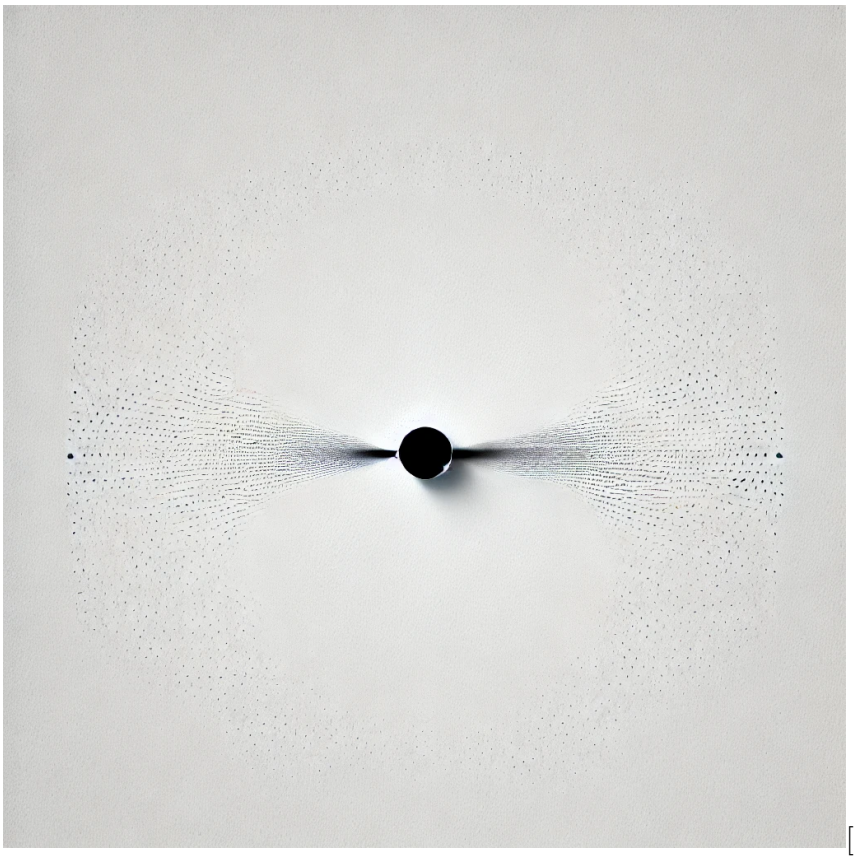
Classification with Lazy Learning and SVM


[1]

Oriol Miró López-Feliu
Marc Gonzalez Vidal
Julia Amenós Dien
Joan Caballero Castro

November 2024

Introduction to Machine Learning (IML)
Practical Work 2: W2
MAI - UPC / UB / URV

Contents

1 Introduction

In dynamic, variable data environments, eager learning's reliance on a fixed model can become a drawback, as frequent retraining is needed to remain accurate; in such scenarios, lazy learning, which delays generalisation until prediction, can become an attractive choice. But to what extent does either method——eager or lazy——demonstrate superiority? This study examines the performance of the K-Nearest Neighbors (KNN) algorithm and Support Vector Machines (SVM), lazy and eager respectively, across two well-established UCI datasets [2].

We begin by implementing KNN from scratch, exploring various parameter configurations, such as distance metrics and voting policies. Later, we conduct statistical analysis to identify the optimal configuration for each dataset—counting for accuracy and efficiency—, named "the best KNN". Similarly, though without implementing it from scratch, we identify the optimal configuration for SVM on each dataset, designating it as the "best SVM". The performance of these optimal configurations is then systematically compared.

A major drawback lazy learning faces is the need to store the entire dataset, which can quickly become computationally prohibitive; for this purpose, instance reduction techniques are often employed. We implemented various of these from scratch, and applied them as a preprocessing step before running KNN and SVM. We then explore the impact of these in terms of accuracy, efficiency, and storage reduction, through rigorous statistical analysis.

Ultimately, this study provides a comparison between the best configurations of SVM and KNN (both with and without instance reduction) and culminates in a discussion of all relevant findings; we also offer insights into the practical benefits and limitations of eager versus lazy learning approaches in machine learning applications.

2 Datasets

In this project, we restricted our dataset selection to a subset of those available in the UCI Machine Learning Repository [2], from which we selected two datasets. The only criteria specified were that one dataset must include both categorical and numeric variables, and one should be of medium to large size while the other could be smaller. For our analysis, we chose to evaluate the algorithms on the Balance [3] and Sick datasets.

2.1 Balance

This dataset models a balance scale to which a weight is added at a certain distance, with the goal of predicting whether it will tilt left, tilt right, or remain balanced. It includes four features and a class label: the left weight, left distance, right weight, and right distance. The class is determined by comparing (left_distance$\times$left_weight) and (right_distance$\times$right_weight); if these are equal, the scale remains balanced. Both weight and distance values are within $\{1, 2, 3, 4, 5\}$, and the class is in $\{L, B, R\}$. The dataset contains 625 instances without missing values.

2.1.1 Preprocessing

Since all the features in the dataset are categorical, One-Hot Encoding[4] is applied to convert each category into distinct binary columns. The initial four attributes, each with five possible categories, are expanded into a total of 20 binary columns.

The categorical class labels were also encoded as numerical values, mapped as follows:

$$f(x) = \begin{cases} 0, & \text{if } x = L \\ 1, & \text{if } x = R \\ 2, & \text{if } x = B \end{cases}$$

2.2 Sick

The second dataset models clinical information for diagnosing thyroid diseases. It contains patient information, including medical indicators such as hormone levels (TSH, T3, TT4, T4U, FTI). The primary objective is to classify individuals as either being healthy or having a thyroid-related condition.

2.2.1 Preprocessing

The Sick Dataset required a more extensive preprocessing. The data contained binary attributes indicating whether certain measurements were recorded; for instance, the TSH_measure column simply noted if the TSH variable was measured. As these columns only confirmed the presence or absence of data without providing additional insights, they were removed. Furthermore, the TBG feature, which was null across all samples, was also excluded from the dataset.

Samples with missing values in any attribute were removed rather than being imputed. This decision was guided by a focus on preserving the original data. Medical datasets reflects patients' precise health conditions, and filling in values through imputation could introduce potential biases or inaccuracies that may risk misrepresentation. Therefore, imputation techniques may compromise data integrity and lead to incorrect diagnoses. Although this resulted in the removal of 30% of the instances, it ensures the remaining data represents reliable cases. We proceeded with this approach after verifying that the models were able to capture key patterns and generalise accurately.

It is worth noting that removing instances with missing values led to a slightly more balanced dataset. The distribution of the original dataset was 93.88% negative and 6.12% positive cases. After preprocessing, it shifted to 91.97% negative and 9.27% positive cases.

Numerical attributes were normalised between [0, 1], in order to promote feature uniformity, and categorical data was transformed using One-Hot Encoding. Finally, the target variable was mapped into a binary format:

$$f(x) = \begin{cases} 0, & \text{if } x = \textit{negative} \\ 1, & \text{if } x = \textit{sick} \end{cases}$$

3 Evaluation Technique

In this work we are going to evaluate two different metrics from the experiments, that will be *accuracy* (i.e. correctly classified instances), and *efficiency* (i.e. training and inference time). For the instance reduction experiments, we also use the *storage* (i.e. the average percentage of cases stored from the training to generalize the testing cases).

To ensure a rigorous evaluation, the dataset is divided into training and test sets. The training set will be used to develop the model's predictive capabilities, while the test set will be used exclusively for evaluation, specifically to compute the *accuracy*. Given that our dataset is not extensive and that we can accommodate the computational cost required (and are also required to do so), we will employ Cross-Validation [5] to enhance the robustness of our results. Predefined folds are provided to standardize comparisons across groups and professors, ensuring consistency in evaluation.

Therefore, to evaluate our algorithms, for each fold we will compute the accuracy, along with the training and inference time. For the instance reduction techniques, we will also calculate the percentage of the dataset retained after the reduction. A detailed discussion of the evaluation of the algorithms will be described in subsequent sections.

Please note that, given the inherent imbalance in our datasets, it would be more prudent to utilise a metric such as the F1-Score [6] rather than accuracy. The F1-Score would provide a more accurate representation of performance, as it would reveal instances where the model may

predominantly predict the majority class. However, the project description specifies adherence to classification accuracy as the evaluation metric.

4 KNN Algorithm

As previously stated, the lazy algorithm we are going to use is K-Nearest Neighbors (KNN) [7]. This algorithm works by identifying the k most similar instances in the training set to the instance we want to predict, using a specified distance metric. Once these k nearest neighbours are identified, each of them contributes differently to the prediction outcome based on the voting policy we use.

4.1 KNN Components

The KNN algorithm relies on three primary components that significantly influence its performance and adaptability to various datasets. These are the distance metrics, voting policies, and feature weighting. Note that the selection of the hyperparameter k is also crucial, as will be demonstrated in the results. However, since it does not require any modifications to the algorithm's implementation, it is not discussed in this section.

4.1.1 Distance Metrics

The distance metrics are the algorithms that will determine similarity between instances. In this work, we were restricted to use Minkowski distance [8][9] with the hyperparameter $r = 1$ and $r = 2$ and one more decided by us.

$$d_M(X, Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^r \right)^{\frac{1}{r}} \qquad \text{(Minkowski distance)}$$

During the process of choosing one distance metric more, we discarded *Manhattan* distance and *Euclidean* distance due to their equivalence to Minkowski when $r = 1$ and $r = 2$, respectively [9]. Further exploration was done, and three more metrics were considered: Mahalanobis [10], Cosine [11], and Hamming [12] distances.

Mahalanobis distance was dismissed due to its reliance on the existence of correlations between features, which also increases the computational time of the algorithm. Cosine distance was rejected because of potential issues in calculating the Cosine similarity. This measure is designed to work with sparse vectors, meaning it is not suitable for vectors containing all zeros. Given that our dataset may contain such cases, this limitation prevents us from reliably computing the formula.

Considering this, we decided to use the Hamming distance, as it tends to cause fewer issues and appears more suitable for our needs. Additionally, an aspect that appealed to us was its simplicity as a distance metric, and we aim to demonstrate that, in certain cases, simpler solutions can indeed be more effective.

$$d_H(X, Y) = \sum_{i=1}^{n} (x_i \neq y_i) \qquad \text{(Hamming distance)}$$

It is essential to highlight several aspects regarding our expectations for this metric. Firstly, it is important to note that for purely categorical datasets, such as the *Balance* dataset, the Hamming distance is entirely equivalent to the Manhattan distance (i.e., Minkowski distance with $r = 1$). This equivalence arises from our use of One-Hot Encoding, which ensures that all features are represented as either 0 or 1. Consequently, the Manhattan distance will yield a value of 0 when the feature values are the same and 1 when they differ, mirroring the behaviour

of the Hamming distance. However, this equivalence only holds when equal weighting is applied to the features, a topic that will be elaborated upon later. Additionally, it is important to recognise that certain feature weightings and voting policies will not affect the order of the nearest neighbours in the space, as the Hamming distance is invariant to linear scaling of features.

One may question the rationale behind employing this metric if it is invariant or equivalent to other metrics we utilise. The response is that this metric is likely faster to compute than the alternatives. If we can achieve comparable performance with a reduced computational cost, we stand to benefit significantly.

4.1.2 Voting Policies

Voting policies delineate how the classes of neighbouring instances contribute to the final class prediction. In this aspect of the algorithm's design, we were not afforded any flexibility, as three specific policies were established for the study.

- Majority Class: The class assigned is determined by the one that holds the highest representation among the closest neighbours.
- Inverse Distance Weighted votes: This method assigns weights to the decisions proportionally based on their distances. In other words, closer neighbours exert greater voting influence compared to those that are farther away.
- Sheppard's work: This approach operates on the same principle as Inverse Distance Weighted votes; however, instead of adhering to linear proportionality, it follows an exponential relationship.

All three approaches may result in ties when k is even. The tie-breaking policy assigns the class of the nearest neighbour.

4.1.3 Feature Weighting

Feature weighting enables the assignment of varying levels of importance to features, typically based on their relevance to the target class. In this project, we are required to implement three distinct feature weighting strategies, with only one restriction: one strategy must use equal weighting, while the other two can be freely chosen. The strategies are as follows:

- Equal Weight: All features weight the same. In other words, no change to the features is needed.
- Information Gain: Measures the decrease in entropy after a dataset is split on an attribute. It's used to identify features that provide the most significant amount of information, helping to reduce uncertainty in predictions.
- χ^2: It evaluates the dependency between each feature and the target variable by calculating the chi-squared statistic. Features with higher χ^2 scores are considered more informative.

4.2 KNN Implementation

In this section, we will discuss the implementation of the techniques presented above. We have chosen to separate this into a distinct section to avoid delving into the implementation details within the theoretical part.

Firstly, one of the most critical decisions made was to utilise NumPy arrays within KNN algorithm instead of employing other data structures, such as Python lists or Pandas DataFrames.

This choice significantly enhances the execution speed of the experiments compared to using more conventional data structures for the data we are utilising. For the sake of usability, the KNN algorithm will receive and return data as DataFrames; however, as previously mentioned, it will operate internally with NumPy arrays.

There are only two methods with which the user will interact: train and predict. In the train method, we will store both the training set and the labels within the class of the algorithm, employing a lazy learning approach. Additionally, we will save the information necessary to convert the labels to integers and vice versa, as some of the methods utilised do not support categorical labels. We will also compute the feature weights as requested. This step is crucial, as we will not utilise any information from the test set during the learning process; therefore, the feature weights must be derived from the training set and stored for use during inference. The predict method will execute the KNN algorithm using the specified voting policy and distance metric, returning the predicted labels as a Pandas Series.

For the implementation of the distance metrics and voting policies, we have directly implemented the mathematical formulas as provided in the description of the work and the referenced sources, without optimising them in a vectorised manner. This decision was made to maintain readability, as the computational cost of calling these functions is not prohibitive because of the use of NumPy arrays.

For the implementation of feature weighting, we utilised pre-existing versions of information gain and the χ^2 metric, as permitted, from the scikit-learn package [13]. Specifically, we employed the *mutual_info_classif* function for information gain and the *chi2* function for the χ^2 metric, both from the *feature_selection* section.

4.3 Analysis of the best KNN configuration

We aim to identify the best KNN configuration for each dataset by considering both classification accuracy and computational efficiency. To balance these two metrics, we first normalise the computational time for each configuration to the range [0, 1] using min-max normalisation, with 1 representing the maximum observed time among all configurations. Then, we define a composite metric M_{config} for each configuration as:

$$M_{\text{config}} = \alpha \times A_{\text{config}} + \beta \times (1 - T_{\text{config}}) \tag{1}$$
$$\text{where} \quad \alpha + \beta = 1$$

A_{config} is the classification accuracy of the configuration, T_{config} is the normalised execution time, and α and β are weighting factors for the relative importance accuracy and efficiency. In our case, we set $\alpha = 0.7$ and $\beta = 0.3$, as we deemed the former more important.

Although accuracy values are already proportions between 0 and 1, in our datasets in some cases the range of accuracy values is relatively narrow (e.g. $[0.78, 0.97]$ for the Sick dataset). To ensure both metrics contribute proportionally, we normalised the accuracy values to the $[0, 1]$ range using min-max normalisation. This prevents larger times from disproportionately influencing M_{config}, and ensures a fair comparison [14, 15].

Given the large number of configurations (108 in total) and the small number of datasets (only two), directly applying statistical tests may lead to numerical instability [16]. Additionally, these tests typically require at least ten datasets to provide reliable results. To mitigate these issues, we opt for the following relaxations:

1. **Augmenting the Number of Datasets**: We consider each cross-validation fold as an independent dataset, resulting in $N = 10$ datasets per original dataset.

2. **Reducing the Number of Configurations**: We rank all configurations based on the composite metric M_{config} and select the top eight configurations for statistical analysis.

We acknowledge that these adjustments introduce violations of certain statistical assumptions, such as the independence of datasets and the avoidance of selection bias (when filtering by top configurations); however, these compromises are necessary given the practical constraints of our project, including time limitations and computational resources.

Moving on to the statistical analysis, we will follow the recommendations in [16] for comparing multiple classifiers across different datasets. We will use the Friedman test to assess whether there are significant differences between the classifiers. If significant differences are found, we will conduct the Nemenyi post-hoc test to identify which classifiers differ from each other. In addition, we will apply Holm's correction to the p-values in order to control the family-wise error rate.

In cases where neither test reveals significant differences among the top configurations, we will default to selecting the configuration with the highest average composite metric M_{config} across datasets (cross-validation folds). We acknowledge that this selection serves as a proxy and carries no statistical significance.

4.3.1 Best KNN Configuration for the Sick Dataset

After evaluating the various KNN configurations on the Sick dataset, we conducted the Friedman test to determine if there were statistically significant differences among the top-performing configurations. The test resulted in a statistic of 43.4182 and a p-value of 0.00001, which is well below the conventional significance threshold of 0.05, pointing to significant differences among configurations. Nevertheless, after running Nemenyi tests with Holm's correction, we found that no configuration differs with statistical significance, and thus we cannot favour any one over the others.

Given the absence of significant differences, we proceeded to rank the top configurations based on the highest composite metric M_{config} (see Table 1), and we selected the first configuration as "the best KNN for the Sick dataset".

Table 1: Top 8 Configurations for the Sick Dataset, according to M_{config}, order in descending order by the same metric.

k	**Feature Weighting**	**Voting Policy**	**Distance Metric**	M_{config}	**Accuracy**	**Time (s)**
7	Equal weighting	Sheppard	Hamming	0.850812	0.935625	1.786
5	Information gain	IDW	Hamming	0.849866	0.939028	1.889
7	Chi-squared	Sheppard	Hamming	0.849832	0.935625	1.794
3	Information gain	Sheppard	Hamming	0.849097	0.939073	1.896
5	Information gain	Sheppard	Hamming	0.848923	0.938976	1.895
7	Equal weighting	Majority	Hamming	0.846057	0.934193	1.783
7	Chi-squared	Majority	Hamming	0.845584	0.934193	1.786
7	Equal weighting	IDW	Hamming	0.844589	0.934193	1.794

Interestingly, the top configurations use Hamming distance, we believe due to its suitability for categorical data. Feature weighting methods vary, with equal weighting (the top configuration) indicating that no single feature dominates prediction, while information gain and chi-squared weighting appear in high-ranking configurations, suggesting some features have slightly more predictive power. Selection methods are also varied, with Sheppard's correction outperforming perhaps due to its ability to handle ties, a frequent issue with Hamming distance. Configurations with $k = 7$ dominate, likely due to the heavy class imbalance in the Sick dataset. Since we are optimising for accuracy, predicting the most common class (as occurs with a higher k) tends to be advantageous for the metric, albeit a mistake in real-world applications. Overall, the best configuration ($k = 7$, equal weighting, Sheppard's correction) balances high accuracy (93.56%) and efficiency.

It's important to note that this selection is not statistically significant due to the results of the Friedman test. Therefore, the choice serves as a practical recommendation rather than a statistically validated conclusion.

4.3.2 Best KNN Configuration for the Balance Scale Dataset

Once again we followed the statistical experiment design previously described. The Friedman test produced a statistic of 47.5316 and a p-value of 0.0000, which is far below the 0.05 significance threshold, indicating significant differences between configurations. Following this, we are able to apply the Nemenyi post-hoc test, with Holm's correction; the Critical Difference (CD) diagram, shown in Figure 1, displays the pairwise comparisons between configurations. Configurations connected by horizontal lines do not exhibit statistically significant differences at the adjusted significance level.

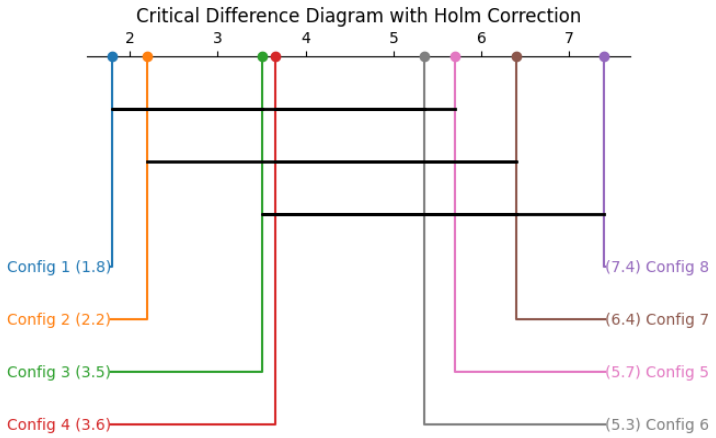


Figure 1: Critical Difference diagram of the Nemenyi post-hoc test results. Config n refers to the n-th configuration in descending order by M_{config} (see Table 2). Left-most is better.

As displayed in the diagram, Configurations 1 and 2 are significantly better than Configurations 7 and 8, with no overlap in their CD intervals; however, we cannot state statistical significance between Configurations 1 and 2 compared to Configurations 4, 5, and 6. Hence, we proceed by utilising the composite metric M_{config}. Based on this, we select Configuration 1 as "the best KNN for the Balance Scale dataset". This configuration uses $k = 7$, equal feature weighting, majority selection, and Hamming distance. It achieves the highest M_{config} value of 0.854814, an accuracy of 79.67%, and a low execution time of 0.0826 seconds (see Table 2 for the full ranking of configurations).

Table 2: Top 8 Configurations for the Balance Scale Dataset, ordered by descending M_{config}.

k	**Feature Weighting**	**Voting Policy**	**Distance Metric**	M_{config}	**Accuracy**	**Time (s)**
7	Equal weighting	Majority	Hamming	0.854814	0.796743	0.0826
7	Chi-squared	Majority	Hamming	0.852890	0.796743	0.0835
7	Equal weighting	IDW	Hamming	0.851293	0.796743	0.0842
7	Equal weighting	Sheppard	Hamming	0.850895	0.796743	0.0843
7	Chi-squared	IDW	Hamming	0.848010	0.796743	0.0856
7	Chi-squared	Sheppard	Hamming	0.847931	0.796743	0.0857
7	Information gain	Majority	Hamming	0.782388	0.801670	0.1180
5	Chi-squared	Majority	Hamming	0.759577	0.735738	0.0835

Reviewing the top 8 configurations, we observe commonalities: as in the previous dataset, all

configurations use Hamming distance, and majority selection is widely employed. Furthermore, $k = 7$ dominates the list, to which we attribute the same argument as with the Sick dataset: a combination of optimising for accuracy and heavy class imbalance. Feature weighting methods vary, with equal weighting and chi-squared appearing frequently; this seems to indicate that no single feature significantly outweighs others in predicting the outcome.

An important note, as was explained earlier (see Section 4.1.1), is that the Hamming distance and the Minkowski distance with $r = 1$ yield equivalent accuracy results for datasets containing only categorical features, such as the Balance Scale dataset. However, Hamming distance computation is faster, explaining its consistent selection over Minkowski in these configurations. This was initially discussed in the Hamming distance section, but bears reiteration here as it directly influences the balance between accuracy and efficiency.

Once again, we must note that, as statistically, we could not ascertain any difference amongst the top 6 configurations, any of these would have been a correct pick; nevertheless, we utilise M_{text} as a proxy to select a single "best" configuration.

5 Support Vector Machine (SVM)

As the eager learning technique selected for this study, Support Vector Machine (SVM) represents a fundamentally different approach to classification compared to lazy learning methods. Rather than deferring generalisation until prediction time, SVM constructs a decision boundary, or hyperplane, at the time of training. This hyperplane aims to maximise the margin between data points of different classes, thereby optimising the model's ability to distinguish between categories when new data is encountered. In this work, we leverage the C-Support Vector Classification (SVC) implementation from scikit-learn [17], which offers efficient handling of classification tasks and various configurable parameters that can be adjusted to suit the characteristics of different datasets.

5.1 SVM Components

SVM's flexibility comes from several core parameters, which we tune to optimise accuracy and computational efficiency across different datasets. Our study focuses on four principal parameters: *kernel*, *C*, *gamma*, and *class weight*.

5.1.1 Kernel

The *kernel* function determines how the input space is transformed, allowing SVM to fit either linear or non-linear decision boundaries. In this work, we test two types of kernels: the linear kernel and the RBF (Radial Basis Function) kernel. The linear kernel is suitable for linearly separable data, yielding a straightforward decision boundary, while the RBF kernel maps data into an infinite-dimensional space, allowing for complex, non-linear boundaries. We chose these kernels as they contrast with each other, allowing us to gain insight into the structure of the data

5.1.2 Regularisation (C)

The C parameter controls the balance between maximising the margin and minimising classification errors. Higher values of C prioritise correct classification of training data points, enhancing accuracy but also risking overfitting; in contrast, lower C values increase the margin width, permitting some misclassified points and promoting generalisation. We test a range of C values (from 0.001 to 100), examining how regularisation impacts the model's performance on each dataset.

5.1.3 Gamma

The *gamma* parameter, applicable **only** to the RBF kernel, defines the influence range of individual data points. Higher gamma values create a more fine-grained decision boundary by narrowing each support vector's reach (potentially leading to overfitting), whereas lower values produce a smoother, simpler boundary by expanding the influence of each support vector. In this study, we explore gamma settings of *scale*, *auto*, and specific values such as 0.1 to understand their impact on boundary complexity and model generalisation.

5.1.4 Class Weight

The *class weight* parameter addresses class imbalance by adjusting the penalty for misclassifications of each class: setting class weight to " balanced" automatically scales weights inversely proportional to class frequencies, ensuring that under-represented classes exert greater influence on the decision boundary. This might prove specially useful for the "Sick" dataset.

5.1.5 Shrinking

Finally, we investigate the *shrinking* heuristic, which accelerates the optimisation process by iteratively discarding support vectors that do not contribute to the final decision boundary. This setting is purely focused on efficiency, a core metric to optimise.

The specific variations for each parameter tested in this study are as follows:

- **Kernel**: `['linear', 'rbf']`
- **C**: `[0.001, 0.01, 0.1, 1, 10, 100]`
- **Gamma** (only applicable to the RBF kernel): `['scale', 'auto', 0.001, 0.01, 0.1, 1]`
- **Class Weight**: `[None, 'balanced']`
- **Shrinking**: `[True, False]`

This results in $6 \times 2 \times 2 = 24$ configurations using the linear kernel, and $6 \times 6 \times 2 \times 2 = 144$ configurations using the RBF kernel. In total, we test $144 + 24 = 168$ configurations.

5.2 Analysis of the Best SVM Configuration

Following the approach used in the KNN configuration analysis, we aim to identify the optimal SVM configuration for each dataset, considering both classification accuracy and computational efficiency. As with KNN, we define a composite metric M_{config} for each SVM configuration, calculated as:

$$M_{\text{config}} = \alpha \times A_{\text{config}} + \beta \times (1 - T_{\text{config}}) \tag{2}$$

where A_{config} represents the classification accuracy, T_{config} is the normalised execution time, and α and β are weighting factors set to $\alpha = 0.7$ and $\beta = 0.3$, prioritising accuracy over efficiency. Accuracy values are normalised to the $[0, 1]$ range using min-max normalisation to ensure fair contributions to M_{config}, as explained in the KNN analysis.

Given the constraints of our project, we adopt the same relaxations for statistical analysis: treating each cross-validation fold as an independent dataset to increase the effective dataset count to $N = 10$, and selecting the top eight configurations based on M_{config} for further analysis. Although these adjustments violate certain statistical assumptions, they enable us to conduct meaningful analysis under our computational constraints.

We will once again follow the experimental approach proposed by [16], performing a Friedman test followed by a Nemenyi post-hoc, with Holm's correction (to control the family-wise error rate).

5.2.1 Best SVM Configuration for the Sick Dataset

After evaluating various SVM configurations on the Sick dataset, we conducted the Friedman test, which yielded a statistic of 60.9818 and a p-value of 0.0000, indicating significant differences among configurations. We then performed the Nemenyi post-hoc test with Holm's correction to identify which configurations differ significantly. The resulting Critical Difference (CD) diagram, shown in Figure 2, illustrates the pairwise comparisons among the top eight configurations, where horizontal lines connect configurations that do not exhibit statistically significant differences at the adjusted significance level.

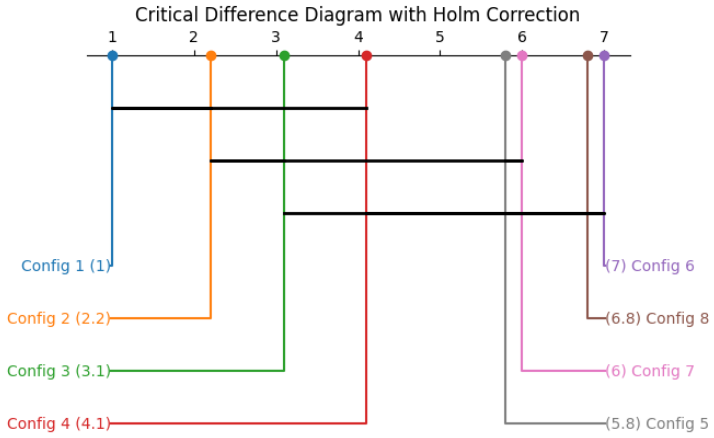


Figure 2: Critical Difference diagram of the Nemenyi post-hoc test results for SVM configurations on the Sick dataset. Configurations connected by horizontal lines do not exhibit statistically significant differences at the adjusted significance level.

In the CD diagram, Configuration 1 is statistically superior to Configurations 5, 6, 7, and 8, as it is not connected to these configurations by any horizontal line. However, Configurations 2, 3, and 4 are not statistically different from Configuration 1, as they are connected by horizontal lines in the diagram. This suggests that any of the top four configurations could be considered competitive candidates, with Configuration 1 achieving the highest ranking based on its composite metric M_{config}.

Among the top configurations, Configuration 1 achieves the highest composite metric M_{config} of 0.9647, with an accuracy of 93.65% and an execution time of 0.0354 seconds. This configuration uses the RBF kernel with $C = 100.0$, balanced class weights, shrinking enabled, and $\gamma = 1$. Given its top M_{config} score and statistically significant performance over Configurations 5 through 8, we selected Configuration 1 as "the best SVM configuration for the Sick dataset".

Table 3: Top 8 SVM Configurations for the Sick Dataset, ordered by M_{config}.

Kernel	C	**Class Weight**	**Shrinking**	γ	$M_{\textbf{config}}$	**Accuracy**	**Time (s)**
rbf	100.0	balanced	True	1	0.964700	0.936469	0.0354
rbf	100.0	balanced	False	1	0.962976	0.936469	0.0445
rbf	100.0	balanced	True	scale	0.957101	0.927084	0.0372
rbf	100.0	balanced	False	scale	0.955694	0.926708	0.0431
rbf	100.0	balanced	True	0.1	0.942668	0.910670	0.0464
rbf	100.0	balanced	False	0.1	0.941420	0.910670	0.0530
rbf	10.0	balanced	True	1	0.940862	0.909149	0.0498
rbf	10.0	balanced	False	1	0.940794	0.909149	0.0501

A closer inspection of the top configurations reveals several common characteristics that may explain their high performance. All top configurations use the RBF kernel, which is well-suited for handling non-linear relationships in data; this suggest the decision boundary for the Sick dataset is complex. Moreover, each configuration applies balanced class weights, extremely useful given the how unbalanced this dataset is. A relatively high C value (mostly $C = 100.0$) indicates that a strong regularisation weight allows the model to capture detailed patterns without underfitting. The use of shrinking is also common among the top configurations, reducing computational effort.

5.2.2 Best SVM Configuration for the Balance Scale Dataset

To determine the optimal SVM configuration for the Balance Scale dataset, we first applied the Friedman test, which yielded a test statistic of 22.2273 and a p-value of 0.0082. This result indicates that there are significant differences among the configurations. To identify specific differences, we performed the Nemenyi post-hoc test with Holm's correction. The resulting Critical Difference (CD) diagram, shown in Figure 3, displays pairwise comparisons among the top eight configurations, where configurations connected by horizontal lines do not exhibit statistically significant differences at the adjusted significance level.

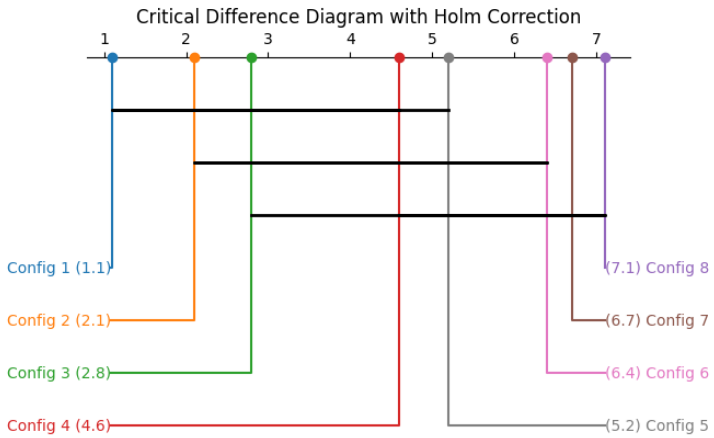


Figure 3: Critical Difference diagram of the Nemenyi post-hoc test results for SVM configurations on the Balance Scale dataset. Configurations connected by horizontal lines do not exhibit statistically significant differences at the adjusted significance level.

According to the CD diagram, Configuration 1 stands out as statistically superior to Configurations 6, 7, and 8, as it is not connected to these configurations by any horizontal line. Configurations 2, 3, 4, and 5, however, are statistically comparable to Configuration 1, as they

are connected by horizontal lines, suggesting that these configurations also perform well on the Balance Scale dataset.

Configuration 1 achieves a composite metric M_{config} of 0.9567, with an accuracy of 95.85% and an execution time of 0.0049 seconds. This configuration uses the linear kernel with $C = 10.0$, balanced class weights, shrinking enabled, and $\gamma = 0$. Given its high M_{config} score and statistically significant differences from Configurations 6 through 8, we selected Configuration 1 as "the best SVM configuration for the Balance Scale dataset".

Table 4: Top 8 SVM Configurations for the Balance Scale Dataset, ordered by M_{config}. "Shrinking" is "N/A" where this parameter is not applicable (outside of the RBF kernel)

Kernel	C	**Class Weight**	**Shrinking**	γ	$M_{\textbf{config}}$	**Accuracy**	**Time (s)**
linear	10.0	balanced	True	N/A	0.956702	0.958488	0.0049
linear	10.0	balanced	False	N/A	0.952529	0.958488	0.0051
linear	100.0	balanced	True	N/A	0.929626	0.979131	0.0073
rbf	100.0	balanced	True	0.1	0.905774	0.955258	0.0077
rbf	100.0	balanced	True	auto	0.903264	0.966450	0.0083
linear	1.0	balanced	True	N/A	0.884529	0.859118	0.0047
rbf	10.0	balanced	True	0.1	0.882676	0.910440	0.0071
rbf	100.0	balanced	False	0.1	0.882213	0.955258	0.0090

Examining the top configurations, we notice commonalities: The linear kernel is prominent in the top-ranked configurations (Configurations 1, 2, 3, and 6), which suggests that the data in the Balance Scale dataset may be more linearly separable; balanced class weights are used across all top configurations, which likely addresses any class imbalance; the parameter C varies among the top configurations, with values ranging from 1.0 to 100.0, indicating some flexibility in regularisation strength depending on the setup. Shrinking is enabled in most of the top configurations, although interestingly the top 2 are the same with the exception of shrinking, indicating that in some cases this parameter does not have much impact.

6 KNN vs SVM

Having analysed the best KNN and SVM configurations for each dataset, we now address the question: which of these two, lazy (KNN) or eager (SVM) learning algorithms, performs better? To answer this, we statistically compare the performance of the best configurations for each algorithm on each dataset, taking into account both accuracy and computational efficiency.

The best configurations identified for each algorithm and dataset are as follows:

1. **Sick Dataset**: For KNN, we use $k = 7$, equal feature weighting, Sheppard's voting, and Hamming distance; for SVM, we employ the RBF kernel with $C = 100.0$, $\gamma = 1$, class weight set to `balanced`, and shrinking enabled.

2. **Balance Scale Dataset**: For KNN, we use $k = 7$, equal feature weighting, majority voting, and Hamming distance; for SVM, we apply the linear kernel with $C = 10.0$, class weight set to `balanced`, and shrinking enabled.

To compare KNN and SVM, we follow the recommendations of [16] for two-classifier comparisons over multiple datasets, applying the Wilcoxon signed-rank test. This non-parametric test is suited for comparing paired, non-normally distributed samples, assessing whether differences between classifiers are statistically significant.

In this analysis, each cross-validation fold is treated as an independent dataset, resulting in $N = 10$ observations per original dataset. While this approach introduces dependencies across folds, it is necessary due to the limited number of datasets.

A few caveats and relaxations should be noted. Firstly, KNN's computational efficiency is challenging to compare directly to SVM's because KNN incurs its computational cost at each prediction, whereas SVM incurs a large cost during training, after which predictions are relatively fast. This cost difference is accounted for within each algorithm and dataset by normalising the accuracy and time for each configuration separately; however, this metric is less suitable for direct comparison between KNN and SVM. Despite this, we once again compute the composite metric M_{config} as a means to evaluate each algorithm holistically within the dataset constraints.

Additionally, it is important to note that KNN was implemented from scratch, while SVM was utilised through an existing library. This difference may affect the results, as the library implementation of SVM is likely optimised for efficiency and performance, potentially giving it an edge over the KNN implementation.

6.1 Best Algorithm for the Sick Dataset

To determine the best algorithm for the Sick dataset, we conducted the Wilcoxon signed-rank test on the composite metric M_{config} across folds. The results are:

- Number of non-zero differences (N): 10
- Sum of ranks for positive differences (R^+): 55.0
- Sum of ranks for negative differences (R^-): 0.0
- Test statistic (T): 0.0
- z-value: -2.8031
- p-value: 0.0051

Since the p-value is less than $\alpha = 0.05$, we reject the null hypothesis, indicating a statistically significant difference between KNN and SVM. The sum of ranks for positive differences ($R^+ = 55.0$) being greater than the sum of ranks for negative differences ($R^- = 0.0$) suggests that SVM outperforms KNN on the Sick dataset.

6.2 Best Algorithm for the Balance Scale Dataset

Similarly, the Wilcoxon signed-rank test was applied to the Balance Scale dataset, yielding the following results:

- Number of non-zero differences (N): 10
- Sum of ranks for positive differences (R^+): 53.0
- Sum of ranks for negative differences (R^-): 2.0
- Test statistic (T): 2.0
- z-value: -2.5992
- p-value: 0.0093

Again, with a p-value below $\alpha = 0.05$, we reject the null hypothesis, concluding that SVM significantly outperforms KNN on the Balance Scale dataset as well. The positive rank sum ($R^+ = 53.0$) further supports SVM's superior performance on this dataset.

6.3 Analysis

The Wilcoxon signed-rank test confirms that SVM significantly outperforms KNN on both datasets in terms of the composite metric M_{config}, underscoring SVM's robustness in accuracy and computational efficiency.

SVM's performance can be attributed to its adaptability via kernel functions. The RBF kernel handles the Sick dataset's non-linear patterns effectively, while the linear kernel achieves high accuracy on the likely linearly separable Balance Scale dataset. KNN, by contrast, often struggles with high dimensionality, or when there are not many data points (as is the case with Balance Scale).

Additionally, SVM's `balanced` class weight setting allows it to handle class imbalance in datasets such as Sick more effectively. In terms of efficiency, SVM's reliance on support vectors reduces computational demands during prediction, contrasting with KNN's need to calculate distances to all training instances.

In summary, based on the Wilcoxon signed-rank test [16], SVM consistently surpasses KNN across both datasets, combining accuracy and efficiency.

7 Instance Reduction Techniques

In this subsection, we evaluate the performance of the KNN and SVM models using three instance reduction techniques: GCNN, ENNTh, and DROP3. The evaluation focuses on three key metrics: accuracy, efficiency, and storage (percentage of training instances used). Our goal is to determine how reducing the training set impacts model performance compared to using the full training set.

For each instance reduction technique, we configured the parameters based on the optimal settings identified in the literature, as detailed in Sections 7.1, 7.2, and 7.3. We conducted the experiments using the best KNN and SVM configurations for both the Sick and Balance datasets, as presented in Table 1 and Table 2 for KNN; and Table 3 and Table 4 for SVM.

Following the approach used in both KNN and SVM configuration analysis, we will conduct our analysis defining a composite metric M_{config} calculated as:

$$M_{\text{config}} = \alpha \times A_{\text{config}} + \beta \times T_{\text{config}} + \omega \times S_{\text{config}} \tag{3}$$

where A_{config} represents the classification accuracy, T_{config} is the normalised execution time, and S_{config} is the normalised storage used; α, β, and ω are weighting factors set to $\alpha = 0.55$, $\beta = 0.05$, and $\omega = 0.4$, prioritising a higher accuracy and lower storage over efficiency. All three metrics are normalised to the [0, 1] range as in the previous analysis, to ensure a fair contribution to M_{config}, which higher values indicate a better overall performance balance.

7.1 GCNN - Condensed

Condensed instance reduction techniques aim to reduce the size of a dataset by selecting a subset of the samples (the prototypes) closer to the decision boundaries. As it removes internal samples, the reduction rate is normally high. In specific, we implemented the GCNN algorithm, a filter approach with incremental direction of search.

The Generalised Condensed Nearest Neighbour (GCNN) selects one prototype for each unique class and then, through an iterative absorption process, it determines whether the remaining samples can be represented by the existing prototypes. If it is the case, the sample is marked as absorbed; otherwise, it remains unabsorbed. For all the unabsorbed samples of a class, it chooses one to become a new prototype. The algorithm stops when all the data is either absorbed or marked as prototype.

The GCNN algorithm is defined by the hyper-parameter p, which controls the tolerance level for the absorption criterion. As the value increases, it becomes more strict and therefore, the number of prototypes is higher. We used the proposed value of 0.7 [18].

7.2 ENNTh - Edited

Edited reduction techniques are the opposite to the condensed strategies: its main goal is to discard the border points from a dataset. The reasoning behind these methods is that samples placed in local regions corresponding to a different class are more likely to be noisy data.

For our project, we have selected the ENNTh method. For each instance, the algorithm evaluates its k-nearest neighbours and computes the probability of belonging to each one of the classes. Samples are removed if they do not meet the following condition: be classified to their true label and have a probability higher than a threshold.

The algorithm has two hyper-parameters: the number of neighbours and the threshold of the condition criteria. We have set k = 7, which was extracted for the best KNN configuration of both datasets. For the threshold we used the proposed value of 0.8 [19].

7.3 DROP3 - Hybrid

Hybrid reduction techniques seek to eliminate redundant and noisy data, while retaining the most relevant instances. It is a combination of both condensed and edited methods. The DROP family [20] is widely recognised as a set of hybrid filter approaches. DROP3 offers an optimal balance between generalisation accuracy and storage reduction, which is why it was selected to be implemented in this project.

The algorithm initially eliminates instances that do not contribute to the decision boundary through a pre-filtering step using the ENNTh algorithm. Subsequently, it iteratively assesses each instance within its k-nearest neighbourhood, retaining only those that enhance the classification accuracy of their neighbours. Instances are removed if their absence does not adversely impact performance.

DROP3 relies on the hyper-parameters of the ENNTh algorithm, where we used the previously specified values. It also requires setting the number of neighbours to consider, which has ben configured to k=7.

7.4 KNN Performance with Instance Reduction

The results for KNN are summarised in Table 5 and Table 6. For each dataset, we calculated the M_{config} metric, and the average accuracy, storage and efficiency metrics across all folds.

Table 5: Results of KNN with Instance Reduction on the Sick Dataset, sorted by M_{config}

IR Method	$M_{\textbf{config}}$	**Accuracy**	**Storage**	**Time (s)**
DROP3	0.827818	0.7738	2.55%	0.0770
GCNN	0.728216	0.6630	1.66%	0.04645
ENNTh	0.573113	0.9240	89.43%	1.8834
No IR	0.543754	0.9390	100%	1.7860

For the Sick dataset, the highest accuracy score (93.90%) is achieved without using any instance reduction. In fact, as smaller the dataset is, the lower accuracy we get. The ENNTh method retains a large portion of the data (89.43% of storage), which aligns with its relatively high accuracy (92.40%). In contrast, DROP3 and GCNN significantly decreases storage, retaining only 2.55% and 1.66% of the dataset, respectively. This extreme reduction comes at the expense of accuracy, which drops to 77.38% and 66.30%. This results suggest that the complete dataset is needed to capture the necessary information to ensure accurate classification.

In terms of M_{config}, DROP3 seems to provide the best compromise, achieving a relatively high accuracy and drastically reduced storage use. ENNTh also offers high accuracy but is penalized due to high storage and time requirements, resulting in a lower M_{config}.

Table 6: Results of KNN with Instance Reduction on the Balance Dataset, sorted by M_{config}

IR Method	$M_{\textbf{config}}$	**Accuracy**	**Storage**	**Time (s)**
ENNTh	0.795069	0.8386	35.00%	0.0348
DROP3	0.536850	0.5202	4.19%	0.0044
GCNN	0.464824	0.7695	87.75%	0.0856
No IR	0.448138	0.7967	100%	0.0826

On the other hand, the Balance Dataset obtains the maximum accuracy with ENNTh (83.86%), which reduces data storage to only the 35% of the original size. The next best accuracy is obtained with the unmodified dataset (79.67% accuracy). This suggest that the dataset contains overlapping regions that interfer with the model's learning. By removing border points, ENNTh sharpen boundaries between classes, thereby enhancing the model's precision.

GCNN and DROP3 show a decline in accuracy. The former maintains 87.75% of the data but achieves lower accuracy of 76.98%. The latter reduces storage drastically to 4.19%, resulting in a significant drop in accuracy to 52.02%. The excessive reduction leads to a worst model performance. Finding the right balance between data retention and noise reduction optimises accuracy on this dataset.

For the M_{config} metric, we can see that ENNTh outperforms other models, effectively balancing high accuracy with moderate storage requirements. DROP3, on the other hand, excels in minimizing storage usage, but its significantly lower accuracy. GCNN and the Baseline model both achieve higher accuracy than DROP3 but are penalized due to their high storage requirements.

In both cases, the computation time seems to be directly proportional to the storage.

It can be concluded that for the Sick dataset, retaining the full dataset or using ENNTh yields the best results, as all the data is relevant to achieve precise accuracy. For the Balance dataset, ENNTh is the most effective technique: it improves performance by removing samples in overlapping regions and lowering computational time.

7.4.1 Statistical Test

We computed the Friedman test of the best KNN configuration with each of the 4 datasets (the original one and the 3 reduced), obtaining a test statistic of 16.6800 and a p-value of 0.0008 for the Sick dataset. Since the p-value is lower than the significant threshold (0.05), there is a statistically significant difference in the performance of the models. The Critical Difference (CD) Diagram of Figure 6 compares the 4 different models discussed previously. Drop3 emerges as the top-performing method, with the lowest rank (1.4). It shows a statistically significant improvement over the Baseline model. However, Drop3, GCNN (2.2), and ENNTH (2.7) are statistically similar, as evidenced by the connecting black lines between these models. This suggests that Drop3, GCNN, and ENNTH deliver comparable results.

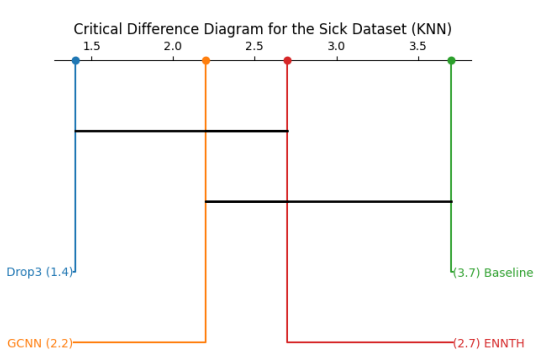


Figure 4: Critical Difference diagram of the Nemenyi post-hoc test results for KNN configurations on the Sick dataset. Configurations connected by horizontal lines do not exhibit statistically significant differences at the adjusted significance level.

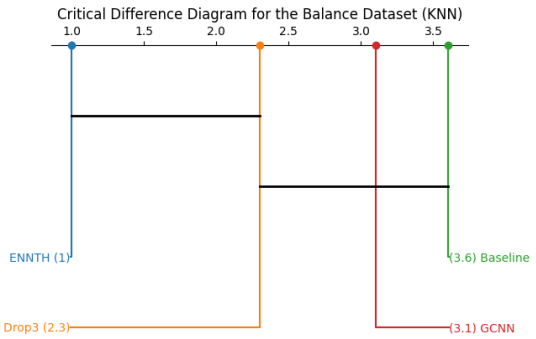


Figure 5: Critical Difference diagram of the Nemenyi post-hoc test results for KNN configurations on the Balance dataset. Configurations connected by horizontal lines do not exhibit statistically significant differences at the adjusted significance level.

Repeating the analysis for the Balance dataset, it yielded a statistic of 23.1600 and a p-value of 0.0. The CD diagram in Figure 5 reveals that ENNTh is the leading model with the lowest rank, followed closely by Drop3. They are statistically similar. GCNN and the Baseline achieved a lower rank, indicating weaker performance.

In conclusion, KNN with the DROP3-reduced dataset is preferred for the Sick dataset, while ENNTH is the optimal choice for the Balance dataset. Both approaches achieve an excellent trade-off between accuracy and storage efficiency, making them ideal choices for their respective datasets.

7.5 SVM Performance with Instance Reduction

The results for SVM are summarised in Table 7 and Table 8. For each dataset, we calculated the M_{config} metric, and the average accuracy, storage, and efficiency metrics across all folds.

Table 7: Results of SVM with Instance Reduction on the Sick Dataset, sorted by M_{config}

IR Method	$M_{\textbf{config}}$	**Accuracy**	**Storage**	**Time (s)**
DROP3	0.89712	0.8720	2.55%	0.0033
GCNN	0.69465	0.6776	1.66%	0.0034
ENNTh	0.62147	0.9452	89.43%	0.0096
No IR	0.52779	0.9365	100%	0.0354

Table 8: Results of SVM with Instance Reduction on the Balance Dataset, sorted by M_{config}

IR Method	$M_{\textbf{config}}$	**Accuracy**	**Storage**	**Time (s)**
ENNTh	0.73431	0.8834	35.00%	0.0027
DROP3	0.65458	0.6830	4.19%	0.0027
GCNN	0.60615	0.9617	87.75%	0.0046
No IR	0.53417	0.9585	100%	0.0049

On the Sick dataset, DROP3 achieved the highest M_{config} of 0.89712, indicating an optimal trade-off between reduced storage and maintained accuracy. This was followed by GCNN with

0.69465 and ENNTh with 0.62147. The baseline method without instance reduction had the lowest M_{config} of 0.52779, highlighting the advantages of applying instance reduction techniques to reduce storage. Similarly, in the Balance dataset, ENNTh led with a M_{config} of 0.73431, demonstrating good storage reduction while preserving high accuracy and efficient computation time. DROP3 and GCNN recorded M_{config} values of 0.65458 and 0.60615, respectively. The baseline SVM again had the lowest M_{config} of 0.53417, further emphasising the effectiveness of instance reduction in enhancing overall model performance.

Regarding performance metrics, the SVM without instance reduction achieved accuracies of 93.65% on the Sick dataset and 95.85% on the Balance dataset. Applying ENNTh slightly improved accuracy to 94.52% (Sick) but decreased it to 88.34% (Balance), while DROP3 and GCNN reduced accuracy to 87.20% and 67.76% (Sick), and 68.30% (Balance), respectively. Computational time was consistently reduced across all instance reduction techniques for both datasets, with the Sick dataset time decreasing from 0.0354s (baseline) to as low as 0.0033s (DROP3), and the Balance dataset time dropping below the baseline of 0.0049s. Storage was also minimised, decreasing to 89.43% with ENNTh, 1.66% with GCNN, and 2.55% with DROP3 for the Sick dataset, and to 35.00% with ENNTh for the Balance dataset. These results indicate that instance reduction techniques effectively decrease computational resources and storage needs, though their impact on accuracy varies depending on the method and dataset. Overall, DROP3 and GCNN offer substantial reductions in storage and time, whereas ENNTh maintains higher accuracy with moderate resource reductions.

To make a more robust comparison between these techniques, we performed a statistical test to draw stronger conclusions.

7.5.1 Statistical Test

We performed the Friedman test across the three instance reduction techniques and the SVM without instance reduction, obtaining a test statistic of 20.4000 and a p-value of 0.0001 for the Sick dataset. The Critical Difference (CD) diagram in Figure 6 illustrates the pairwise comparisons among the four algorithms, where horizontal lines connect techniques that do not exhibit statistically significant differences at the adjusted significance level.

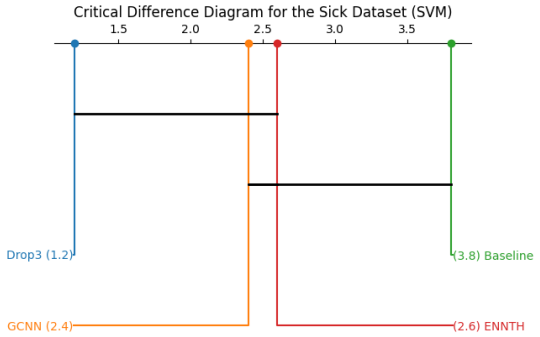


Figure 6: Critical Difference diagram of the Nemenyi post-hoc test results for SVM configurations on the Sick dataset. Configurations connected by horizontal lines do not exhibit statistically significant differences at the adjusted significance level.

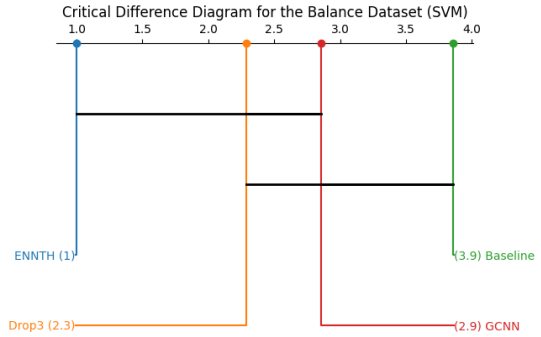


Figure 7: Critical Difference diagram of the Nemenyi post-hoc test results for SVM configurations on the Balance dataset. Configurations connected by horizontal lines do not exhibit statistically significant differences at the adjusted significance level.

According to the CD diagram, DROP3, GCNN, and ENNTh are statistically comparable, as they are connected by a horizontal line. Additionally, DROP3 is the only technique that is

statistically superior to the SVM without instance reduction. As shown in Table 7, DROP3 achieved a high accuracy of 0.8720 with an amazing storage of only 2.55% of training instances.

For the Balance dataset, the Friedman test obtained a test statistic of 17.9143 and a p-value of 0.0005. ENNTh, DROP3, and GCNN are statistically comparable, and ENNTh is the only technique that is statistically superior to the SVM without instance reduction.

8 Discussion

The purpose of this study was to compare the performance of lazy (KNN) and eager (SVM) learning algorithms on two distinct datasets, Sick and Balance Scale, and to investigate the impact of instance reduction techniques on their performance. We aimed to study such matters in terms of accuracy, efficiency, and where applicable, storage. We showed that SVM consistently outperformed KNN on both datasets, as evidenced by the Wilcoxon signed-rank test, hinting at its more robust and adaptable performance when compared to lazy learning. The superior performance of SVM can be attributed to its ability to handle non-linear relationships through kernel functions and its effective management of class imbalance using the balanced class weight setting.

For KNN, the optimal configurations for both datasets used $k = 7$ and Hamming distance, but differed in the voting policy (Sheppard's for Sick and majority for Balance Scale). The consistent use of high k values across both datasets may be attributed to the class imbalance present in the data, as this way we favour the majority class, while the preference for Hamming distance indicates its efficiency when handling features. The difference in voting policies suggests that the Sick dataset may benefit from a more nuanced approach to handling ties, while the Balance Scale dataset may have a clearer distinction between classes.

In the case of SVM, the best configuration for the Sick dataset employed the RBF kernel, while the Balance Scale dataset favoured the linear kernel. The former may indicate the presence of non-linear relationships in the data—which the RBF is well-suited to handle—, while the latter may once again support the hypothesis of the classes being more easily—perhaps linearly— separable. Both datasets benefited from the use of balanced class weights, showcasing the importance of addressing class imbalance.

The application of instance reduction techniques yielded mixed results depending on the dataset and the specific technique used. For KNN, the DROP3 technique showed to be the most effective for the Sick dataset, while ENNTh showed most effective for the Balance Scale dataset. These findings indicate that the effectiveness of each technique may heavily depend on the characteristics of each dataset, such as the presence of noisy or redundant instances. Similarly, for SVM, DROP3 proved the best for the Sick dataset, while ENNTh was the most effective for the Balance Scale dataset.

This study presents certain limitations. Firstly, we treated folds as independent datasets following the practical work instructions; however, this approach may not be ideal, as testing should account for dataset independence, as suggested by [16]. Secondly, our evaluations focused on accuracy; however, incorporating additional metrics such as F1-Score and AUC would be preferable, especially given the imbalanced nature of the data, particularly the Sick dataset. Accuracy was used because it was specified in the practical work instructions. Thirdly, when comparing KNN and SVM head-to-head, we did not account for SVM's execution time being heavily reduced after training: this might have biased our results, however such direct comparison is tricky to perform. Fourthly, in the instance reduction experiments, we configured the techniques using parameters previously tested in other studies. Conducting our own parameter search, such as a Grid Search, could potentially enhance the performance of instance reduction techniques, as indicated in [18] [19].

Future research should address these limitations to develop more robust and reliable models and results than those presented in the current study .

9 Conclusions

This study aimed to evaluate eager and lazy learning techniques, specifically Support Vector Machine (SVM) and K-Nearest Neighbours (KNN), and to investigate the impact of instance reduction techniques on their performance. Through rigorous experimentation and statistical analysis, we determined the optimal parameter configurations for both algorithms across the Sick and Balance Scale datasets. Our findings revealed that SVM consistently outperformed KNN in both datasets, demonstrating its robustness and adaptability in handling complex data relationships and class imbalance. The application of instance reduction techniques further improved the overall balance of performance between accuracy, storage, and time. Statistical tests confirmed that employing instance reduction achieved superior performance compared to algorithms using the full training set. The implications of this study are significant for researchers and practitioners working with classification problems, as it highlights the potential of eager learning algorithms and instance reduction techniques for optimising machine learning workflows. We recommend carefully evaluating the dataset's characteristics to choose the most suitable algorithm and instance reduction technique, enabling researchers to develop models that are more accurate, efficient, and resource-friendly across various applications.

References

[1] OpenAI. This image was created with the assistance of dall·e 2. `https://openai.com/dall-e-2`, 2024. Accessed: 12-10-2024.

[2] Kolby Nottingham Markelle Kelly, Rachel Longjohn. The uci machine learning repository. Accessed: 2024-10-12.

[3] R. Siegler. Balance Scale. UCI Machine Learning Repository, 1976. DOI: https://doi.org/10.24432/C5488X.

[4] Wikipedia contributors. One-hot — Wikipedia, the free encyclopedia, 2024. [Online; Accessed: 2024-10-12].

[5] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, page 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[6] Wikipedia contributors. F-score — Wikipedia, the free encyclopedia, 2024. [Online; Accessed: 2024-10-12].

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

[8] Mahinda Mailagaha Kumbure and Pasi Luukka. Local means-based fuzzy k-nearest neighbor classifier with minkowski distance and relevance-complementarity feature weighting. *Granular Computing*, 9(4):73, August 2024.

[9] Wikipedia contributors. Minkowski distance — Wikipedia, the free encyclopedia, 2024. [Online; Accessed: 2024-10-12].

[10] Wikipedia contributors. Mahalanobis distance — Wikipedia, the free encyclopedia, 2024. [Online; Accessed: 2024-10-12].

[11] Wikipedia contributors. Cosine similarity — Wikipedia, the free encyclopedia, 2024. [Online; Accessed: 2024-10-12].

[12] Wikipedia contributors. Hamming distance — Wikipedia, the free encyclopedia, 2024. [Online; Accessed: 2024-10-12].

[13] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[14] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 3rd edition, 2011.

[15] Jain AK and Douglas Zongker. Feature selection: Evaluation, application, and small sample performance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19:153 – 158, 03 1997.

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

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python, 2011.

[18] Fu Chang, Chin-Chin Lin, and Chi-Jen Lu. Adaptive prototype learning algorithms: Theoretical and experimental studies. *Journal of Machine Learning Research*, 7:2125–2148, 10 2006.

[19] Fernando Vázquez, Josep Sánchez, and Filiberto Pla. A stochastic approach to wilson's editing algorithm. pages 35–42, 01 2005.

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