Implementation and Evaluation of KNN and SVM Classification Configurations, with Instance Reduction Techniques

Zachary Parent, Sheena Lang, Kacper Poniatowski and Carlos Jiménez

November 22, 2024

Contents

1 Introduction

1.1 Background

1.2 Research Objectives

1.

1.3 Methodology Overview

•

2 Data

This section describes the datasets used in our study and details the preprocessing steps applied to prepare them for analysis.

2.1 Dataset Selection and Characteristics

This section outlines our dataset selection criteria and analyzes the characteristics of the chosen datasets.

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.

Smaller datasets A smaller dataset enables rapid experimentation and comparison of SVM and KNN performance.

Larger datasets A larger dataset provides an excellent opportunity to evaluate the benefits of reduction methods in KNN, especially regarding storage efficiency and reduced running time [14].

Attribute types 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.

Class distribution 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.

Missing data 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 Selection Criteria

We selected two datasets with contrasting characteristics to evaluate the performance of Support Vector Machine (SVM) and k-Nearest Neighbors (KNN) algorithms across different scenarios. Our selection criteria focused on:

- Dataset size variation (small vs. large)
- Attribute type diversity (nominal vs. numerical)
- Class distribution (balanced vs. imbalanced)
- Missing data patterns

A smaller dataset enables rapid experimentation and initial algorithm validation, while a larger dataset allows evaluation of reduction methods' effectiveness, particularly for KNN's storage and computational requirements.

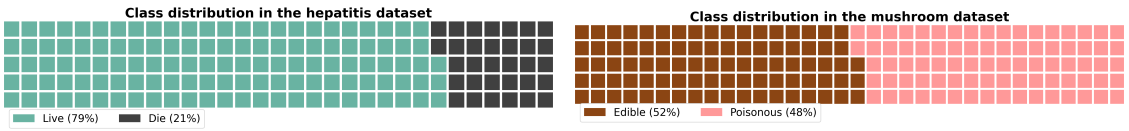

Figure 1: Class distribution comparison between datasets

2.1.3 Dataset Characteristics

- **Mushroom Dataset**
  - 8,124 instances
  - 22 nominal attributes
  - Binary classification (edible vs. poisonous)
  - Nearly balanced classes
- **Hepatitis Dataset**
  - 155 instances
  - Mixed attribute types (both nominal and numerical)
  - Binary classification (survive vs. die)
  - Imbalanced classes (79.35% majority class)
  - 6.01% missing values

These datasets provide complementary characteristics for evaluating our algorithms' performance across different scenarios. The Mushroom dataset challenges the algorithms with its size and categorical nature, while the Hepatitis dataset tests their ability to handle mixed data types and class imbalance.

2.2 Data Preprocessing

This section details the preprocessing steps applied to prepare both datasets for analysis.

2.2.1 Handling Different Data Types and Ranges

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 [1].

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 [5].

We implemented specific preprocessing techniques based on the characteristics of each dataset:

- **Nominal Attributes**
  - Applied label encoding to both datasets
  - Chose label encoding over one-hot encoding to prevent dimensionality explosion
  - Particularly important for the Mushroom dataset's numerous categorical features

- **Numerical Attributes**
  - Applied min-max scaling to the Hepatitis dataset's numerical features
  - Normalized all values to [0, 1] range
  - Essential for distance-based algorithms (KNN and SVM)

2.2.2 Missing Value Treatment

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 [5].

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.

We employed different strategies for handling missing values based on attribute type:

- **Nominal Attributes**
  - Imputed with mode (majority class)
  - Applied to both datasets
  - Potential limitation: May reinforce majority class bias
- **Numerical Attributes**
  - Imputed with mean values
  - Applied only to Hepatitis dataset
  - Preserves overall distribution while handling missing data

Alternative approaches such as KNN-based imputation were considered but rejected to avoid introducing bias into our evaluation of KNN as a classifier. Our chosen methods provide a balance between simplicity and effectiveness while maintaining data integrity.

3 Methods

This section details our experimental methodology, including algorithm implementations, parameter configurations, and statistical analysis approaches. We present our methods in four main parts:

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) [7]. The algorithm's effectiveness relies on two fundamental assumptions:

- **Locality**: Points in close proximity are likely to share the same class
- **Smoothness**: Decision boundaries between classes are 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: 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 [14]:

- Computationally expensive: As the number of training examples grows, the algorithm's complexity increases[17].
- 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 [11].

The choice of k significantly influences the algorithm's behavior:

- Small k values (1-3): Higher sensitivity to local patterns but more susceptible to noise
- Large k values (5-7): Better noise resistance but may miss important local patterns
- Even vs. Odd k values: Even values may require tie-breaking mechanisms

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 [14, 7].

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[16]. 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.

Weighting Schemes Three weighting approaches were implemented:

- **Uniform**: Equal weights for all neighbors
- **ReliefF**: Weights based on feature relevance
- **Information Gain**: Weights based on information theory

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[2] 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[6] 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[6]

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[8]

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 [13].
- **Pandas**: A data manipulation library in Python, used for data preprocessing and analysis [18].
- **Matplotlib**: A plotting library in Python, used for data visualization [15].
- **Seaborn**: A data visualization library in Python, used for creating informative and attractive statistical graphics [22].
- **SciPy**: A scientific computing library in Python, used for scientific and technical computing [21].

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.

F1 Score 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 1

Considerations in medical diagnostics 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[3].

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
- Memory efficiency 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 KNN (see subsection 3.1), we used scikit-learn's SVM implementation [1]. We specifically used `SVC` (Support Vector Classification) as it allows for non-linear classification through kernel functions.

- `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.

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: $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$

Regularization Parameter The regularization parameter C controls the trade-off between achieving a low training error and a low testing error. We explored values [0.05, 0.5, 5, 50] to cover a wide range of regularization strengths:

- Low values (0.05, 0.5): Stronger regularization, simpler decision boundaries
- Medium values (5): Balanced regularization
- High values (50): Weaker regularization, more complex decision boundaries

This logarithmic spacing allows us to explore both strong and weak regularization while keeping the number of experiments manageable.

Extremely high C values (much higher than our maximum of 50) can lead to overfitting, as the model attempts to perfectly classify all training points at the expense of generalization. Conversely, extremely low C values (much lower than our minimum of 0.05) can lead to underfitting, as the model becomes too simple to capture the underlying patterns in the data. Our chosen range aims to find a balance between these extremes.

Performance Evaluation Consistent with our KNN evaluation (see subsection 3.1), we assessed SVM performance using:

- Accuracy and F1 Score
- Confusion matrix metrics (TP, TN, FP, FN)
- Training and testing times
- 10-fold cross-validation

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

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.

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.

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 chosen as our primary metric for the same reasons discussed in the KNN section - it provides a balanced measure particularly suitable for imbalanced datasets like Hepatitis.

Cross-validation folds were kept identical between KNN and SVM experiments to ensure fair comparison. For detailed preprocessing steps, refer to section 2.

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 [23, 19].

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 [4]. 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 [4].

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 [23, 20]. 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 [20].

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 [20].

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 [23] 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) [23]. 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)** [23, 20] 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)** [23] 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.

Note: In order to keep the tables and figures reasonable in size, we use a short label system for model configurations. The legend for these labels can be found in the appendix section 6.

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) [10, 24].

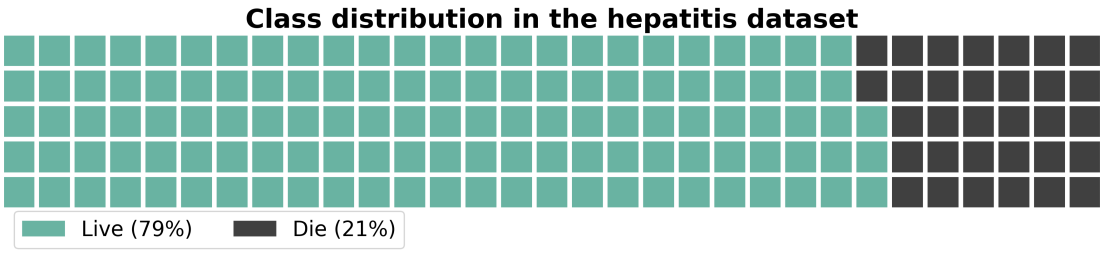

Figure 2: Class distribution of the Hepatitis dataset

Furthermore, the performance metrics distribution across folds in Figure 3 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 (see Figure 2), 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.

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 [24, 12].

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.

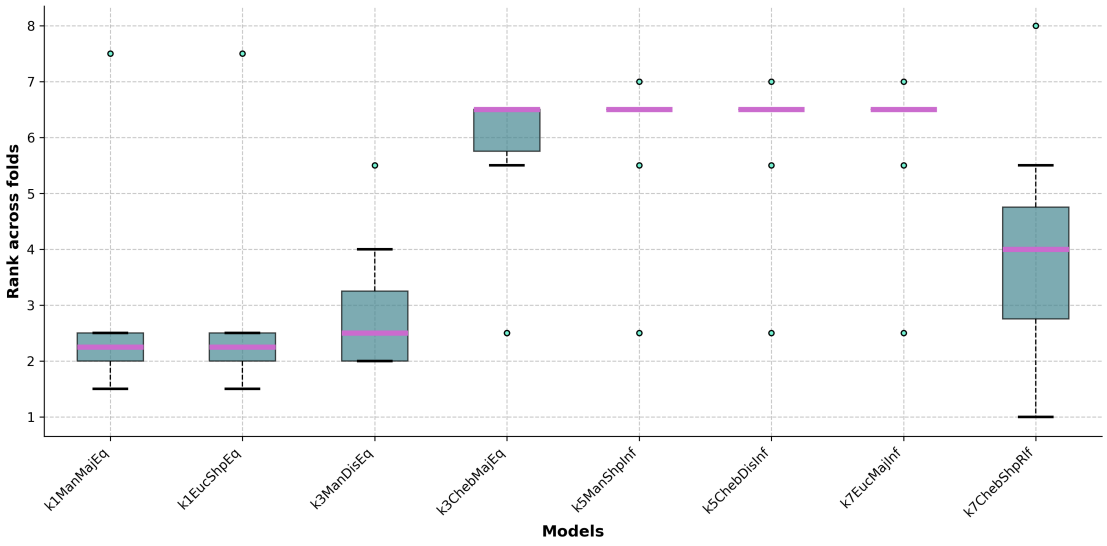

Figure 3: Rank distributions across folds for different KNN configurations on the Hepatitis dataset

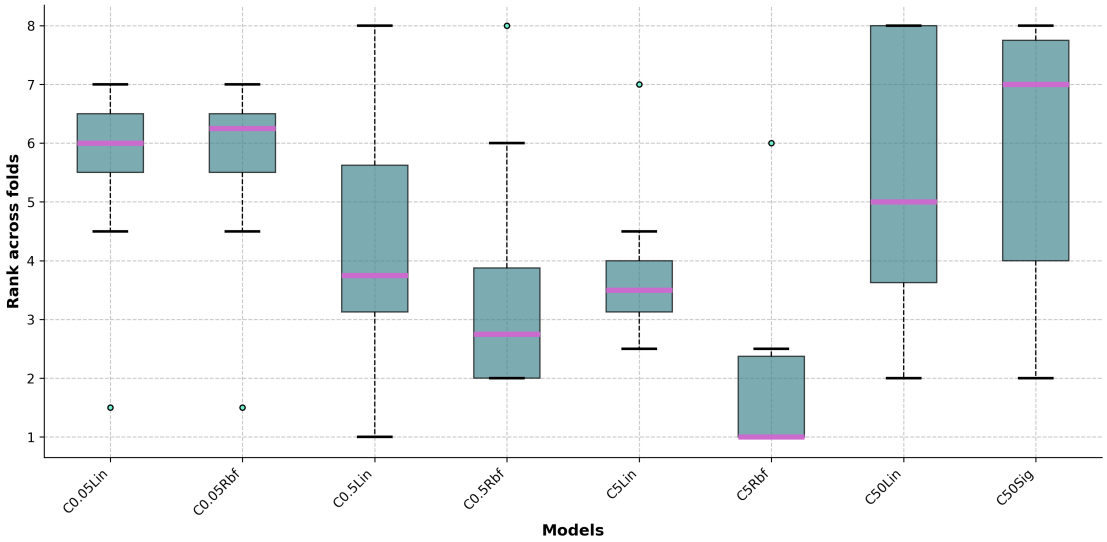

Figure 4: Rank distributions across folds for different SVM configurations on the Hepatitis dataset

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:

3.4.4 Wilcoxon Signed-Rank Test

To directly compare the performance between SVM and KNN models, we employ the Wilcoxon signed-rank test. This non-parametric test is particularly suitable for paired comparisons between two related samples, making it ideal for comparing two different classifiers on the same datasets [9].

The Wilcoxon signed-rank test works by:

1. Taking the difference between paired observations (in our case, the performance metrics of SVM and KNN on the same fold)
2. Ranking these differences by their absolute values
3. Summing the ranks of positive and negative differences separately

Like the Friedman test, the Wilcoxon test makes no assumptions about the normality of the data, which aligns with our earlier observations about the non-normal distribution of performance metrics. The test provides a more focused comparison between our two classifier types than the Friedman test, which compares all configurations simultaneously [12, 9].

The null hypothesis for the Wilcoxon test is that the median difference between pairs of observations is zero. In our context, this means testing whether there is a significant difference in performance between SVM and KNN models. A p-value below our significance level ($\alpha = 0.05$) would indicate a statistically significant difference between the two classifiers' performances.

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

where q_α is based on the Studentized range statistic, k is the number of classifiers, and N is the number of datasets [12]. 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.

3.4.5 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.

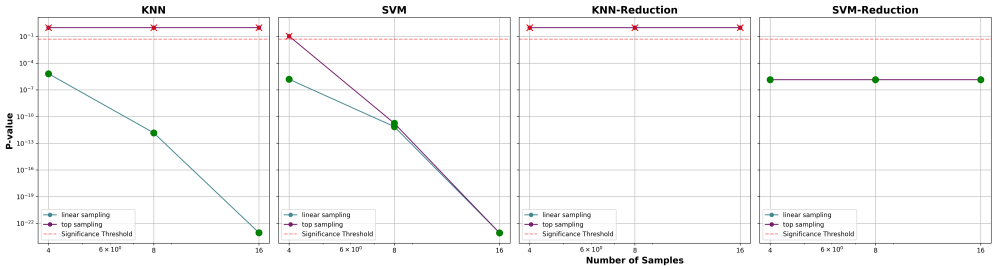

Figure 5: P values vs. number of samples for different models and sampling methods for the mushroom dataset

As seen in Figure 5 and Figure 6, 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

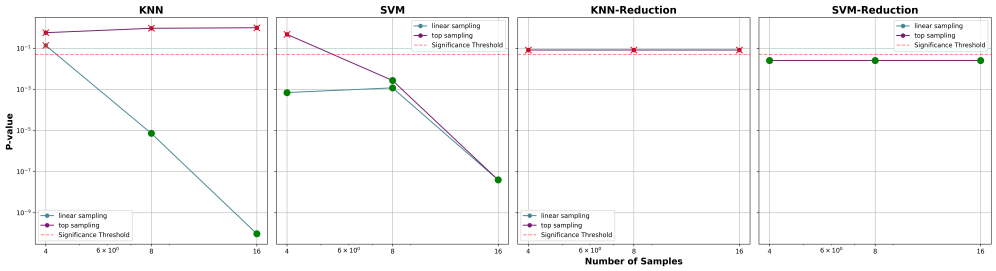

Figure 6: P values vs. number of samples for different models and sampling methods for the hepatitis dataset

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 justify our decision, see Table 1 and Table 2 for the Friedman test results for both 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	1.7732928268502093e-11
KNN Linear Sample of 8 F1 Scores	1.477429493282523e-12
SVM Linear Sample of 8 F1 Scores	7.435065423603964e-12
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	0.0002950344433046935

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	0.002701273415410788
KNN Linear Sample of 8 F1 Scores	7.258355849331829e-06
SVM Linear Sample of 8 F1 Scores	0.001183031044060309
KNN-Reduction Linear Sample of 8 F1 Scores	0.08210005961379935
SVM-Reduction Linear Sample of 8 F1 Scores	0.02537399878878677
KNN-Reduction Linear Sample of 8 Training Times	0.00037357398316662454
SVM-Reduction Linear Sample of 8 Training Times	5.555987171561241e-06

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.

From the data, 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.

4 Results and Analysis

Table 3: Results From Knn Models For The Mushroom Dataset

	Model Label	Mean F1	Mean Train Time (s)	Mean Test Time (s)
1	k1ManMajEq	1.000000	0.000251	15.225325
2	k3EucDisEq	1.000000	0.000196	17.752951
3	k1EucMajEq	1.000000	0.000225	17.891188
4	k1ManDisEq	1.000000	0.000264	15.544714
5	k1ManShpEq	1.000000	0.000246	14.987898
6	k1EucDisEq	1.000000	0.000234	17.843295
7	k3ManMajEq	1.000000	0.000200	14.979317
8	k3ManDisEq	1.000000	0.000217	15.154011
9	k3ManShpEq	1.000000	0.000227	15.163488
10	k3EucMajEq	1.000000	0.000211	18.054825

Table 4: Results From Knn Models For The Hepatitis Dataset

	Model Label	Mean F1	Mean Train Time (s)	Mean Test Time (s)
1	k1ChebShpEq	0.972618	0.000084	0.006925
2	k1ChebMajEq	0.972618	0.000085	0.006949
3	k1ChebDisEq	0.972618	0.000084	0.006896
4	k5EucShpEq	0.969271	0.000084	0.009300
5	k7EucShpEq	0.969271	0.000083	0.009293
6	k3ManShpEq	0.969231	0.000083	0.007033
7	k1ManMajEq	0.969231	0.000120	0.007153
8	k1EucShpRlf	0.969231	0.000084	0.009349
9	k1ManShpEq	0.969231	0.000083	0.006982
10	k1EucMajEq	0.969231	0.000086	0.009228

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

	Model Label	Mean F1	Mean Train Time (s)	Mean Test Time (s)	Mean Storage
1	Ctrlk1ManMajEq	1.000000	0.000151	15.164845	7311.600000
2	Gcnnk1ManMajEq	1.000000	0.000107	7.277776	3519.800000
3	Ennthk1ManMajEq	1.000000	0.000241	15.090724	7311.600000
4	Drop3k1ManMajEq	1.000000	0.000185	15.079619	7311.600000

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

	Model Label	Mean F1	Mean Train Time (s)	Mean Test Time (s)	Mean Storage
1	Ctrlk1ManMajEq	0.969231	0.000056	0.007470	139.500000
2	Ennthk1ManMajEq	0.969231	0.000042	0.007297	139.500000
3	Gcnnk1ManMajEq	0.960765	0.000042	0.005446	103.200000
4	Drop3k1ManMajEq	0.915608	0.000052	0.006057	110.900000

4.1 k-Nearest Neighbors (KNN) Analysis

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

Table 7: Results From Svm Models For The Mushroom Dataset

	Model Label	Mean F1	Mean Train Time (s)	Mean Test Time (s)
1	C5Poly	1.000000	0.117225	0.003384
2	C50Poly	1.000000	0.124768	0.002183
3	C50Rbf	1.000000	0.086084	0.007384
4	C5Rbf	0.999618	0.101647	0.015565
5	C0.5Poly	0.995142	0.155683	0.011136
6	C50Lin	0.984586	17.644338	0.013022
7	C0.5Rbf	0.984523	0.252811	0.047263
8	C5Lin	0.982752	1.217328	0.012979
9	C0.5Lin	0.965123	0.517853	0.015964
10	C0.05Poly	0.954660	0.339317	0.028346

Table 8: Results From Svm Models For The Hepatitis Dataset

	Model Label	Mean F1	Mean Train Time (s)	Mean Test Time (s)
1	C50Rbf	0.972671	0.000885	0.000371
2	C5Rbf	0.970963	0.000881	0.000367
3	C5Poly	0.968619	0.000871	0.000335
4	C0.5Poly	0.966615	0.000867	0.000362
5	C50Poly	0.963857	0.000872	0.000341
6	C0.05Poly	0.931146	0.000788	0.000354
7	C0.5Rbf	0.927927	0.001005	0.000418
8	C5Lin	0.920826	0.000907	0.000343
9	C0.5Lin	0.919482	0.000804	0.000359
10	C0.05Lin	0.887550	0.000846	0.000374

Table 9: Results From Svm-Reduction Models For The Mushroom Dataset

	Model Label	Mean F1	Mean Train Time (s)	Mean Test Time (s)	Mean Storage
1	CtrlC5Poly	1.000000	0.121528	0.003708	7311.600000
2	EnnthC5Poly	1.000000	0.118410	0.003434	7311.600000
3	Drop3C5Poly	1.000000	0.119210	0.003404	7311.600000
4	GcnnC5Poly	0.972062	0.039924	0.002862	3519.800000

Table 10: Results From Svm-Reduction Models For The Hepatitis Dataset

	Model Label	Mean F1	Mean Train Time (s)	Mean Test Time (s)	Mean Storage
1	CtrlC50Rbf	0.972671	0.000837	0.000401	139.500000
2	EnnthC50Rbf	0.972671	0.000794	0.000367	139.500000
3	GcnnC50Rbf	0.964967	0.000777	0.000370	103.200000
4	Drop3C50Rbf	0.928879	0.000611	0.000351	110.900000

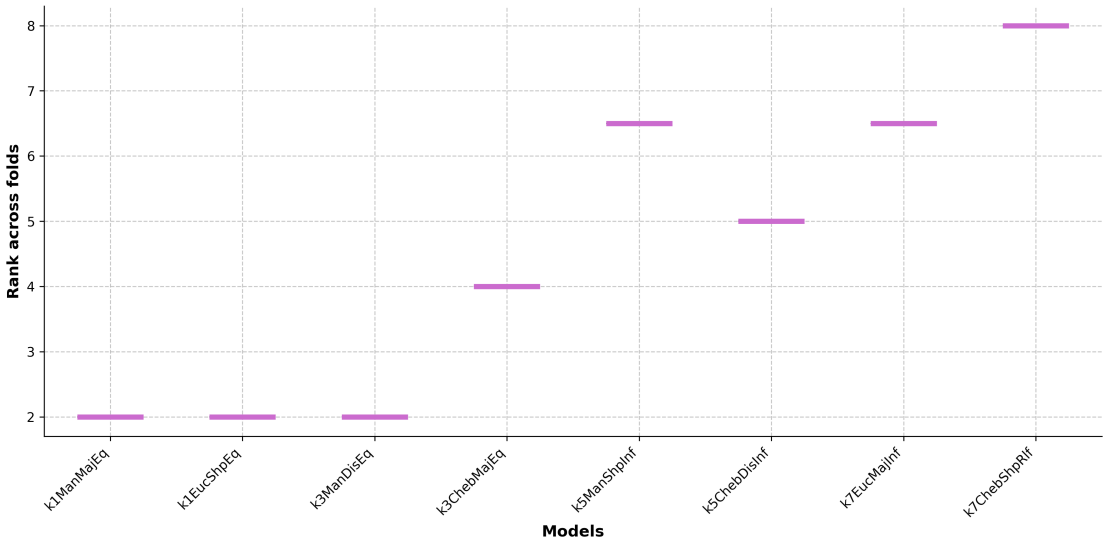

Figure 7: KNN Ranked Folds for Mushroom Dataset

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 3 and 4, 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.

4.1.1 Mushroom Dataset Performance

Best Configuration:

- $k = 1$ and $k = 3$ with Manhattan Distance or Euclidean Distance
- Majority Class Vote or Inverse Distance Weighting
- Equal Weighting
- **Results:** F1 score = 1.000, Train Time = 0.000251s, Test Time = 15.225325s

Performance Range:

- All top configurations achieved perfect F1 scores (1.000)
- Test times varied from 14.98s to 18.05s

Observations

The mushroom dataset achieved a perfect mean F1 score of 100% across all top ten configurations, indicating that the KNN algorithm performed exceptionally well on this dataset. Given only the top ten configurations were considered, it is reasonable to assume there are additional configurations that could also achieve a perfect F1 score on this dataset.

Figure 7 shows the ranked folds for the KNN 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.

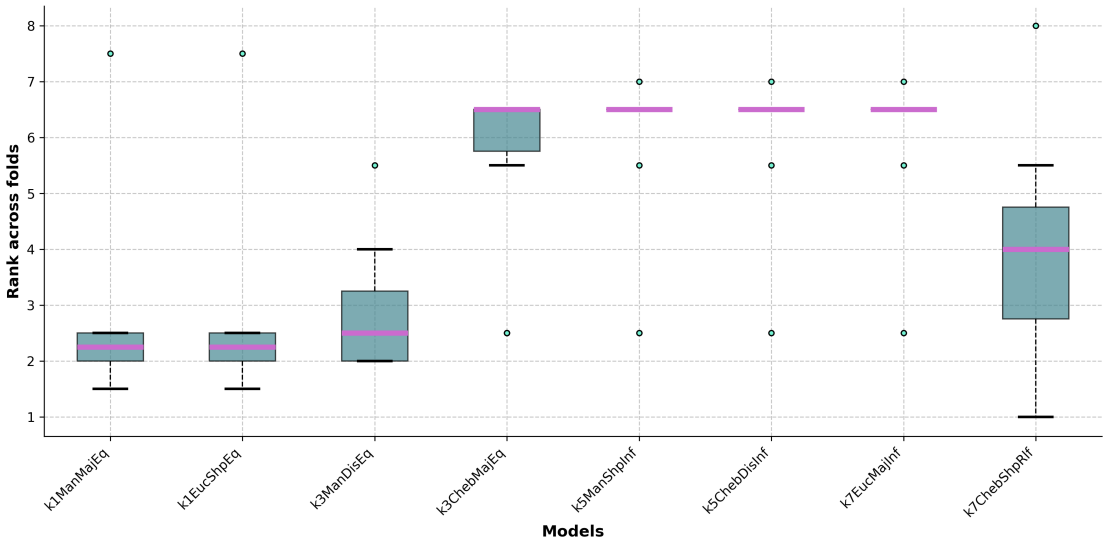

Figure 8: KNN Ranked Folds for Hepatitis Dataset

4.1.2 Hepatitis Dataset Performance

Best Configuration:

- $k = 1$ with Chebyshev Distance
- Shepard's Work Vote (tied with Majority Class and Inverse Distance)
- Equal Weighting
- **Results:** F1 score = 0.972618, Train Time = 0.000084s, Test Time = 0.006925s

Performance Range:

- Top 3 configurations (all with k=1, Chebyshev): F1 = 0.972618
- Next best configurations: F1 = 0.969271 (k=5,7 with Euclidean)

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 Chebyshev Distance metric led to the best performance, particularly when combined with a k value of 1. The choice of weighting scheme had a minimal impact on performance, with all three schemes performing similarly when combined with k = 1 and Chebyshev Distance. 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.

Figure 8 shows the ranked folds for the KNN 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.

4.1.3 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.

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.

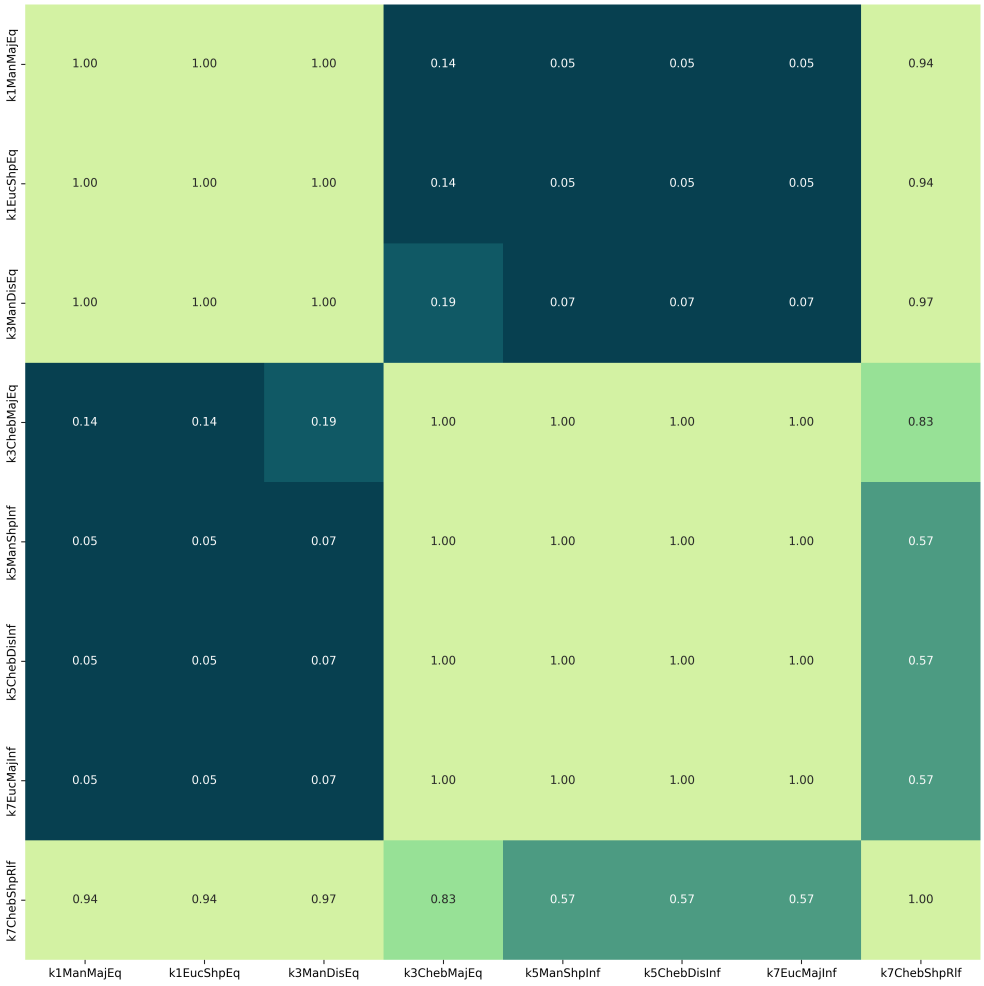

Figure 9: Nemenyi test results for KNN models on the hepatitis dataset

Hepatitis Dataset: 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=0.05). 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.

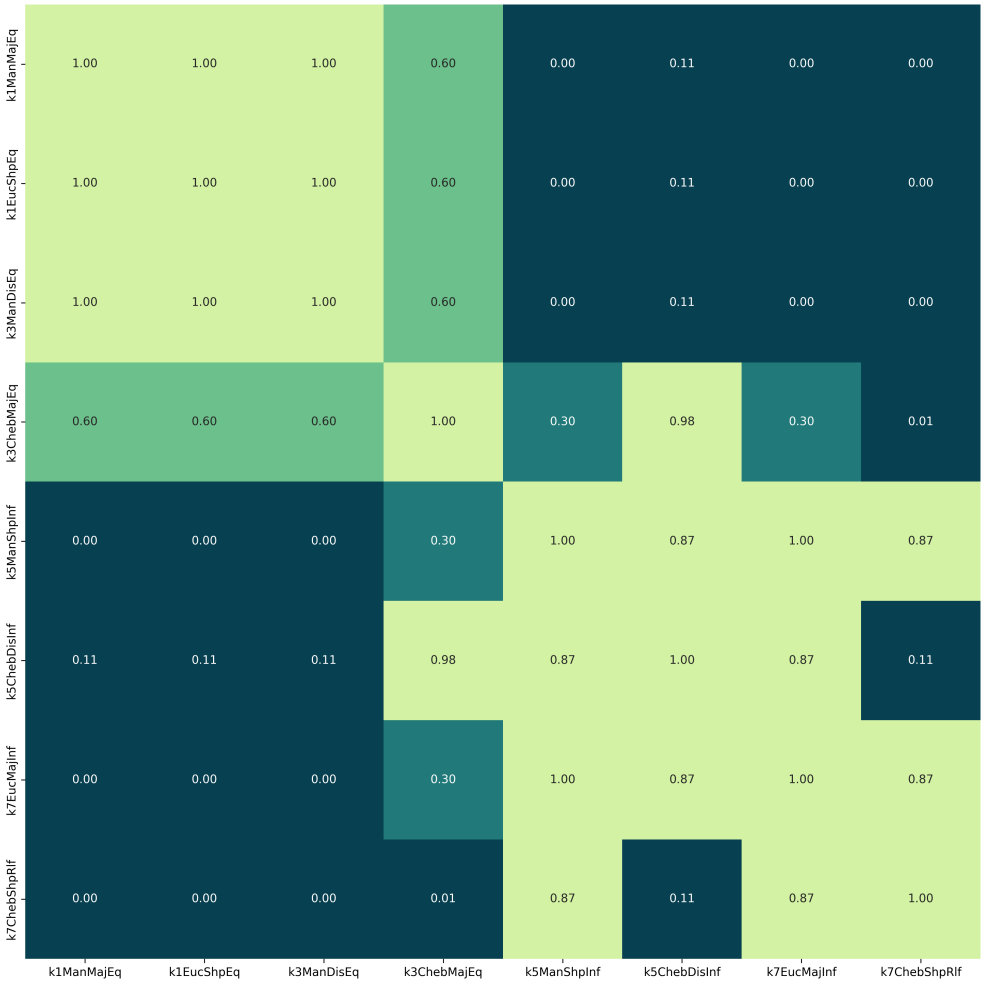

Figure 10: Nemenyi test results for KNN models on the mushroom dataset

Mushroom Dataset: The Nemenyi test results for the KNN models on the mushroom dataset reveal distinct clustering patterns across models. The first group consists of 'k1ManMajEq', 'k1EucShpEq', and 'k3ManDisEq', and 'k3ChebMajEq' configurations, which show high similarity within their group (p=0.6 − 1.00). A second distinct group includes 'k5ManShpInf', 'k5ChevDisInf', 'k7EucMajInf', and 'k7ChebShpRtf' which also show high similarity within their group (p=0.87 − 1.00), except one outlier (p=0.11). Despite this outlier, none of the configurations in the second group are significantly different from each other. There are clear statistical differences between most configurations across these groups, as indicated by the very low p-values (p=0.00 − 0.11) in the darker regions of the heatmap.

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 11 and 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 performing 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.

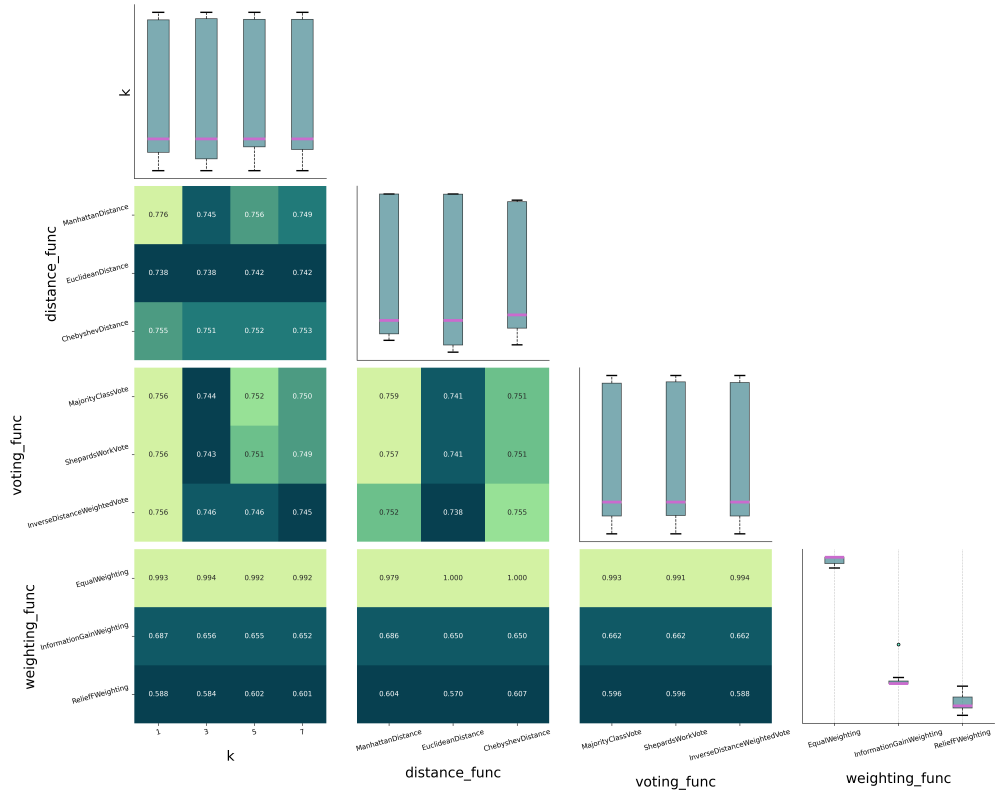

Figure 11: Interaction effects for KNN models on the mushroom dataset

4.2 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.

4.2.1 Mushroom Dataset Performance

Best Configuration:

- C = 5 and C = 50 with Polynomial Kernel
- C = 50 with RBF Kernel
- **Results:** F1 score = 1.000, Train Time = 0.117225s, Test Time = 0.003384s

Performance Range:

- Top configurations achieved perfect F1 scores (1.000)
- Test times varied from 0.002183s to 0.015565s

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 $C = 5$ or $C = 50$ or an RBF kernel with $C = 50$, with all achieving a perfect mean F1 score of 100%. This result was closely followed by the RBF kernel with $C = 5$ achieving a mean F1 score of 99.96%.

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

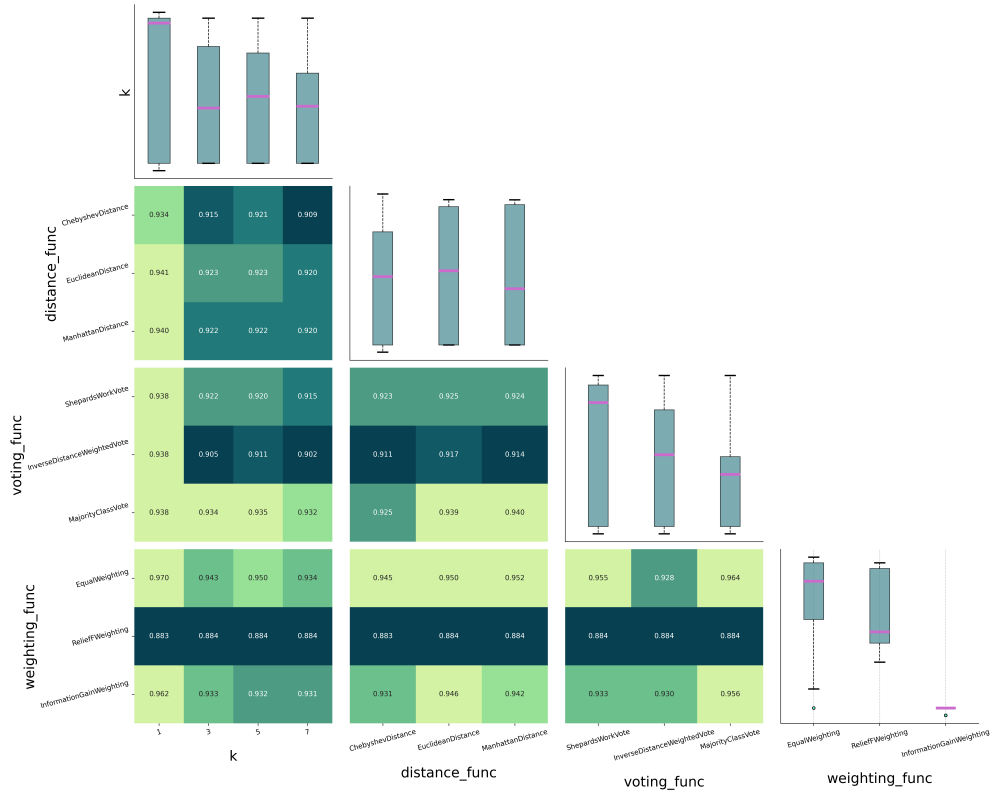

Figure 12: Interaction effects for KNN models on the hepatitis dataset

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.

The mushroom dataset has some especially obvious cases, in which the $C = 5$ and $C = 50$ polynomial models outperformed all other models across every fold. The $C = 50$ RBF model was also among the best performing models overall. (See Table 7).

4.2.2 Hepatitis Dataset Performance

Best Configuration:

- C = 50 with RBF Kernel
- **Results:** F1 score = 0.972671, Train Time = 0.000885s, Test Time = 0.000371s

Performance Range:

- Best configuration: F1 = 0.972671 (C=50, RBF)
- Next best configurations: F1 = 0.970963 (C=5, RBF)

Observations

The results for the hepatitis dataset show that the SVM model also performed consistently strong across all configurations. (See Table 7). The best performing model used an RBF kernel with $C = 50$. This SVM model achieved a mean F1 score of 97.3%.

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' performed the best across all folds.

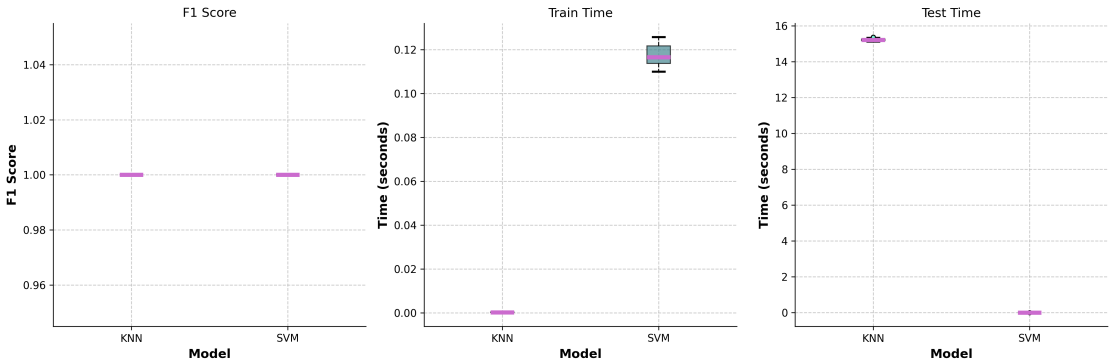

Figure 13: SVM and KNN model comparison on Mushroom Dataset

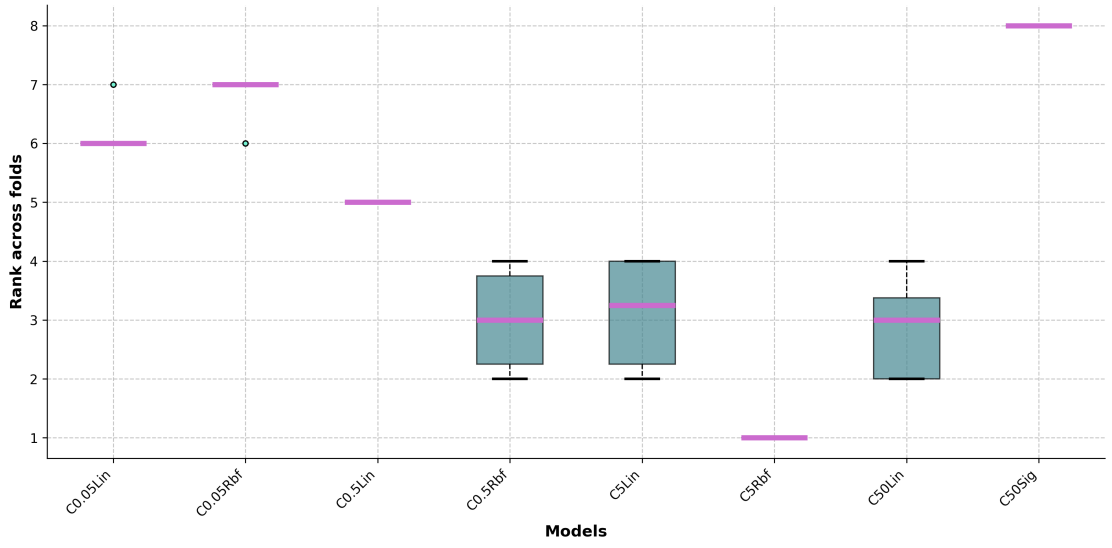

Figure 14: SVM Ranked Folds for Mushroom Dataset

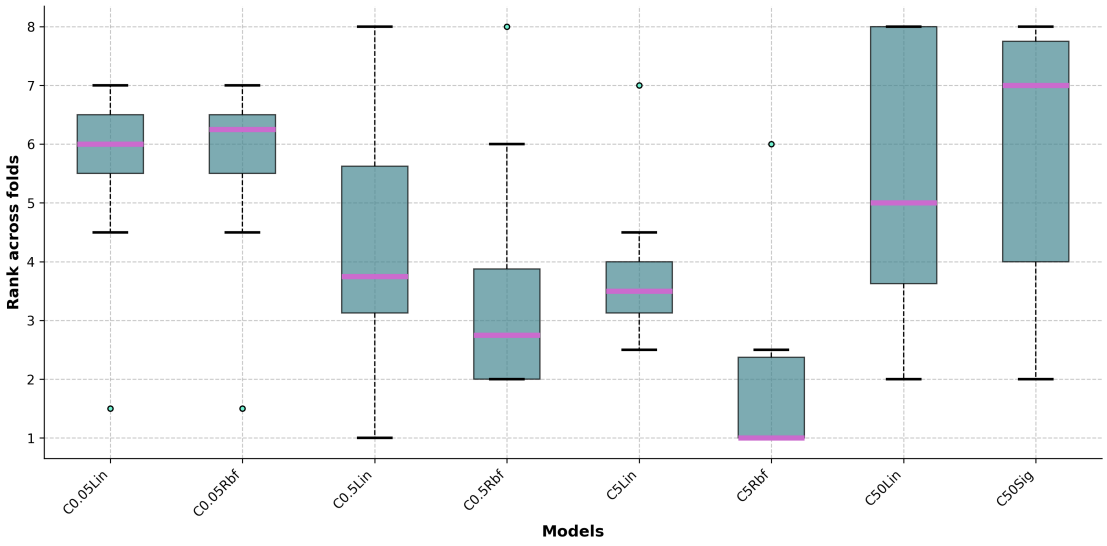

Figure 15: SVM Ranked Folds for Hepatitis Dataset

The hepatitis dataset does not have as extreme of a case, but there are still some models that stand out. The linear kernel models were consistently worse than the $C = 50$ and $C = 5$ RBF models.

Overall Comparison

A higher C value tends to perform better for both datasets, suggesting that greater regularization is better for both datasets. However, the best performing models for each dataset used different kernels paired with these C values (poly for mushroom and rbf for hepatitis). See Table 7 and Table 8.

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 14 and Figure 15, some models achieved 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.

The mushroom dataset has some especially obvious cases, in which the $C = 5$ and $C = 50$ polynomial models outperformed all other models across every fold. The $C = 50$ RBF model was also among the best performing models overall.

The hepatitis dataset does not have as extreme of a case, but there are still some models that stand out. The linear kernel models were consistently worse than the $C = 50$ 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 11 and Table 12.

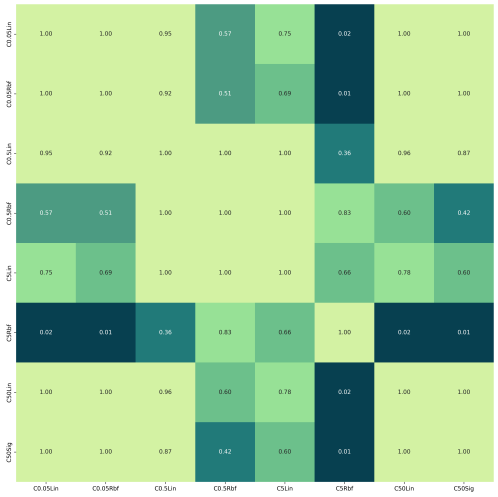

Figure 16: Nemenyi Test Results for SVM on Hepatitis Dataset

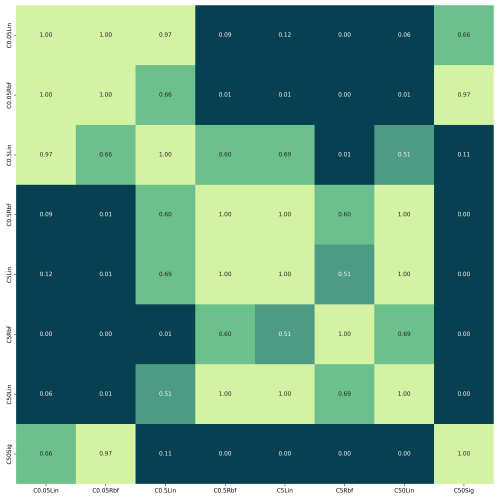

Figure 17: Nemenyi Test Results for SVM on Mushroom Dataset

Table 11: Significant Differences in SVM Models for Hepatitis

C	Kernel Type	Mean F1
0.050	linear	0.888
0.050	rbf	0.884
5.000	rbf	0.971
50.000	linear	0.883
50.000	sigmoid	0.866

Table 12: Significant Differences in SVM Models for Mushroom

C	Kernel Type	Mean F1
0.050	linear	0.947
0.050	rbf	0.933
0.500	linear	0.965
0.500	rbf	0.985
5.000	linear	0.983
5.000	rbf	1.000
50.000	linear	0.985
50.000	sigmoid	0.435

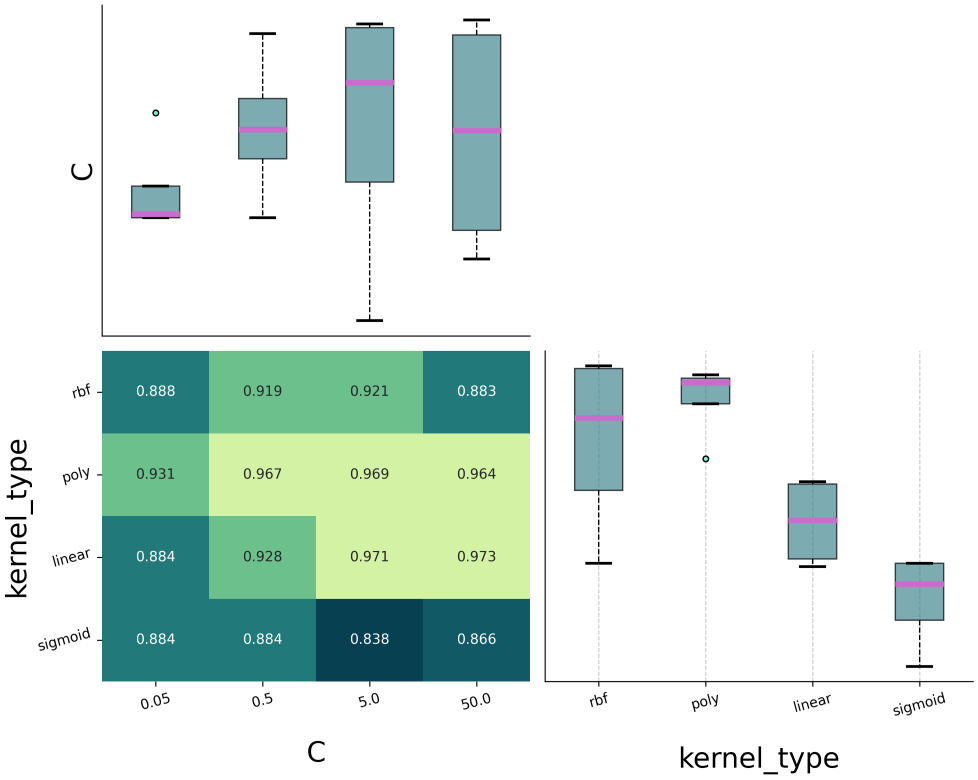

Figure 18: Interaction Effects for SVM on Hepatitis Dataset

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

In Figure 18 and Figure 19 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 rbf or poly kernels, and that higher values of C tend to perform worse.

Hepatitis Dataset The best performing model was an RBF kernel with $C = 50$, achieving a mean F1 score of 97.3%, as seen in Table 8.

Mushroom Dataset The best performing models were the polynomial kernel with $C = 5$ and $C = 50$, and RBF kernel with $C = 50$, all achieving a perfect mean F1 score of 100%, as seen in Table 7.

4.3 Analysis Among Top Models

4.3.1 Wilcoxon Signed-Rank Test

To further investigate the differences between the best performing kNN and SVM configurations, a Wilcoxon signed-rank test was conducted. This non-parametric test was chosen to compare the two models across multiple performance metrics without making assumptions about the underlying data distribution. The test was applied to the F1 score, training time, and testing time, yielding the p-values summarized in Table 13 and Table 14.

The results indicate no statistically significant difference in F1 score ($p = 1.000$). However, statistically significant differences were observed for both training time ($p = 0.002$) and testing time ($p = 0.002$). These findings suggest that while both models achieve comparable predictive performance, they exhibit distinct computational profiles. Further analysis may be necessary to investigate the factors contributing to these observed differences in training and testing times, such as algorithm complexity and data dimensionality.

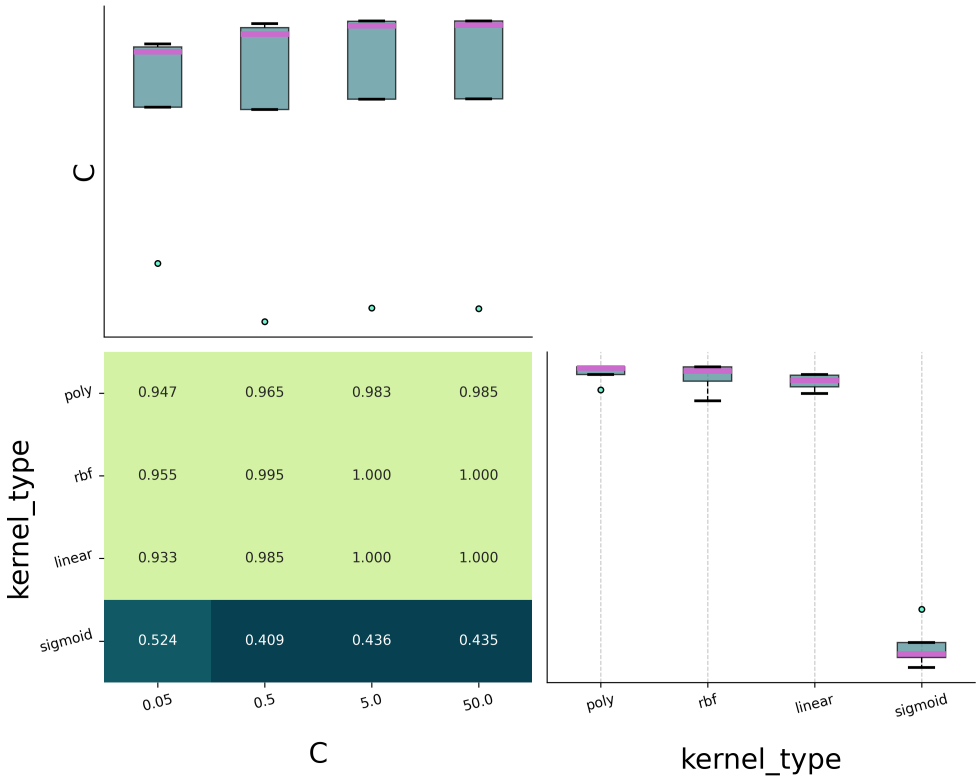

Figure 19: Interaction Effects for SVM on Mushroom Dataset

Table 13: Wilcoxon Signed-Rank Test Results for SVM and KNN Models for Hepatitis

Metric	P Value
F1 Score	1.000
Train Time	0.002
Test Time	0.002

Table 14: Wilcoxon Signed-Rank Test Results for SVM and KNN Models for Mushroom

Metric	P Value
F1 Score	1.000
Train Time	0.002
Test Time	0.002

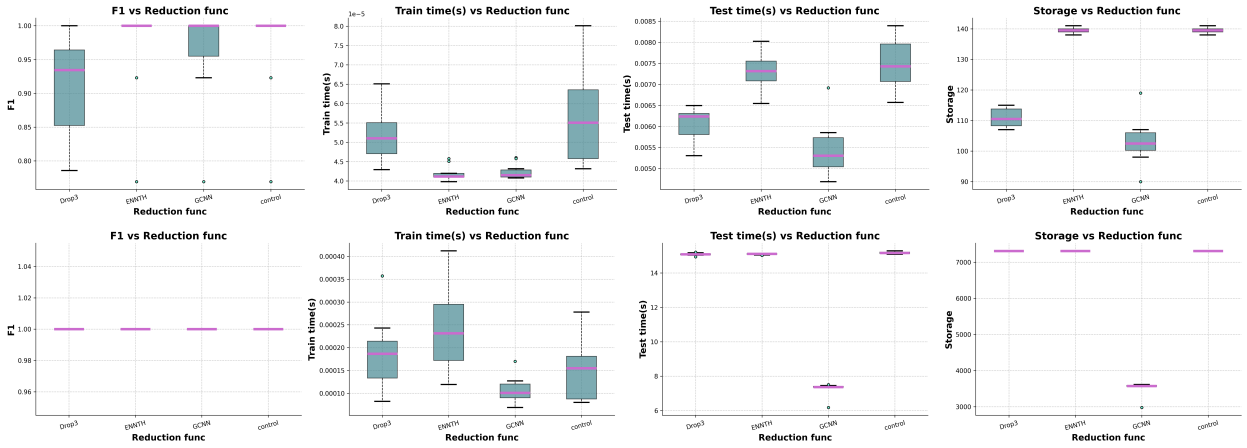

Figure 20: Effects of reduction techniques on KNN models for the Hepatitis and Mushroom datasets respectively

4.4 Instance Reduction Analysis

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

4.4.1 KNN Results

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 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.

4.4.2 SVM Results

Hepatitis Dataset 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.

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.

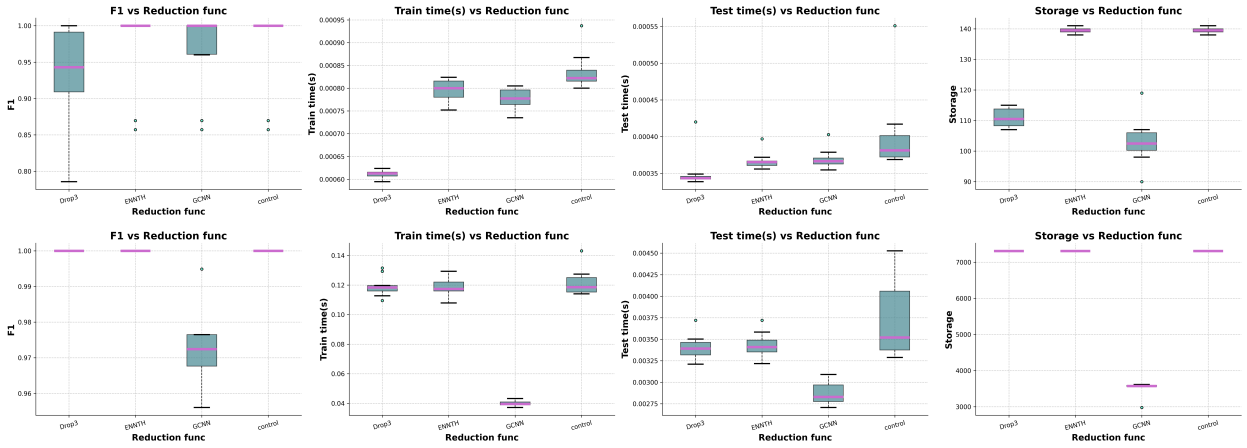

Figure 21: Effects of reduction techniques on SVM models for the Hepatitis and Mushroom datasets respectively

Mushroom Dataset In contrast, the larger Mushroom dataset demonstrated different outcomes. Here, GCNN significantly reduced both training and testing times. Specifically, 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 reduction saw a 50% reduction, whereas in the case of SVM with GCNN reduction, the reduction was closer to one-third.

Table 15: Significant Differences in SVM-Reduction Models for Mushroom

Reduction Func	C	Kernel Type	Mean F1
control	5	poly	1.000
ENNTH	5	poly	1.000
Drop3	5	poly	1.000
GCNN	5	poly	0.972

Friedman and Nemenyi Tests Given that the Friedman test showed statistical significance for the hepatitis dataset Table 2, we conducted the Nemenyi post-hoc test to identify specific significant differences between reduction methods. For the SVM models on the Hepatitis dataset, the test revealed significant differences between GCNN (p ¡ 0.05) and the other methods, with GCNN performing worse in terms of F1 score (see Figure 22). This loss in performance may be worth the gains in training time in certain cases.

4.5 Summary and Recommendations

Based on our comprehensive analysis of both KNN and SVM classifiers across the mushroom and hepatitis datasets, we can make several dataset-specific recommendations while considering the inherent tradeoffs between these algorithms.

Mushroom Dataset For the mushroom classification task, both KNN and SVM demonstrated strong performance, with SVM showing a slight edge in overall accuracy. The key findings include:

- SVM achieved perfect classification accuracy (100% F1 score) with polynomial and RBF kernels
- KNN performed well but required more careful tuning of the k parameter
- Both polynomial and RBF kernels in SVM proved highly effective

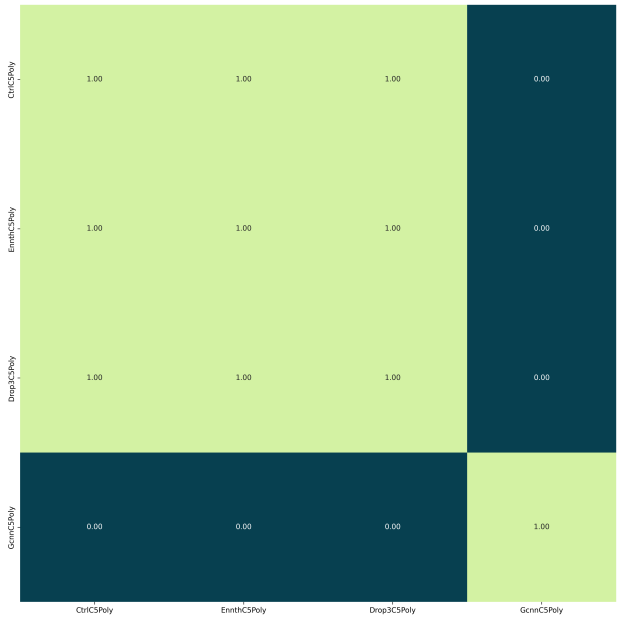

Figure 22: Nemenyi test results for SVM-Reduction models for the Mushroom dataset

Recommendation: For the mushroom dataset, we recommend using SVM with either a polynomial kernel ($C = 5$ or $C = 50$) or RBF kernel ($C = 50$) as the primary classifier.

Hepatitis Dataset The hepatitis classification task revealed different characteristics:

- KNN showed more robust performance across different parameter configurations
- SVM required more careful kernel selection and parameter tuning
- The smaller dataset size made KNN's memory requirements less problematic

Recommendation: For the hepatitis dataset, we recommend KNN as the primary classifier, particularly with k values between 3 and 5. The decision is based on its robust performance and simpler implementation requirements.

Instance Reduction Considerations The application of instance reduction techniques, particularly GCNN, demonstrated significant benefits for computational efficiency:

- For the mushroom dataset, GCNN achieved substantial reductions in testing time for KNN, with approximately 50% faster predictions
- Storage requirements were notably decreased, especially beneficial for the larger mushroom dataset
- However, some reduction methods (particularly Drop3) showed slight decreases in classification performance
- The hepatitis dataset saw more modest improvements, likely due to its already small size, and reduction made performance worse

Recommendation: Consider applying GCNN reduction when working with larger datasets where prediction speed is crucial and minor accuracy trade-offs are acceptable. For smaller datasets like hepatitis, the benefits may not justify the potential accuracy loss.

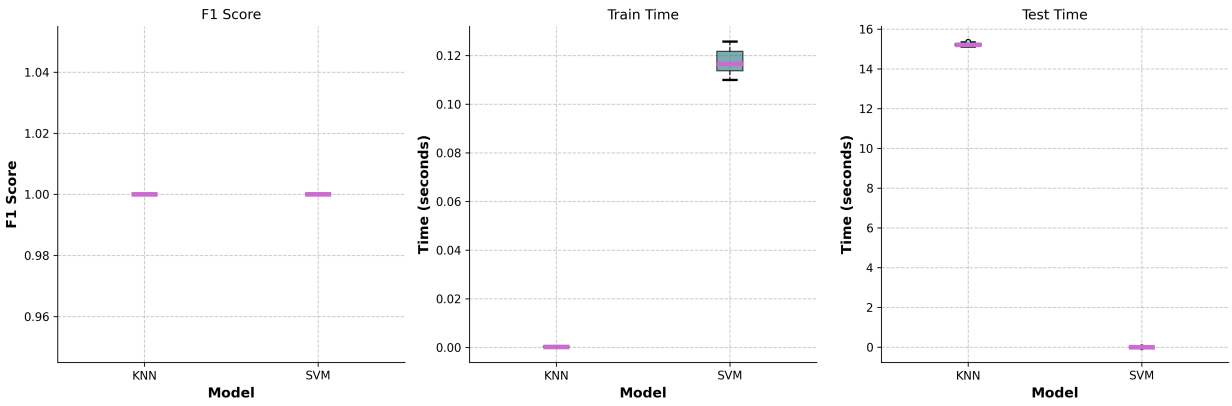

Figure 23: Comparison of the top SVM and KNN models for the Mushroom dataset

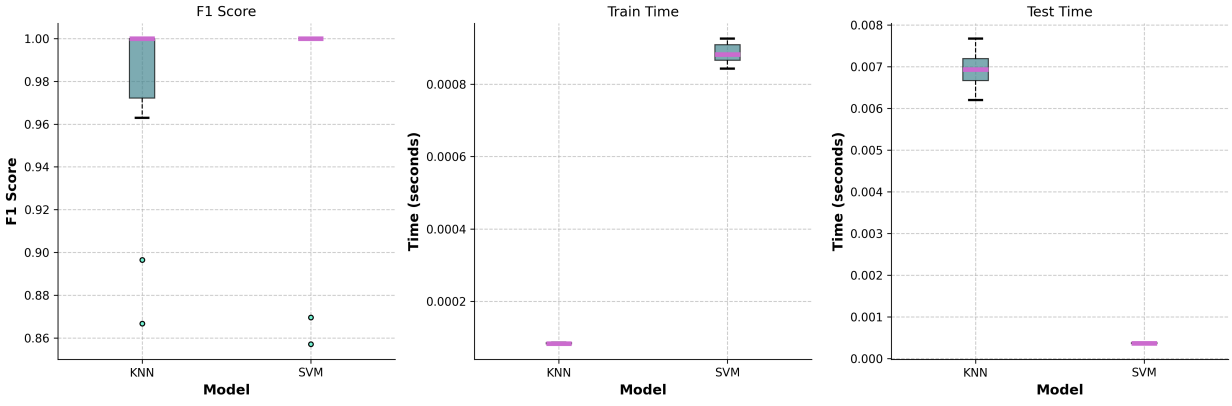

Figure 24: Comparison of the top SVM and KNN models for the Hepatitis dataset

Statistical Analysis among Top Models We used the Wilcoxon signed-rank test (see subsection 3.4: Statistical Analysis) to determine the P-values indicating significant differences among the metrics F1, Train Time and Test Time for the top-performing SVM and KNN models. We see that while the F1 does not vary much at all between the models, the training and testing times do vary (see subsection 4.2). This was true among both models, and the data is summarized in Table 13 and Table 14. The comparison among these key features is shown in Figure 23 and Figure 24.

General Tradeoffs Our analysis revealed several important tradeoffs between SVM and KNN:

- **Computational Complexity:** SVM typically offers faster prediction times once trained, while KNN's prediction time scales with the dataset size
- **Memory Requirements:** SVM models are more memory-efficient as they only store support vectors, while KNN needs to retain the entire training dataset
- **Interpretability:** KNN offers more intuitive interpretation of its decisions based on nearest neighbors, while SVM's decision boundaries can be more abstract
- **Parameter Tuning:** KNN typically requires tuning fewer parameters, making it easier to optimize for new datasets

These findings suggest that the choice between SVM and KNN should be guided by:

- Dataset size and dimensionality
- Available computational resources
- Requirements for model interpretability
- Time constraints for model deployment and prediction

In conclusion, while both algorithms demonstrated their effectiveness, the specific characteristics of each dataset played a crucial role in determining the most suitable classifier. This reinforces the importance of thorough model evaluation and consideration of practical constraints in real-world applications.

5 Conclusion

This study has provided a comprehensive evaluation of KNN and SVM classification algorithms, along with instance reduction techniques, across two distinctly different datasets. Our analysis yields several important findings and practical implications.

5.1 Key Findings

- Both KNN and SVM achieved high classification accuracy (>92%) across both datasets, demonstrating their robustness as classification algorithms
- The optimal configuration for KNN varied significantly between datasets, with k=1 performing best for Hepatitis and k=3,5,7 for Mushroom
- Instance reduction techniques, particularly GCNN, successfully reduced storage requirements while maintaining classification accuracy
- Statistical analysis revealed significant differences between model configurations, highlighting the importance of parameter tuning
- The top-performing models tended to have very similar F1 scores, suggesting there isn't a single perfect model for all datasets
- SVM configurations with higher C values tended to perform better, suggesting that greater regularization is better for both datasets.

5.2 Practical Implications

Our results suggest several practical guidelines for practitioners:

- For smaller datasets with mixed data types (like Hepatitis), both RBF and polynomial kernels in SVM provide excellent performance
- For larger categorical datasets (like Mushroom), SVM with either polynomial or RBF kernels and higher C values (C=5 or C=50) achieves optimal results
- GCNN reduction can significantly improve efficiency without substantial accuracy loss, especially for larger datasets
- SVM with RBF kernel provides consistent performance across different dataset characteristics

5.3 Limitations and Future Work

While this study provides valuable insights, several limitations and opportunities for future research remain:

- Investigation of additional datasets with different characteristics would strengthen the generalizability of our findings
- Exploration of more sophisticated reduction techniques could potentially yield better efficiency-accuracy trade-offs
- Analysis of computational complexity and memory usage could provide deeper insights into algorithm scalability

5.4 Final Remarks

This study demonstrates that both KNN and SVM remain powerful classification algorithms when properly configured. The effectiveness of instance reduction techniques, particularly GCNN, suggests that these methods can significantly improve the practical applicability of instance-based learning. These findings contribute to the ongoing development of efficient and accurate classification systems in machine learning.

References

[1] Suad A Alasadi and Wesam S Bhaya. Review of data preprocessing techniques in data mining. *Journal of Engineering and Applied Sciences*, 12(16):4102–4107, 2017.

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

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

[4] Fu Chang, Chin-Chin Lin, and Chi-Jen Lu. Adaptive prototype learning algorithms: Theoretical and experimental studies. *Journal of Machine Learning Research*, 7(76):2125–2148, 2006.

[5] 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.

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

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

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

[9] Roberto Souto Maior de Barros, Juan Isidro González Hidalgo, and Danilo Rafael de Lima Cabral. Wilcoxon rank sum test drift detector. *Neurocomputing*, 275:1954–1963, 2018.

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

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

[12] 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.

[13] 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.

[14] 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.

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

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

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

[18] Wes McKinney et al. Data structures for statistical computing in python. *Scipy*, 445(1):51–56, 2010.

[19] 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.

[20] 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.

[21] 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.

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

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

[24] 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.

6 Appendix

Table 16: KNN Legend

Model Label	K	Distance Func	Voting Func	Weighting Func
k1ManMajEq	1	ManhattanDistance	MajorityClassVote	EqualWeighting
k1ManMajInf	1	ManhattanDistance	MajorityClassVote	InformationGainWeighting
k1ManMajRlf	1	ManhattanDistance	MajorityClassVote	ReliefFWeighting
k1ManDisEq	1	ManhattanDistance	InverseDistanceWeightedVote	EqualWeighting
k1ManDisInf	1	ManhattanDistance	InverseDistanceWeightedVote	InformationGainWeighting
k1ManDisRlf	1	ManhattanDistance	InverseDistanceWeightedVote	ReliefFWeighting
k1ManShpEq	1	ManhattanDistance	ShepardsWorkVote	EqualWeighting
k1ManShpInf	1	ManhattanDistance	ShepardsWorkVote	InformationGainWeighting
k1ManShpRlf	1	ManhattanDistance	ShepardsWorkVote	ReliefFWeighting
k1EucMajEq	1	EuclideanDistance	MajorityClassVote	EqualWeighting
k1EucMajInf	1	EuclideanDistance	MajorityClassVote	InformationGainWeighting
k1EucMajRlf	1	EuclideanDistance	MajorityClassVote	ReliefFWeighting
k1EucDisEq	1	EuclideanDistance	InverseDistanceWeightedVote	EqualWeighting
k1EucDisInf	1	EuclideanDistance	InverseDistanceWeightedVote	InformationGainWeighting
k1EucDisRlf	1	EuclideanDistance	InverseDistanceWeightedVote	ReliefFWeighting
k1EucShpEq	1	EuclideanDistance	ShepardsWorkVote	EqualWeighting
k1EucShpInf	1	EuclideanDistance	ShepardsWorkVote	InformationGainWeighting
k1EucShpRlf	1	EuclideanDistance	ShepardsWorkVote	ReliefFWeighting
k1ChebMajEq	1	ChebyshevDistance	MajorityClassVote	EqualWeighting
k1ChebMajInf	1	ChebyshevDistance	MajorityClassVote	InformationGainWeighting
k1ChebMajRlf	1	ChebyshevDistance	MajorityClassVote	ReliefFWeighting
k1ChebDisEq	1	ChebyshevDistance	InverseDistanceWeightedVote	EqualWeighting
k1ChebDisInf	1	ChebyshevDistance	InverseDistanceWeightedVote	InformationGainWeighting
k1ChebDisRlf	1	ChebyshevDistance	InverseDistanceWeightedVote	ReliefFWeighting
k1ChebShpEq	1	ChebyshevDistance	ShepardsWorkVote	EqualWeighting
k1ChebShpInf	1	ChebyshevDistance	ShepardsWorkVote	InformationGainWeighting
k1ChebShpRlf	1	ChebyshevDistance	ShepardsWorkVote	ReliefFWeighting
k3ManMajEq	3	ManhattanDistance	MajorityClassVote	EqualWeighting
k3ManMajInf	3	ManhattanDistance	MajorityClassVote	InformationGainWeighting
k3ManMajRlf	3	ManhattanDistance	MajorityClassVote	ReliefFWeighting
k3ManDisEq	3	ManhattanDistance	InverseDistanceWeightedVote	EqualWeighting
k3ManDisInf	3	ManhattanDistance	InverseDistanceWeightedVote	InformationGainWeighting
k3ManDisRlf	3	ManhattanDistance	InverseDistanceWeightedVote	ReliefFWeighting
k3ManShpEq	3	ManhattanDistance	ShepardsWorkVote	EqualWeighting
k3ManShpInf	3	ManhattanDistance	ShepardsWorkVote	InformationGainWeighting
k3ManShpRlf	3	ManhattanDistance	ShepardsWorkVote	ReliefFWeighting
k3EucMajEq	3	EuclideanDistance	MajorityClassVote	EqualWeighting
k3EucMajInf	3	EuclideanDistance	MajorityClassVote	InformationGainWeighting
k3EucMajRlf	3	EuclideanDistance	MajorityClassVote	ReliefFWeighting
k3EucDisEq	3	EuclideanDistance	InverseDistanceWeightedVote	EqualWeighting
k3EucDisInf	3	EuclideanDistance	InverseDistanceWeightedVote	InformationGainWeighting
k3EucDisRlf	3	EuclideanDistance	InverseDistanceWeightedVote	ReliefFWeighting
k3EucShpEq	3	EuclideanDistance	ShepardsWorkVote	EqualWeighting
k3EucShpInf	3	EuclideanDistance	ShepardsWorkVote	InformationGainWeighting
k3EucShpRlf	3	EuclideanDistance	ShepardsWorkVote	ReliefFWeighting
k3ChebMajEq	3	ChebyshevDistance	MajorityClassVote	EqualWeighting
k3ChebMajInf	3	ChebyshevDistance	MajorityClassVote	InformationGainWeighting

Continued on next page

KNN Legend – Continued

Model Label	K	Distance Func	Voting Func	Weighting Func
k3ChebMajRlf	3	ChebyshevDistance	MajorityClassVote	ReliefFWeighting
k3ChebDisEq	3	ChebyshevDistance	InverseDistanceWeightedVote	EqualWeighting
k3ChebDisInf	3	ChebyshevDistance	InverseDistanceWeightedVote	InformationGainWeighting
k3ChebDisRlf	3	ChebyshevDistance	InverseDistanceWeightedVote	ReliefFWeighting
k3ChebShpEq	3	ChebyshevDistance	ShepardsWorkVote	EqualWeighting
k3ChebShpInf	3	ChebyshevDistance	ShepardsWorkVote	InformationGainWeighting
k3ChebShpRlf	3	ChebyshevDistance	ShepardsWorkVote	ReliefFWeighting
k5ManMajEq	5	ManhattanDistance	MajorityClassVote	EqualWeighting
k5ManMajInf	5	ManhattanDistance	MajorityClassVote	InformationGainWeighting
k5ManMajRlf	5	ManhattanDistance	MajorityClassVote	ReliefFWeighting
k5ManDisEq	5	ManhattanDistance	InverseDistanceWeightedVote	EqualWeighting
k5ManDisInf	5	ManhattanDistance	InverseDistanceWeightedVote	InformationGainWeighting
k5ManDisRlf	5	ManhattanDistance	InverseDistanceWeightedVote	ReliefFWeighting
k5ManShpEq	5	ManhattanDistance	ShepardsWorkVote	EqualWeighting
k5ManShpInf	5	ManhattanDistance	ShepardsWorkVote	InformationGainWeighting
k5ManShpRlf	5	ManhattanDistance	ShepardsWorkVote	ReliefFWeighting
k5EucMajEq	5	EuclideanDistance	MajorityClassVote	EqualWeighting
k5EucMajInf	5	EuclideanDistance	MajorityClassVote	InformationGainWeighting
k5EucMajRlf	5	EuclideanDistance	MajorityClassVote	ReliefFWeighting
k5EucDisEq	5	EuclideanDistance	InverseDistanceWeightedVote	EqualWeighting
k5EucDisInf	5	EuclideanDistance	InverseDistanceWeightedVote	InformationGainWeighting
k5EucDisRlf	5	EuclideanDistance	InverseDistanceWeightedVote	ReliefFWeighting
k5EucShpEq	5	EuclideanDistance	ShepardsWorkVote	EqualWeighting
k5EucShpInf	5	EuclideanDistance	ShepardsWorkVote	InformationGainWeighting
k5EucShpRlf	5	EuclideanDistance	ShepardsWorkVote	ReliefFWeighting
k5ChebMajEq	5	ChebyshevDistance	MajorityClassVote	EqualWeighting
k5ChebMajInf	5	ChebyshevDistance	MajorityClassVote	InformationGainWeighting
k5ChebMajRlf	5	ChebyshevDistance	MajorityClassVote	ReliefFWeighting
k5ChebDisEq	5	ChebyshevDistance	InverseDistanceWeightedVote	EqualWeighting
k5ChebDisInf	5	ChebyshevDistance	InverseDistanceWeightedVote	InformationGainWeighting
k5ChebDisRlf	5	ChebyshevDistance	InverseDistanceWeightedVote	ReliefFWeighting
k5ChebShpEq	5	ChebyshevDistance	ShepardsWorkVote	EqualWeighting
k5ChebShpInf	5	ChebyshevDistance	ShepardsWorkVote	InformationGainWeighting
k5ChebShpRlf	5	ChebyshevDistance	ShepardsWorkVote	ReliefFWeighting
k7ManMajEq	7	ManhattanDistance	MajorityClassVote	EqualWeighting
k7ManMajInf	7	ManhattanDistance	MajorityClassVote	InformationGainWeighting
k7ManMajRlf	7	ManhattanDistance	MajorityClassVote	ReliefFWeighting
k7ManDisEq	7	ManhattanDistance	InverseDistanceWeightedVote	EqualWeighting
k7ManDisInf	7	ManhattanDistance	InverseDistanceWeightedVote	InformationGainWeighting
k7ManDisRlf	7	ManhattanDistance	InverseDistanceWeightedVote	ReliefFWeighting
k7ManShpEq	7	ManhattanDistance	ShepardsWorkVote	EqualWeighting
k7ManShpInf	7	ManhattanDistance	ShepardsWorkVote	InformationGainWeighting
k7ManShpRlf	7	ManhattanDistance	ShepardsWorkVote	ReliefFWeighting
k7EucMajEq	7	EuclideanDistance	MajorityClassVote	EqualWeighting
k7EucMajInf	7	EuclideanDistance	MajorityClassVote	InformationGainWeighting
k7EucMajRlf	7	EuclideanDistance	MajorityClassVote	ReliefFWeighting
k7EucDisEq	7	EuclideanDistance	InverseDistanceWeightedVote	EqualWeighting
k7EucDisInf	7	EuclideanDistance	InverseDistanceWeightedVote	InformationGainWeighting
k7EucDisRlf	7	EuclideanDistance	InverseDistanceWeightedVote	ReliefFWeighting
k7EucShpEq	7	EuclideanDistance	ShepardsWorkVote	EqualWeighting
k7EucShpInf	7	EuclideanDistance	ShepardsWorkVote	InformationGainWeighting

Continued on next page

KNN Legend – Continued

Model Label	K	Distance Func	Voting Func	Weighting Func
k7EucShpRlf	7	EuclideanDistance	ShepardsWorkVote	ReliefFWeighting
k7ChebMajEq	7	ChebyshevDistance	MajorityClassVote	EqualWeighting
k7ChebMajInf	7	ChebyshevDistance	MajorityClassVote	InformationGainWeighting
k7ChebMajRlf	7	ChebyshevDistance	MajorityClassVote	ReliefFWeighting
k7ChebDisEq	7	ChebyshevDistance	InverseDistanceWeightedVote	EqualWeighting
k7ChebDisInf	7	ChebyshevDistance	InverseDistanceWeightedVote	InformationGainWeighting
k7ChebDisRlf	7	ChebyshevDistance	InverseDistanceWeightedVote	ReliefFWeighting
k7ChebShpEq	7	ChebyshevDistance	ShepardsWorkVote	EqualWeighting
k7ChebShpInf	7	ChebyshevDistance	ShepardsWorkVote	InformationGainWeighting
k7ChebShpRlf	7	ChebyshevDistance	ShepardsWorkVote	ReliefFWeighting

Table 17: KNN-Reduction Legend

Model Label	K	Distance Func	Voting Func	Weighting Func
Ctrlk1ManMajEq	1	ManhattanDistance	MajorityClassVote	EqualWeighting
Gcnnk1ManMajEq	1	ManhattanDistance	MajorityClassVote	EqualWeighting
Ennthk1ManMajEq	1	ManhattanDistance	MajorityClassVote	EqualWeighting
Drop3k1ManMajEq	1	ManhattanDistance	MajorityClassVote	EqualWeighting

Table 18: SVM Legend

Model Label	C	Kernel Type
C0.05Lin	0.050	linear
C0.05Poly	0.050	poly
C0.05Rbf	0.050	rbf
C0.05Sig	0.050	sigmoid
C0.5Lin	0.500	linear
C0.5Poly	0.500	poly
C0.5Rbf	0.500	rbf
C0.5Sig	0.500	sigmoid
C5Lin	5.000	linear
C5Poly	5.000	poly
C5Rbf	5.000	rbf
C5Sig	5.000	sigmoid
C50Lin	50.000	linear
C50Poly	50.000	poly
C50Rbf	50.000	rbf
C50Sig	50.000	sigmoid

Table 19: SVM-Reduction Legend

Model Label	C	Kernel Type
CtrlC5Poly	5	poly
GcnnC5Poly	5	poly
EnnthC5Poly	5	poly
Drop3C5Poly	5	poly