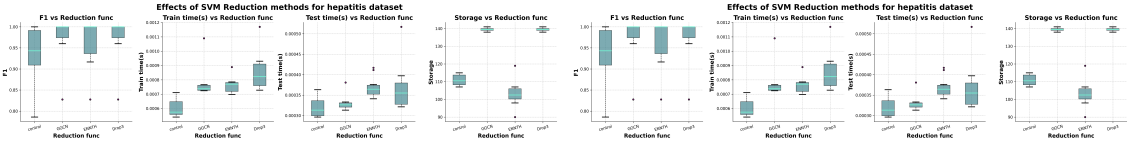


Figure 4: Class distributions

from GGCN's ability to eliminate redundancy while retaining essential data points, making it particularly advantageous for large datasets.

ENNTH and Drop3 retain the same storage requirement as the control, both at 7311.6 units, with only slight improvements in training and testing time. ENNTH preserves the perfect F1 score by focusing on noise removal, which is less impactful in this dataset due to its inherent cleanliness. Similarly, Drop3 achieves a perfect F1 score, as its filtering approach does not affect the well-structured Mushroom dataset.

In the context of the Hepatitis dataset using SVM, using no reduction method produces a high F1 score of 0.9719, requiring moderate storage of 139.5 units and a testing time of 0.0037 seconds. The GGCN technique yields a slightly lower F1 score of 0.9597, but it reduces storage to 103.2 units while maintaining a similar testing time of 0.0037 seconds.

ENNTH maintains the original F1 score of 0.9719 and achieves a testing time of 0.0033 seconds, but it results in almost no storage reduction, as its approach primarily focuses on eliminating noisy boundary points. The relatively low noise in the Hepatitis dataset means ENNTH performs comparably to the control in terms of storage usage. On the other hand, Drop3 reduces storage to 110.9 units but results in a substantial drop in F1 score to 0.9291, reflecting its aggressive filtering approach, which can lead to the loss of critical instances and, consequently, a decline in predictive performance.

For SVM applied to the Mushroom dataset, all methods, yield a perfect F1 score of 1.0, ensuring no predictive accuracy is sacrificed. GGCN proves especially effective by reducing storage from 7311.6 units to 3519.8 units and decreasing testing time from 0.0421 seconds to 0.0337 seconds.

ENNTH and Drop3 maintain the same storage requirements as the control, both at 7311.6 units, with minor improvements in training and testing time. ENNTH's focus on noise removal contributes to its sustained perfect F1 score, while Drop3's method similarly does not impact storage significantly in this densely populated dataset.

Overall, GGCN emerges as the most effective reduction technique across both models, showcasing substantial efficiency gains while preserving predictive accuracy, particularly for larger datasets like Mushroom.

4.3 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:

4.3.1 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) [CITE].

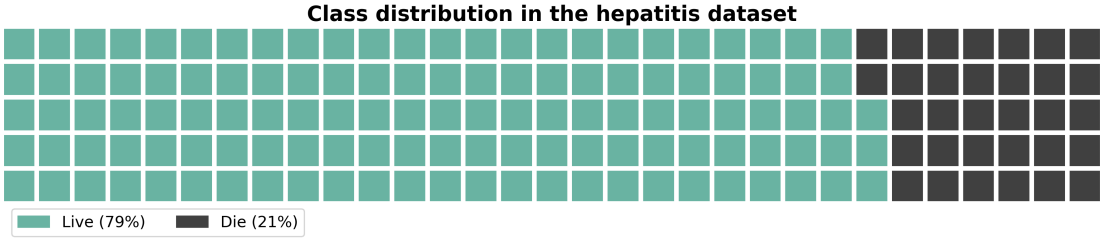


Figure 5: Class distribution of the Hepatitis dataset

Furthermore, the performance metrics distribution across folds in Figure 6 further demonstrates non-normal distribution:

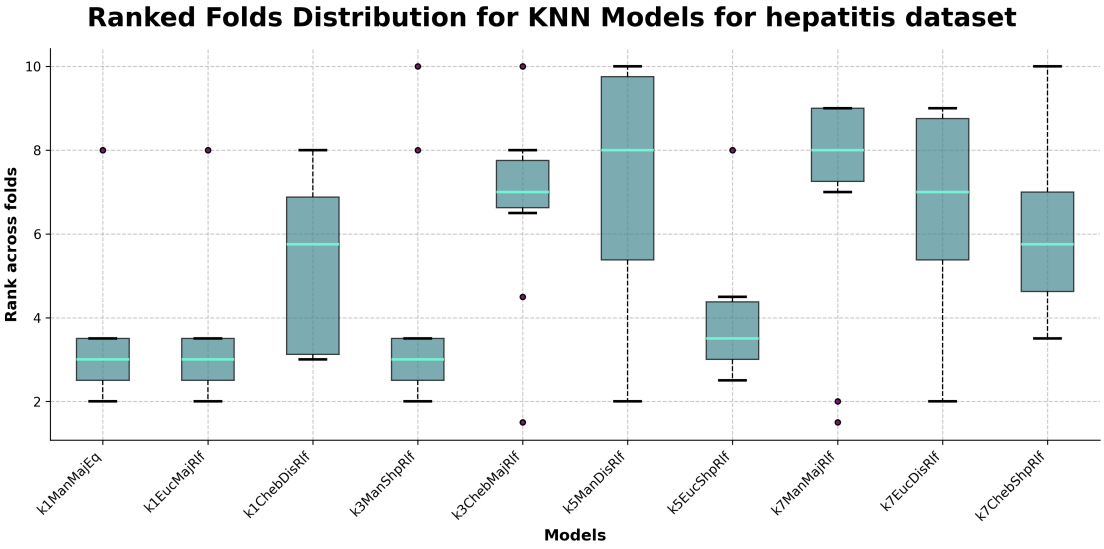


Figure 6: Rank distributions across folds for different KNN configurations on the Hepatitis dataset

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.

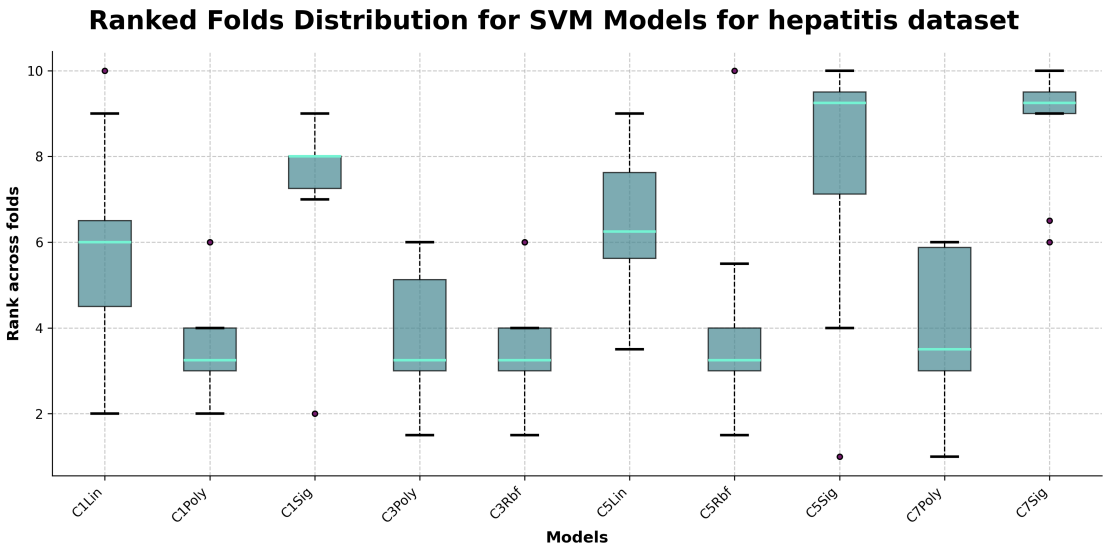


Figure 7: Rank distributions across folds for different KNN configurations on the Hepatitis dataset

4.3.2 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 [CITE].

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.

4.3.3 Post-Hoc Tests

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 [CITE]. 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.

4.3.4 Test Results

In the visualization 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.

For the below analysis of both the k-NN and SVM models, 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.

k-Nearest Neighbors (KNN) Analysis

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

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.

k-Nearest Neighbors (SVM) Analysis

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

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] University College Cork. Lecture notes - classification.

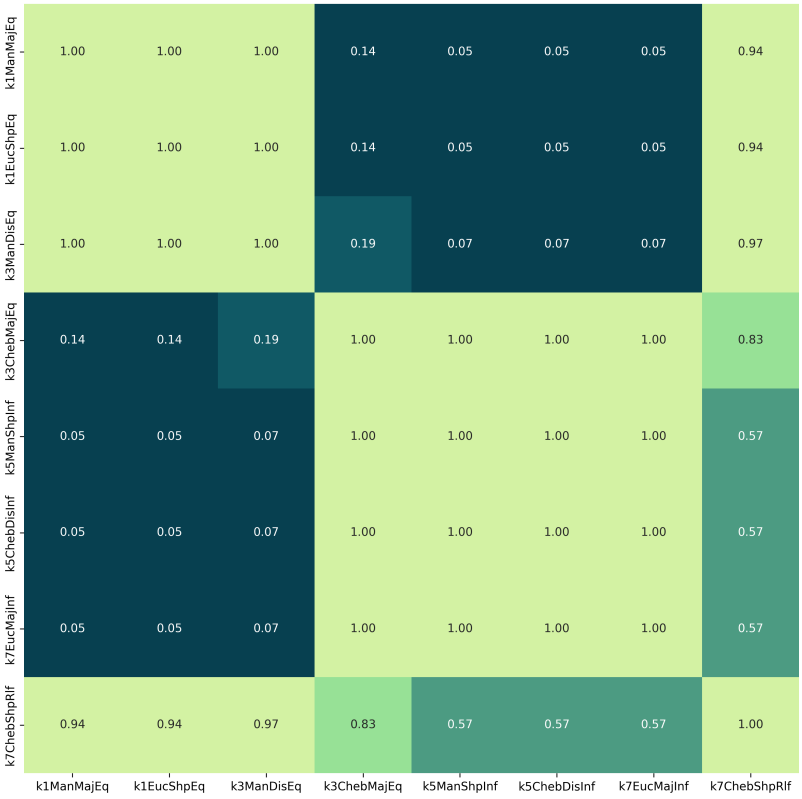
[4] Padraig Cunningham and Sarah Jane Delany. k-nearest neighbour classifiers - a tutorial. *ACM Computing Surveys*, 54(6):128:1–128:25, 2021.

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

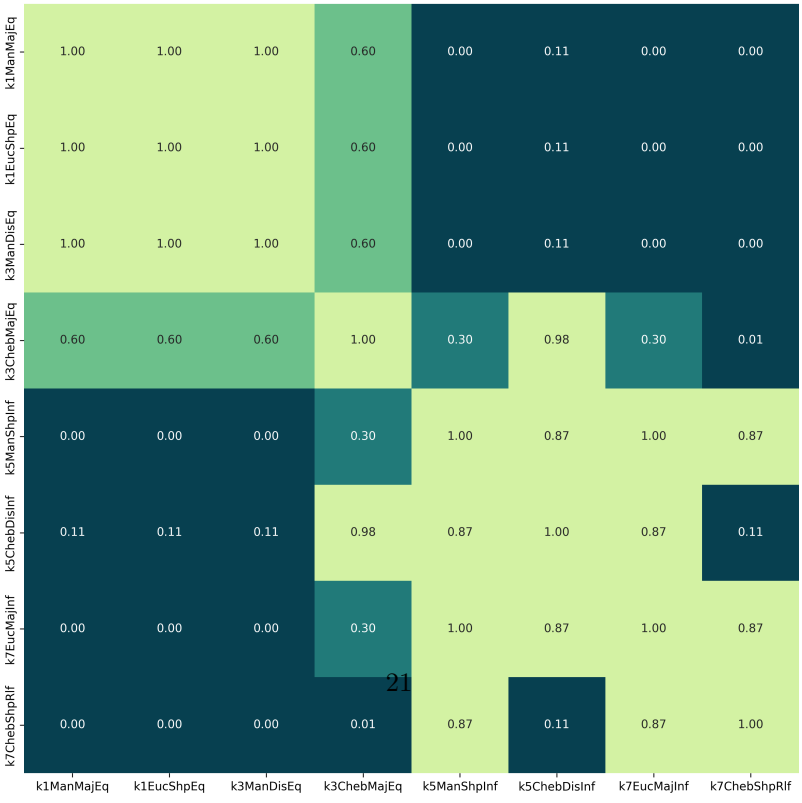
[6] Trevor LaViale. Deep dive on knn: Understanding and implementing the k-nearest neighbors algorithm, 2023.

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

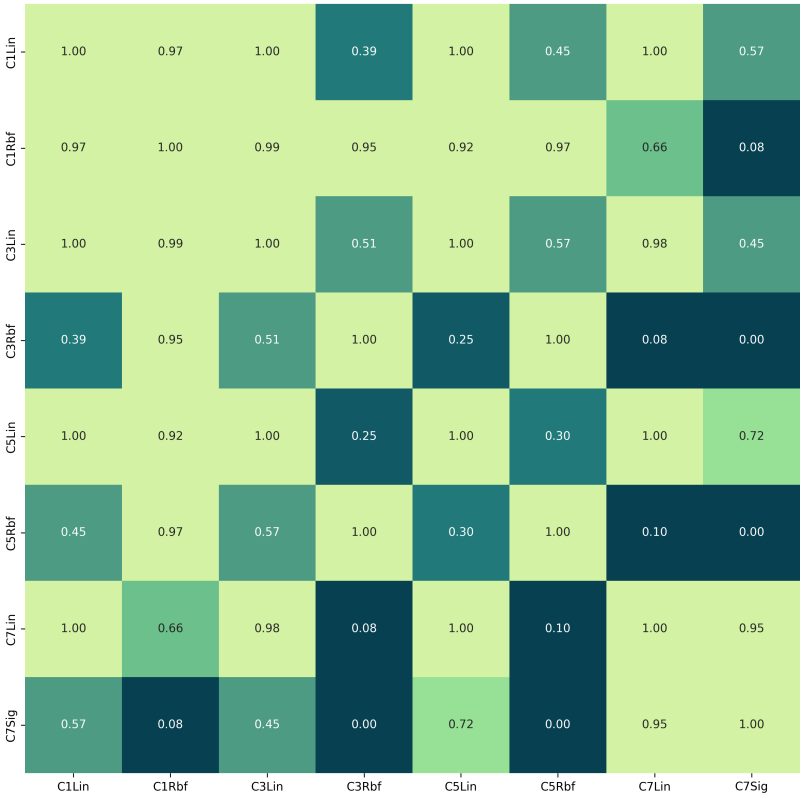
Nemenyi Test Results for KNN Models for hepatitis dataset



Nemenyi Test Results for KNN Models for mushroom dataset



Nemenyi Test Results for KNN Models for hepatitis dataset



Nemenyi Test Results for KNN Models for mushroom dataset