Project Report

Introduction to Machine Learning

Master in Artificial Intelligence - UPC/UB/URV

Work 2. Classification with Lazy Learning and SVMs

Authors:
Andreu Garcies
Maria Guasch
Helena Sánchez
Natalia Muñoz

Submitted to:
Maria Salamó

November 2025

Introduction to Machine Learning: Work 2. Classification with Lazy Learning and SVMs.

Andreu Garcies Ramon[1], Maria Guasch Torres[1], Natalia Muñoz Moruno[1] and Helena Sánchez Ulloa[1]

[1] *Master in Artificial Intelligence, Universitat Politècnica de Catalunya UPC, Barcelona, Catalonia.*

Abstract—The objective of this practice was to implement and compare instance-based classifiers (k-IBL) and Support Vector Machines (SVMs) across different datasets. For k-IBL we implemented multiple distance metrics (Euclidean, Cosine, IVDM), voting (Borda Count and Modified Plurality) and retention policies (never retain, always retain, different class and degree of disagreement), feature-weighting methods (Relief, SFS) and instance-reduction algorithms (AllKNN, MCNN, ICF). For SVMs we explored linear, polynomial and RBF kernels and a range of hyperparameters. Experiments report accuracy and total runtime, and results are analyzed with the Friedman test and Nemenyi post-hoc comparisons to draw statistically significant conclusions. The practical outcome shows SVM (RBF, $C = 10$, γ=scale) as the preferred choice for complex or large datasets, while a tuned k-IBL (Euclidean + Borda count, k=5, always retain) remains competitive on structured datasets with relatively small number of instances.

Keywords—Machine learning, k-Instance Based Learning, Support Vector Machines, Statistical analysis, Nemenyi test, Friedman test

I. Introduction

Classification tasks represent one of the most fundamental problems in machine learning, with applications ranging from handwritten digit recognition to credit risk assessment. The effectiveness of classification algorithms depends not only on the learning algorithm itself but also on numerous design decisions, including similarity measures, feature relevance, and data preprocessing strategies. Among classification algorithm approaches, instance-based learning methods and Support Vector Machines (SVMs) are two effective methods. For this reason, this work focuses on an analysis of k-Instance-Based Learning (k-IBL) algorithms and their comparison with SVMs. Instance-based learning relies on local similarity comparisons and delays generalization until query time, while SVMs seek globally optimal decision boundaries through the maximum margin principle [1, 2].

k-IBL extends the idea of the k-Nearest Neighbors (k-NN) algorithm and classifies new instances by comparing them to the k most similar examples in the training set. Its main advantage lies in its simplicity and interpretability, as it makes decisions based directly on observed data rather than relying on complex model assumptions. Additionally, k-IBL can adapt naturally to changes in the dataset and handle complex, nonlinear decision boundaries. However, it also has several limitations. Performance can deteriorate in high-dimensional spaces due to the "curse of dimensionality", and the method can be computationally expensive at query time. Some versions of k-IBL can also be sensitive to irrelevant or noisy features, which can distort similarity measures and reduce classification accuracy [1].

To address these challenges and optimize k-IBL performance, we investigate the impact of multiple algorithmic components and preprocessing techniques. Specifically, we examine three distance metrics (Euclidean, cosine, and Interpolated Value Difference Metric (IVDM)), three values of k (3, 5, and 7), two voting policies (modified plurality and borda count), and four retention strategies (never retain, always retain, different class, and degree of disagreement). Beyond the base algorithm, we explore two critical preprocessing approaches: feature weighting, which identifies and emphasizes relevant attributes while giving less importance to irrelevant ones, and instance reduction, which aims to compress the training set while preserving classification accuracy.

In parallel, we have also analyzed SVM classifiers with different kernel functions. By comparing optimized k-IBL configurations with SVM variants, trade-offs between local, instance-based reasoning and global, margin-based learning have been also analyzed.

II. Datasets

a. Credit Approval

The first dataset chosen is Credit Approval, from now on denoted as *credit-a*, from the UCI repository [3]. This dataset contains 690 instances and 15 features and is used for classification. Specifically, it contains data from credit card applications, combining continuous and categorical attributes. It also contains missing values.

However, the names and values of the attributes have been encrypted to protect confidentiality. For this reason, the official source [3] does not indicate what each attribute consists of. However, based on [4] and inference of the values, it is hypothesized that each feature corresponds to the descriptions listed in Table 1.

Finally, as can be seen in Table 2, it is a binary classification problem with classes representing whether the credit has been approved or not, and the classes are fairly balanced, with 55.51% of the samples representing class '-' and 44.49% of the samples representing class '+' [3].

b. Pen-Based Recognition of Handwritten Digits

For the second dataset, Pen-Based Recognition of Handwritten Digits [5], from now on denoted as *pen-based*, was chosen. This dataset consists of a database of digits from 44 different writers, with a total of 10,992 instances. To capture how people write numbers, a digitizing tablet (WACOM PL-100V) was used, with a sampling frequency of 100 ms and a resolution of 500×500 pixels. The data captured by the tablet were the x and y coordinates and the pressure level of the pen; however, the latter was ignored in the dataset.

After capturing the points, the coordinates were normalized, making the digits comparable. To do this, translation was eliminated by centering, and the scale was eliminated so that all digits had the same relative size.

One of the problems encountered was that the digits had a different number of captured points. However, in order to use the data in classifiers, all digits must have the same vector size. To achieve this, spatial resampling was used. This method uses linear interpolation to obtain a sequence of equally spaced points in arc lenght, with the digits represented in a 2T-dimensional vector. After testing several values for T, they found that $T = 8$ presented the best balance between accuracy and complexity. Thus, this dataset contains 16 features.

As can be seen in Table 3, there are 10 different classes, representing the digits 0 to 9, which are fairly balanced.

c. TAO-Grid

The TAO-Grid dataset, from now on denoted as *grid*, was used for statistical analysis purposes and consists of a two-dimensional geometric pattern inspired by the Yin-Yang symbol. Each instance corresponds to a point on the plane (x, y coordinates) and it is assigned a class label (`black` or `white`). The dataset consists of 1888 instances and 2 features.

d. Vowel

This dataset was used for statistical analysis and consists of 990 instances and 12 features. One feature indicates whether the instance belongs to the training or test set, another identifies the speaker (15 in total), other the gender and the other remaining correspond to real values [6]. The labels represent 11 classes of English vowels, coded as "h+d" with a central vowel.

e. Preprocessing

For data preprocessing, the training and test files were first merged. This step was necessary to apply the same preprocessing to all folds, which requires the entire dataset to compute global statistics, such as feature means. To do this, the function `merge_fold` was implemented. Additionally, this function also removes instances with more than three missing values, as these were considered to contain excessive missing data for relaible imputation. After this initial step, the *credit-a* dataset contained 684 examples while *pen-based* remained the same as it had no missing values.

Next, the remaining missing values were imputed. On the one hand, in the case of categorical data, two methods were implemented: KNNImputer and imputation using the most frequent value. In the first method, imputation is performed using the k-NN, so that the average value of these is used to impute the missing value. It should be noted that two instances are defined as similar if the features that are not missing present similar values [7] . This method was implemented using the `KNNImputer` [7] class from scikit-learn[1]. The second method simply replaces the missing value with the most frequent value in the column. In this case, the SimpleImputer function [8] from scikit-learn was used, with the `most_frequent` strategy.

On the other hand, for the imputation of numerical data, the scikit-learn's `SimpleImputer` function [8] was again used. This method allows data to be imputed based on the mean, median, most frequent value, and a constant value.

For the normalization of numerical data, scikit-learn's `MinMaxScaler` [9] was used. This transformation scales the data to a given range, in our case [0,1]. The transition is given by Equation 1.

$$X_{\text{scaled}} = X_{\text{std}} \cdot (\max - \min) + \min. \tag{1}$$

The categorical variables were coded using One Hot encoding, transforming each categorical feature into multiple binary columns, representing the different classes. To do this, the `OneHotEncoder` [10] function from scikit-learn was used. It should be noted that for features that were already binary initially, `LabelEncoder` [11], also from scikit-learn, was used instead, converting the classes to 0 and 1.

Finally, the preprocessed data was separated again into training and test for all folds.

Among the different preprocessing methods implemented, `KNNImputer` was selected for imputing categori-

[1]https://scikit-learn.org/stable/

Variable	Type	Description	Missing values
A1	Categorical	Gender {b, a}	Yes
A2	Continuous	Age in years	Yes
A3	Continuous	Debt ratio	No
A4	Categorical	Marital status {u, y, l, t}	Yes
A5	Categorical	Bank Customer {g, p, gg}	Yes
A6	Categorical	Education level {c, d, cc, i, j, k, m, r, q, w, x, e, aa, ff}	Yes
A7	Categorical	Ethnicity {v, h, bb, j, n, z, dd, ff, o}	Yes
A8	Continuous	Years Employed	No
A9	Categorical	Prior Default {t, f}	No
A10	Categorical	Employed {t, f}	No
A11	Continuous	Credit Score	No
A12	Categorical	Drivers License {t, f}	No
A13	Categorical	Citizen {g, p, s}	No
A14	Continuous	Income	Yes
A15	Continuous	Zip Code	No

Table 1: Credit Approval dataset feature description.

Class	Num. of instances	Percentage (%)
+	307	44,49
-	383	55,51

Table 2: Output of Credit Approval dataset

Class	Num. of instances	Percentage(%)
0	1143	10,39
1	1143	10,39
2	1144	10,41
3	1055	9,59
4	1144	10,41
5	1055	9,59
6	1056	9,61
7	1142	10,39
8	1055	9,59
9	1055	9,59

Table 3: Output of Pen-based dataset

cal data, as it better preserves the structure of the data by studying the correlation between instances. For the imputation of numerical data, the median was chosen, as it is a robust measure of central tendency and is less affected by outliers.

For the *credit-a* dataset, given that it had missing values, imputation and normalization were performed. Figure 1 shows the resulting dataset, reduced to two dimensions using t-SNE[2]. In the case of *pen-based*, as there were no missing values, only its attributes were normalized. Figure 2 shows the two-dimensional representation of the preprocessed data.

III. Methods

a. Instance Based Learning (IBL)

The main output of Instance Based Learning (IBL) algorithm is the concept descriptor (CD), which, given a new instance, allows the classification to be produced. This CD can change after processing different instances, deciding to add them according to a similarity function and an update function. These two functions are two of the three key components of IBL algorithms [13]:

1. **Similarity function:** Calculates the similarity between an instance and those stored in the CD.
2. **Classification function:** Produces a class prediction for the instance based on the similarity function.
3. **CD updater:** Decides which instances to include in the CD.

Next, the pseudocode of our IBL model is presented in Algorithm 1.

[2]t-Distributed Stochastic Neighbor Embedding (t-SNE) is a non-linear dimensionality-reduction algorithm used to visualize high-dimensional data in two or three dimensions [12].

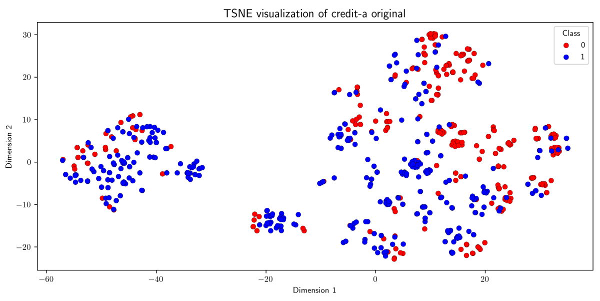

Figure 1: Two-dimensional visualization of the *credit-a* dataset.

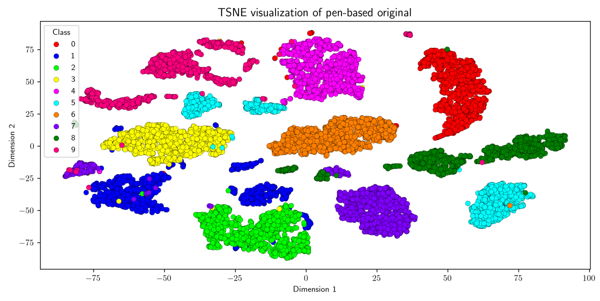

Figure 2: Two-dimensional visualization of the *pen-based* dataset.

Algorithm 1 KIBLALGORITHM$(test_df)$.
Instance-Based Learning prediction process.

```
1: Initialize an empty list $predictions \leftarrow []$
2: for each instance $i$ in $test\_df$ do
3:     $distances \leftarrow$ COMPUTE_DISTANCE$(i)$
4:     $nearest\_outputs \leftarrow$ RETURN_NN$(distances)$
5:     $output \leftarrow$ VOTING_SCHEMA$(nearest\_outputs)$
6:     Append $output$ to $predictions$
7:     UPDATE_CD$(i, output, nearest\_outputs)$
8: end for
9: return list $predictions$
```

Below, we describe the different methods used to compute each of the three main components of IBL.

1. Similarity metrics

Euclidean distance

One of the distance metrics implemented is Euclidean distance. This metric is defined as the distance between two points in Euclidean space, measured as the length of the line segment connecting both points [14], as described in Equation 2.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}. \tag{2}$$

To compute this distance, the `numpy.linalg.norm` function [15] from numpy[3] was used.

Cosine Similarity

The cosine distance can be defined as 1 minus the cosine similarity, and can be represented as shown in Equation 3.

$$\text{Cosine Distance}(\mathbf{A}, \mathbf{B}) = 1 - \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \, \|\mathbf{B}\|}. \tag{3}$$

To calculate the norm of each vector, `numpyp.linalg.norm()` [15] was used, while to calculate the row-by-row dot product, `numpy.einsum()` [16] was employed.

Interpolated Value Difference Metric (IVDM)

Datasets often contain a mix of numerical and categorical attributes, which can be challenging to handle for some distance metrics. To address this issue, we implement the IVDM [17] in our k-IBL algorithm. Unlike Euclidean distance, which requires preprocessing for categorical data, and VDM, which cannot handle continuous numerical attributes, IVDM can handle both types directly.

The main idea of this metric is to keep the continuous nature of numerical attributes when computing class-conditional probabilities, but doing so in a discrete way (as it is done in VDM). During the training phase, continuous features are descretized into a fixed number of same-size intervals. These are later used to estimate the class probabilities of unseen instances, but they do not replace the actual values.

[3] https://numpy.org/doc/stable/

Let $vdm_a(x,y)$ define the distance between two values x and y of a discrete attribute a as:

$$vdm_a(x,y) = \sum_{c=1}^{C} \left| \frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right|^q = \sum_{c=1}^{C} \left| P_{a,x,c} - P_{a,y,c} \right|^q, \tag{4}$$

where:

- $N_{a,x}$ is the number of instances in the training set T that have value x for attribute a.
- $N_{a,x,c}$ is the number of instances in T that have value x for attribute a and output class c.
- C is the number of output classes.
- q is a constant value, usually 1 or 2.
- $P_{a,x,c}$ is the conditional probability that the output class is c given that attribute a has value x ($P(c|x_a)$).

It can be inferred from Equation 4 that $P_{a,x,c}$ is defined as:

$$P_{a,x,c} = \frac{N_{a,x,c}}{N_{a,x}},$$

where $N_{a,x}$ is the sum of $N_{a,x,c}$ over all classes:

$$N_{a,x} = \sum_{c=1}^{C} N_{a,x,c}$$

and the sum of $P_{a,x,c}$ over all C classes is 1, for a fixed value of a and x.

While VDM performs well for nominal attributes, as stated above, it cannot directly handle continuous data. So, IVDM extends VDM by introducing a discretization step that approximates continuous features through interval-based probability estimation.

The number of intervals s, into which we discretize the continuous data, is defined as:

$$s = \max(5, C),$$

where C is the number of output classes.

The width of each interval is given by:

$$w_a = \frac{|\max_a - \min_a|}{s}, \tag{5}$$

where $\max_a$ and $\min_a$ are, respectively, the maximum and minimum value found in the training set T for attribute a.

The number of intervals s controls how finely the attribute is divided, while the width w_a specifies the size of each interval.

We denote the discretized value v of a continuous value x for attribute a as an integer from 1 to s (to which interval it falls into), and it is given by:

$$v = discretize_a(x) = \begin{cases} x & \text{if } a \text{ is discrete,} \\ s & \text{if } x = \max_a, \\ \left\lfloor \frac{(x - \min_a)}{w_a} \right\rfloor + 1 & \text{otherwise.} \end{cases} \tag{6}$$

Once the intervals are defined, the training set is processed to count the number of class occurrences within each discrete interval, and to compute the conditional probabilities $P_{a,v,c}$. The process is summarized in Algorithm 2.

Algorithm 2 LEARNP(training set T).
Computing class-conditional probabilities.

```
 1: for each attribute $a$ do
 2:     for each instance $i$ in $T$ do
 3:         Let $x$ be the input value for attribute $a$ of instance $i$
 4:         $v = discretize_a(x)$                ▷ See Equation 6
 5:         Let $c$ be the output class of instance $i$
 6:         Increment $N_{a,v,c}$ by 1
 7:         Increment $N_{a,v}$ by 1
 8:     end for
 9:     for each discrete value $v$ (of attribute $a$) do
10:         for each class $c$ do
11:             if $N_{a,v} = 0$ then
12:                 $P_{a,v,c} = 0$
13:             else
14:                 $P_{a,v,c} = N_{a,v,c}/N_{a,v}$
15:             end if
16:         end for
17:     end for
18: end for
19: return 3-D array $P_{a,v,c}$
```

IVDM uses these probabilities to estimate the distance between instances. The overall distance is given by the following expression:

$$IVDM(x,y) = \sum_{a=1}^{m} ivdm_a(x_a,y_a)^2, \tag{7}$$

where $ivdm_a$ is defined as:

$$ivdm_a(x,y) = \begin{cases} vdm_a(x,y) & \text{if } a \text{ is discrete,} \\ \sum_{c=1}^{C} |p_{a,c}(x) - p_{a,c}(y)|^2 & \text{otherwise.} \end{cases} \tag{8}$$

For continuous attributes, the probability values $p_{a,c(x)}$ and $p_{a,c(y)}$ are obtained by interpolating between the probabilities of the two nearest discrete intervals, in order to have a smooth transition across the range of the attribute.

The interpolated probability value $p_{a,c}(x)$ of a continuous value x for attribute a and class c is given by:

$$p_{a,c}(x) = P_{a,u,c} + \frac{x - mid_{a,u}}{mid_{a,u+1} - mid_{a,u}} \cdot (P_{a,u+1,c} - P_{a,u,c}) \tag{9}$$

where $mid_{a,u}$ and $mid_{a,u+1}$ are midpoints of two consecutive discretized intervals, such that $mid_{a,u} \leq x < mid_{a,u+1}$.

Then, $P_{a,u,c}$ is the probability value of the discretized range u, or more specifically, the probability value of its midpoint (and similarly for $P_{a,u+1,c}$). This value u is found by doing $u = discretize_a(x)$ and subtracting 1 from it if $x < mid_{a,u}$. The midpoint $mid_{a,u}$ of interval u is computed as follows (recall that w_a is the width of the interval):

$$mid_{a,u} = min_a + w_a + (u + 0.5) \tag{10}$$

This formulation allows IVDM to seamlessly integrate continuous and categorical features within the same distance metric, providing a smooth metric for k-IBL.

2. Voting policies

When it comes to voting policies two were compared: modified plurality and borda count.

Modified Plurality

Modified Plurality is a tie-breaking voting strategy used in lazy learning algorithms. It computes the most frequent class among the k nearest neighbors and if there is a tie, it removes the farthest neighbor. If after recomputing the votes for the $k-1$ neighbors there is still a tie, it removes again the farthest neighbor. The algorithm stops either when a class has the majority or if only one neighbor remains, in which case its class is returned [18].

The pseudocode is represented in Algorithm 3.

Algorithm 3 MODIFIED_PLURALITY($nearest\_outputs$).
Resolving ties in class voting.

```
 1: Count the occurrences of each class in $nearest\_outputs$
 2: Sort the classes by frequency in descending order
 3: Store the result in $list\_app$
 4: while the number of classes in $list\_app > 1$ do
 5:     if the top two classes have the same frequency then   ▷ Tie detected → remove the farthest neighbor
 6:         Remove the last element from $nearest\_outputs$
 7:         Recount and re-sort class frequencies
 8:         Update $list\_app$
 9:     else                                ▷ Clear majority found
10:         return the most frequent class $list\_app[0][0]$
11:     end if
12: end while
13: if only one class remains in $list\_app$ then
14:     return that class $list\_app[0][0]$
15: end if
```

Borda Count

Borda count uses full ranking information provided by the nearest neighbours. To implement so, a descending number of points to candidates based on their rank was assigned: the top-ranked choice received the maximum points ($N-1$), the second-ranked received one point less, and so on, with the last-ranked choice receiving zero.

The function `borda_count(nearest_outputs)` implements also a two-stage tie-breaking mechanism. First, if multiple labels are tied for the highest borda count, a plurality vote is used among those tied labels. This selects the one that appeared most frequently (i.e. has the highest number of "first-place" votes in the K-neighbor list). Second, if a tie persists even after the plurality check (meaning the labels share the same borda count and frequency), the tie is broken by selecting the label that appeared earliest in the `nearest_outputs` list. This final step selects the label belonging to the nearest training instance, with which we aim to favor of the spatially closest data point.

3. Retention policies

The retention policies refer to the set of rules that determine which examples should be stored (retained) in the CD and which should be discarded.

In this work, we developed the code for a total of four retention policies, implemented via the `update_cd(instance, output, nearest_outputs)` function to control the growth of the stored instance set.

Never retain and Always retain

The simplest policies implemented were `never_retain`, which completely discards the new instance to minimize storage, and `always_retain`, which stores every instance presented, behaving like the standard K-NN algorithm.

Different Class

One of the more complex policies implemented is the `different_class` policy, which retains a new instance only if the classification it receives does not match its true class label. This retention policy focuses the CD on instances that lie near the decision boundaries, as these are the ones most likely to be misclassified.

Degree of Disagreement

The other complex policy is `degree_of_disagreement` policy, which calculates a metric that quantifies the conflict among the k nearest neighbors, was implemented. The function created first identifies the `majority_class` among the `nearest_outputs` using a two-stage process. A tie in the frequency count of classes could exist, and for this reason, our code included a tie-breaking mechanism. This mechanism determines the class with the highest frequency (plurality vote), and if a tie occurs, it selects the class belonging to the spatially nearest instance among the tied candidates. Once the unique `majority_class` is chosen, the function calculates the disagreement metric (d). This metric quantifies the ratio of disagreement (neighbors not belonging to the majority class) normalized by the size and strength of the majority vote (see Equation 11). If the calculated value of d meets or exceeds a defined `self.threshold`, the new instance is retained.

$$d = \frac{\texttt{num\_remaining\_cases}}{(\texttt{num\_classes-1}) \cdot \texttt{num\_majority\_class}}. \quad (11)$$

4. Feature weighting

Relief

First, we selected the Relief algorithm using its implementation from scikit-learn [19]. This algorithm belongs to the filter family of methods, which means that they do not depend on a model, but rather evaluate each feature according to the statistical or geometric properties of the data, which makes them fast [20].

Relief is inspired by the k-NN algorithm. For each sample in the dataset, it identifies two neighbors: the closest sample from the same class (called the nearHit, *nH*) and the closest sample from a different class (called the nearMiss, *nM*). The principle is based on the fact that features that differ greatly between *nM* and the sample are given more weight, as they help to distinguish between classes, while features that differ greatly between *nH* and the sample are given less weight, as they are of the same type and are expected to be more similar. Thus, the weights are updated according to Equation 12 [20]:

$$w_f \leftarrow w_f - (x_f - nH_f)^2 + (x_f - nM_f)^2. \quad (12)$$

By doing this, the algorithm subtracts the quadratic difference with *nH* because it wants them to be more similar, and it adds the quadratic difference with *nM* because it wants the classes to be more separated. This process is repeated iteratively for several samples, and finally a list of weights for all features is obtained [20].

Sequential Forward Selection (SFS)

As a wrapper algorithm, we decided to use the Sequential Forward Selection (SFS) algorithm. This approach incrementally builds a subset by evaluating a model, in this case a KNN classifier. The algorithm starts by using only one feature, then selects the one that provides the highest accuracy and repeats the process until it has a certain number of features [21].

To implement it, the `SequentialFeatureSelector` function from mlxtend.feature_selection [22] was used.

It should be noted that SFS was chosen because, as seen in [20] , it is less computationally expensive, tends to choose smaller and more generalizable sets, and presents fewer overfitting problems than other models such as Backward Elimination.

5. Reduction techniques

To address the storage and time complexities in k-IBL, we explored several instance reduction techniques designed to selectively manage the CD. These methods, which reduce the training data, fall into three main categories: condensation, edition, and hybrid techniques.

On the one hand, condensation aims to keep only a small, representative subset of instances essential for defining the classification boundary, for this, we implemented the Modified Condensed Nearest Neighbor (MCNN) algorithm. On the other hand edition techniques, focus primarily on removing noisy or interior instances to smooth decision boundaries and improve robustness, and for this category, we implemented the All K-Nearest Neighbors (All-kNN) algorithm.

Finally, hybrid techniques combine the goals of both condensation and edition to achieve a balanced reduction in size while maximizing classification accuracy, with our approach in this category being the Iterative Case Filter (ICF) algorithm. In this section we are going to now explain how MCNN, All-kNN, and ICF algorithms were implemented.

Condensation: MCNN

MCNN reduces the training set size while ensuring the reduced set of instances (prototypes) can accurately classify the entire original training data using the 1-Nearest Neighbor (1-NN) rule. To do so, we did the MCNN implementation by using an iterative process to find this minimal, representative subset. The algorithm, which can be found in `_condensation.py`, operates through nested loops.

It starts with the entire training dataset (`T`) as the working set (`S_current`) and an empty final set of prototypes (`Q`). In the inner loop, the algorithm repeatedly extracts new prototypes from the current working set (`S_new`). For each class present in `S_new`, it finds the centroid (mean position) and selects the actual data point that is closest to that centroid as a representative prototype. These newly extracted prototypes (`P_current`) are then used to classify all instances in `S_new`. Any instance that is misclassified forms the next working set (`S_misclassified`), forcing the algorithm to find new prototypes from these problematic points until the inner loop successfully classifies all its samples. The outer loop then takes all the prototypes found in the successful inner loop and adds them to the final set `Q`. Moreover, it performs a global consistency check: it uses the entire accumulated set of prototypes (`Q`) to classify every single sample in the original training set (`T`). If `Q` correctly classifies all samples in `T`, the algorithm terminates, and `Q` is returned as the final reduced `CD`. If any samples from `T` are still misclassified, those samples form the new `S_current`, and the entire process restarts. This iterative process guarantees that the final set of prototypes (`Q`) is sufficient to correctly classify all original training data. The notation used for implementing this code is the same as in [23].

Figure 3 shows a two-dimensional representation of the pen-based dataset after applying the MCNN reduction algorithm.

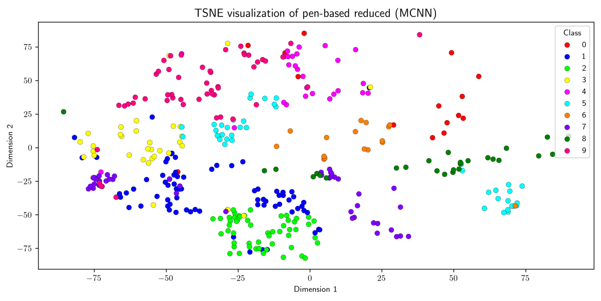

Figure 3: Two-dimensional visualization of the *pen-based* dataset after MCNN feature reduction.

Edition: All-kNN

For the editing method, we have chosen the All-kNN algorithm [24]. For each data sample x, this algorithm determines the label predicted by its k nearest neighbors, with k being a user-defined parameter. Samples which are misclassified by the majority of their neighbors are considered noisy, and are later removed from the CD. This editing step reduces the influence of mislabeled or ambiguous instances, improving the overall reliability of the model. The corresponding pseudocode is provided in Algorithm 4.

Figure 4 shows a two-dimensional representation of the *pen-based* dataset after applying the All-kNN reduction algorithm. As it is observable, the dataset still contains a large number of samples. This is in contrast to ICF, explained in the next section, which makes the CD noticeably sparser.

Algorithm 4 ALL-KNN(D, k)

```
 1: for each sample $x$ in dataset $D$ do
 2:     $i \leftarrow 1$
 3:     flag($x$) $\leftarrow 1$
 4:     while $i \leq k$ do
 5:         Find $i$ nearest neighbors of $x$: $NN(i,x)$
 6:         if majority of $NN(i,x)$ misclassify $x$ then
 7:             flag($x$) $\leftarrow 0$
 8:             break
 9:         end if
10:         $i \leftarrow i+1$
11:     end while
12: end for
13: Remove all samples $x$ from $D$ such that flag($x$) $= 0$
```

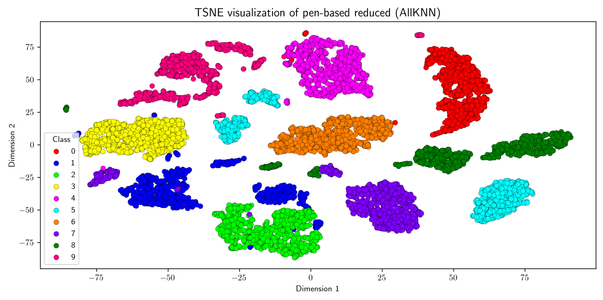

Figure 4: Two-dimensional visualization of the *pen-based* dataset after All-kNN feature reduction.

Hybrid: ICF

As a hybrid model, we decided to use the ICF. This algorithm is used in Case-Based Reasoning (CBR) or lazy learning and, like the previous ones, aims to eliminate redundant or unnecessary cases, keeping the minimum useful ones to classify new examples [25]. However, it should be noted that hybrid models aim to minimize the impact of noise and redundant instances, both near and far from the decision boundaries [26].

The algorithm consists of two main stages. In the first stage, noise filtering (Wilson Editing) is performed, in which cases misclassified by their neighbors are eliminated using k-NN. Specifically, if a case is incorrectly classified by most of its neighbors, it is considered noise or an outlier and is eliminated [25].

In the second stage, redundant cases are eliminated. To do this, the algorithm computes the reachable set and the coverage set for each example. The reachable set consists of the cases that can help to classify the given example, while the coverage set consists of the cases that the given example can help to classify. If a case has a reachable set greater than its coverage set, it means that it is solved by more cases than it solves, so others can do its job and it is considered redundant [25]. In other words, a case is eliminated when more cases can classify it than it helps to classify.

After each iteration, a set with fewer samples is obtained, for which the reachable set and coverage set are recomputed, which are used to remove redundant cases again. The process is repeated until a set is obtained in which none of the cases meet the removal condition [25].

The pseudocode of our proposed ICF model is presented in Algorithm 5.

In Figure 5, the *pen-based* dataset after ICF reduction can be observed. In this case, consistent with hybrid algorithms, both interior and border points have been removed.

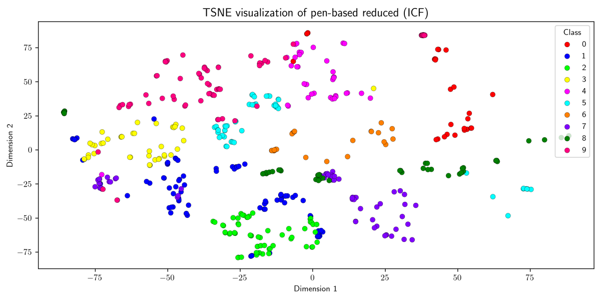

Figure 5: Two-dimensional visualization of the *pen-based* dataset after ICF feature reduction.

It should be mentioned, that ICF was selected instead of IB3 since in the literature and it has been observed that ICF can mantain classification accuracy, while achieving the highest level of instance set reduction [27, 25].

Algorithm 5 ICF class.
Iterative Case Filtering algorithm.

```
1:  function _WILSON_EDITING($X, y, k$)
2:      Train $k$-NN classifier using $(X, y)$ with Euclidean distance
3:      $y_{pred} \leftarrow$ model predictions on $X$
4:      Remove instances where $y_{pred} \neq y$
5:      return filtered $(X, y)$
6:  end function
7:  function _REACHABLE($X, y$)
8:      for each instance $x_j$ in $X$ do
9:          Compute Euclidean distances between $x_j$ and all others
10:         Sort by increasing distance
11:         Add indices of same-class neighbors to reachable_list
12:     end for
13:     return reachable_list
14: end function
15: function _COVERAGE($reachable$)
16:     Initialize coverage as list of empty sets
17:     for each instance $i$ and its reachable set do
18:         for each $r$ in reachable[$i$] do Add $i$ to coverage[$r$]
19:         end for
20:     end for
21:     return coverage
22: end function
23: function REDUCE($data, k$)
24:     Split data into $(X, y)$
25:     $(X, y) \leftarrow$ _Wilson_Editing$(X, y, k)$
26:     repeat
27:         reachable $\leftarrow$ _Reachable$(X, y)$
28:         coverage $\leftarrow$ _Coverage$(reachable)$
29:         Keep instances where $|reachable[i]| \leq |coverage[i]|$
30:     until no further reduction occurs
31:     return reduced dataset $(X, y)$
32: end function
```

b. Support Vector Machine (SVM)

The SVM is a supervised classification algorithm that aims to find the optimal separating hyperplane between classes in the feature space, maximizing the margin defined by the closest data points, known as support vectors. This algorithm was implemented with the function `svmAlgorithm(df_train, df_test, kernel, svc_params,c)`. The model's primary control is the regularization parameter C which controls the penalty for misclassifying training examples: a low C promotes a larger, softer margin that tolerates misclassifications (avoiding overfitting), while a high C enforces a stricter, narrower margin to minimize training error (risking overfitting). On our implementation three different values for C were explored ranging from 0.1, 1.0 and 10.0.

Moreover, we explored the effect of using different kernels: linear, radial basis function (RBF), polynomial (poly), and sigmoid and the effect of their hyperparameters. The suitability of each kernel is highly dependent on the dataset's structure. A brief explanation of the different kernels is provided below to help a better understanding of the results obtained in the Results section [28]:

1. **Linear kernel:** $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$

   Suitability: Best for datasets that are linearly separable or have a **very** large number of features.

   Tuning: Only the regularization parameter C is critical.

2. **Radial Basis Function (RBF):** $K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2}$

   Suitability: Really useful for non-linearly separable data due to its ability to map data to an infinite-dimensional space.

Tuning: The γ parameter controls the influence radius of training examples. A low γ leads to a smooth, potentially underfit boundary, while a high γ creates a complex, potentially overfit boundary γ_{scale} which uses uses $1/(\text{n\_features} \times \mathbf{X}.\text{var}())$ and γ_{auto} which uses $1/\text{n\_features}$ were tried as two different kernel parameters.

3. **Polynomial kernel:** $K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^T \mathbf{x}_j + \text{coef0})^d$

   Suitability: Effective for non-linear problems where features may have a clear polynomial relationship.

   Tuning: The degree (d) is the primary control, a higher d increases complexity and risk of overfitting. coef0 biases the model towards higher or lower order feature interactions. In this case the hyperparameters tried for the kernel were $d = \{1, 3, 5\}$, $\text{coef0} = \{0.0, 1.0\}$, and $\gamma = \{\text{scale}, \text{auto}\}$.

4. **Sigmoid kernel:** $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i^T \mathbf{x}_j + \text{coef0})$

   Suitability: Often performs less reliably than RBF and can be prone to numerical instability when C and γ values are large.

   Tuning: Requires careful tuning of γ and coef0 to maintain numerical stability and avoid non-positive definite results. $\text{coef0} = \{0.0, 1.0\}$, and $\gamma = \{\text{scale}, \text{auto}\}$ were tried.

c. Evaluation metrics

Finally, two different metrics were used to evaluate the models: accuracy and total time. Accuracy measures the proportion of correct classifications with respect to all classifications, as shown in Equation 13. Since it incorporates the four results of the confusion matrix, in balanced datasets such as ours, accuracy can be used as an approximate measure of model quality [29].

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{13}$$

Total time was calculated by recording the start and end times using the `time.time()` function [30] and calculating their difference. This metric allows us to measure which is more optimal in terms of computational speed.

d. Statistical Significance

To assess the statistical significance of the differences in performance between our k-IBL configurations, we usedd non-parametric statistical tests, which are designed to compare multiple algorithms across multiple datasets.

We first performeded the *Friedman test* to determine whether there are significant differences in performance among all the algorithm configurations. The null hypothesis of our test states that all configurations provide the same performance. In case of rejection of this null hypothesis, the Nemenyi post-hoc test was used to identify which specific pairs of configurations had a statistically significant difference between them. We deemed these tests suitable for our analysis as ANOVA's assumptions may not be satisfied when analyzing machine learning algorithm performances [31].

1. Note on datasets

To perform the statistical comparison of all the k-IBL configurations, we initially used the *pen-based* and *credit-a* datasets (the chosen datasets for our project). However, this did not fulfill one of the assumptions of the *Friedman test*, which requires the number of datasets (N) to be greater than 10. To address this limitation, we treated each of the cross-validation folds as an individual dataset, resulting in $N = 20$. Considering the large number of algorithms to compare ($k = 72$), we thought that this ratio of data-to-algorithms was still not optimal. Therefore, we added two more datasets (provided in the collection of possible datasets for the project): `vowel` and `grid`, each divided into 10 folds as well. This increased the number of datasets to $N = 40$.

2. Friedman Test

We used the pre-implemented function for the *Friedman Test* available in Python (from the `scipy.stats` package [32]) to compute the Friedman statistic and the associated p-value. The Friedman test works by ranking the algorithms within each dataset according to their performance, and then computing the average rank of each model across all datasets. It evaluates whether the observed differences in ranks are larger than would be expected by chance. A p-value below the significance level α indicates that at least one algorithm performs significantly differently from others, but it does not give information about which specific pairs are significantly different. This motivates the use of post-hoc tests, such as the *Nemenyi test*, to identify which pairs of algorithms present statistically significant differences.

3. Nemenyi Post-Hoc Test

After the Friedman test indicated significant differences among some of the algorithms, we applied the *Nemenyi post-hoc test* to determine which specific pairs of algorithms were responsible for this result. Nemenyi compares the average ranks of all algorithm pairs, and then determines whether the difference between them exceeds a critical threshold, called the critical difference.

We approached the Nemenyi test in two ways. First, we used a pre-implemented Python function (from the `scikit-posthocs`[4]) to compute the critical difference and identify the pairs of algorithms with statistically significant differences. However, to get a better understanding of the results, we decided to also implement this post-hoc test from scratch, following the formulations of [31].

The average rank R_j for each algorithm j across all datasets is given by:

$$R_j = \frac{1}{N} \sum_{i=1}^{N} r_{i,j},$$

[4]https://scikit-posthocs.readthedocs.io/en/latest/

where $r_{i,j}$ is the rank of algorithm j on dataset i, and N is the number of datasets. We the computed the critical difference CD as:

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

where k is the number of algorithms and q_α is the critical value based on the Studentized range statistic at the chosen significance level α. Values of q_α are typically obtained from pre-computed tables, given the values of k and N.

Then, two algorithms i, j are considered significantly different if the absolute difference of their average ranks is greater than the critical difference CD:

$$|R_i - R_j| > CD.$$

Using this two-step statistical analysis, we were able to assess the differences in performance among all of our k-IBL configurations. The Friedman test confirmed that not all algorithms perform equally as good, and the Nemenyi post-hoc test allowed us to find the specific pairs of algorithms whose differences were statistically significant.

IV. Results

The code for the project is available at: `https://github.com/mariagt02/IML_Labs_2025_MAI`.

All the times reported in this section were obtained using a computer running Ubuntu on an AMD Ryzen 9 7900X 12-core processor. Therefore, the particular results may vary when executed on different hardware or software environments. Nevertheless, the relative performance differences between models should remain consistent.

a. Choosing the best model

Table 4 presents the results of the Friedman test and the Nemenyi post-hoc analysis for different k-IBL hyperparameter configurations, in terms of accuracy at a significance level of $\alpha = 0.1$.

Model (top 5)	**Average Rank**
`euc_mp_3_ar`	22.75
`euc_bc_5_ar`	23.04
`euc_bc_7_dd`	23.19
`euc_bc_5_dd`	23.86
`euc_bc_5_dc`	23.99
Friedman Test Summary	
Statistic	339.26
p-value	1.27×10^{-36}
Reject H_0 at $\alpha = 0.1$?	**Yes**

Table 4: Results of the Friedman and Nemenyi statistical tests for different k-IBL hyperparameter configuration, in terms of accuracy. Average ranks correspond to the Nemenyi post-hoc analysis ($\alpha = 0.1$).

Figure 6 presents the Nemenyi post-hoc significance matrix for different k-IBL model configurations, which displays the pairwise comparison results where a red square indicates a statistically significant difference in performance.

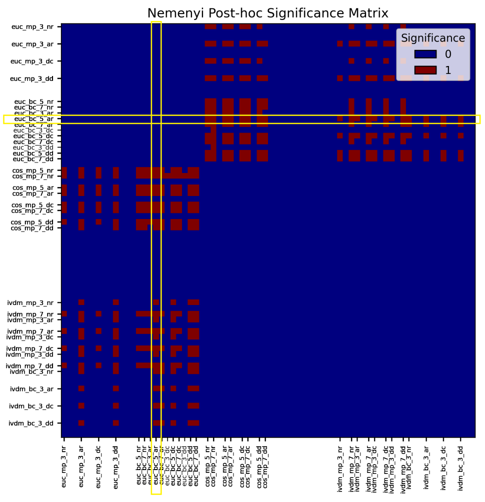

Figure 6: Nemenyi post-hoc significance matrix indicating statistically significant differences (1) and non-significant differences (0) between various method pairs.

Table 5 presents the performance metrics (classification accuracy and total execution time) for nine different k-IBL configurations evaluated across four datasets. The configurations are organized to show the impact of varying a single component (e.g. k, distance metric, instance weighting) relative to the baseline configuration, `euc_bc_5_ar`.

Configuration	Accuracy (%)	Total time (s)
	credit-a	
euc_bc_3_ar	82.39	0.09
euc_bc_5_ar	**85.22**	**0.09**
euc_bc_7_ar	85.37	0.09
cos_bc_5_ar	83.58	0.10
ivdm_bc_5_ar	86.72	0.97
euc_mp_5_ar	85.07	0.08
euc_bc_5_nr	85.37	0.08
euc_bc_5_dc	85.07	0.08
euc_bc_5_dd	85.37	0.08
	pen-based	
euc_bc_3_ar	99.12	8.56
euc_bc_5_ar	**99.11**	**8.15**
euc_bc_7_ar	99.13	8.14
cos_bc_5_ar	99.14	9.40
ivdm_bc_5_ar	98.45	436.73
euc_mp_5_ar	99.01	8.44
euc_bc_5_nr	99.12	7.40
euc_bc_5_dc	99.13	8.32
euc_bc_5_dd	99.11	7.49
	grid	
euc_bc_3_ar	96.22	0.19
euc_bc_5_ar	**97.23**	0.19
euc_bc_7_ar	97.13	0.19
cos_bc_5_ar	56.12	0.22
ivdm_bc_5_ar	96.76	1.83
euc_mp_5_ar	97.13	0.19
euc_bc_5_nr	96.97	0.18
euc_bc_5_dc	97.07	0.18
euc_bc_5_dd	97.02	0.19
	vowel	
euc_bc_3_ar	99.19	0.13
euc_bc_5_ar	**97.68**	**0.13**
euc_bc_7_ar	97.27	0.13
cos_bc_5_ar	97.37	0.15
ivdm_bc_5_ar	92.22	1.75
euc_mp_5_ar	94.04	0.13
euc_bc_5_nr	97.07	0.11
euc_bc_5_dc	97.47	0.11
euc_bc_5_dd	97.58	0.12

Table 5: Performance comparison of various distance metrics and configurations across four datasets. Each block reports classification accuracy (%) and total computation time (seconds). Bold values correspond to the euc_bc_5_ar configuration.

Weighting method	Average Rank
`SFS`	1.813
`None`	1.888
`Relief`	2.3
Friedman Test Summary	
Statistic	6.89
p-value	0.0318
Reject H_0 at $\alpha = 0.1$?	**Yes**

Table 6: Results of the Friedman and Nemenyi statistical tests for `euc_bc_5_ar` model with feature weighting techniques, in terms of accuracy. Average ranks correspond to the Nemenyi post-hoc analysis ($\alpha = 0.1$).

Weighting method	Average Rank
`SFS`	1.25
`None`	1.75
`Relief`	3.00
Friedman Test Summary	
Statistic	65.0
p-value	7.6812×10^{-15}
Reject H_0 at $\alpha = 0.1$?	**Yes**

Table 7: Results of the Friedman and Nemenyi statistical tests for `euc_bc_5_ar` model with feature weighting techniques, in terms of execution time. Average ranks correspond to the Nemenyi post-hoc analysis ($\alpha = 0.1$).

b. Feature weighting vs. best model

Table 6 shows the results of the Friedman and Nemenyi for the `euc_bc_5_ar` model with different feature weighting techniques in terms of accuracy at a significance level $\alpha = 0.1$.

Table 7 shows the results of the Friedman and Nemenyi for the `euc_bc_5_ar` model with different feature weighting techniques in terms of execution time at a significance level $\alpha = 0.1$.

Table 8 presents the performance metrics in terms of accuracy and execution time of the `euc_bc_5_ar` model with SFS feature weighting algorithm against the baseline model.

Model	Time (s)	Accuracy (%)
	credit-a	
SFS	16.01	**86.41**
None	**0.08**	85.22
	pen-based	
SFS	52.76	**99.14**
None	**8.77**	99.11
	vowel	
SFS	9.69	**96.96**
None	**0.12**	97.67
	grid	
SFS	0.62	**97.23**
None	**0.19**	97.23

Table 8: Performance comparison of feature weighting for the `euc_bc_5_ar` model across four datasets. Each block reports total computation time (seconds) and classification accuracy (%).

c. Reduced vs. best model

Table 9 shows the results of the Friedman and Nemenyi tests for the `euc_bc_5_ar` model with different instance reduction techniques at a significance level $\alpha = 0.01$

Figure 7, shows the Nemenyi post-hoc significance matrix for the `euc_bc_5_ar` model with different instance reduction techniques, at significance level $\alpha = 0.01$.

Table 10 shows the performance of the same `euc_mp_5_ar` in terms of average time and accuracy across the different folds for the four datasets.

Model	Average Rank
None	1.625
All-kNN	2.300
ICF	2.800
MCNN	3.275
Friedman Test Summary	
Statistic	37.02
p-value	4.56×10^{-8}
Reject H_0 at $\alpha = 0.01$?	**Yes**

Table 9: Results of the Friedman and Nemenyi tests for `euc_bc_5_ar` with different instance reduction techniques. Average ranks are obtained from the Nemenyi post-hoc analysis ($\alpha = 0.01$).

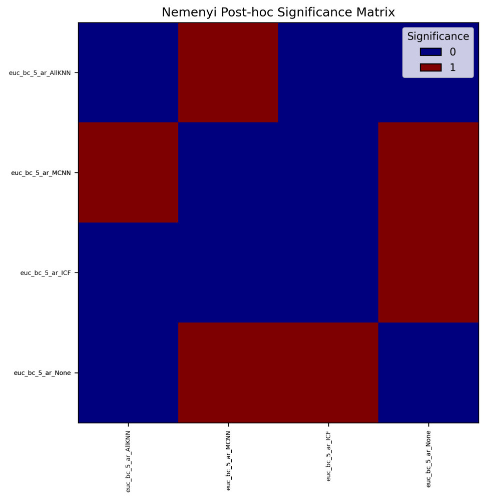

Figure 7: Nemenyi post-hoc significance matrix indicating statistically significant differences (1) and non-significant differences (0) between various reduction pairs.

Reduction	Time (s)	Accuracy (%)	# Samples
	credit-a		
None	**0.08**	85.22	613
All-kNN	0.67	84.62	458
ICF	0.59	**87.46**	143
MCNN	2.68	83.28	210
	pen-based		
None	**9.38**	**99.11**	9894
All-kNN	107.36	98.98	9799
ICF	96.26	98.61	538
MCNN	34.57	98.30	383
	vowel		
None	**0.12**	**97.67**	891
All-kNN	1.08	88.98	834
ICF	1.52	93.43	763
MCNN	0.88	68.08	212
	grid		
None	**0.18**	97.23	1700
All-kNN	2.12	**97.76**	1615
ICF	2.45	91.06	197
MCNN	3.83	96.06	149

Table 10: Performance comparison of instance reduction techniques across datasets for the `euc_bc_5_ar` model. Each block reports total computation time (seconds), accuracy (%), and number of selected samples.

d. Selecting the best SVM configuration

Table 11 shows the results of the Friedman and Nemenyi tests for different SVM configurations at a significance level $\alpha = 0.1$.

Model (top 5)	Average Rank
`rbf_c_10.0_`γ`-scale`	6.44
`poly_c_0.1_deg-5_coef0-1.0_`γ`-scale`	8.79
`poly_c_1.0_deg-3_coef0-1.0_`γ`-scale`	8.81
`poly_c_10.0_deg-3_coef0-1.0_`γ`-scale`	10.93
`poly_c_10.0_deg-3_coef0-0.0_`γ`-scale`	11.55
Friedman Test Summary	
Statistic	1422.23
p-value	1.41×10^{-260}
Reject H_0 at $\alpha = 0.1$?	**Yes**

Table 11: Results of the Friedman and Nemenyi tests for SVM configurations. Average ranks correspond to the Nemenyi post-hoc analysis ($\alpha = 0.1$).

e. SVM best vs. k-IBL best

Table 12 shows the performance of the best k-IBL algorithm (`euc_bc_5_ar`) against the best SVM (`svm.svc_rbf_c_10.0_gamma-scale`), in terms of accuracy and total computation time.

Configuration	Accuracy (%)	Total time (s)
	credit-a	
Best k-IBL	85.22	**0.08**
Best SVM	**87.33**	0.13
	pen-based	
Best k-IBL	99.11	9.38
Best SVM	**99.63**	**2.42**
	vowel	
Best k-IBL	97.67	**0.12**
Best SVM	**98.18**	0.31
	grid	
Best k-IBL	**97.23**	**0.18**
Best SVM	92.37	0.28

Table 12: Performance comparison of the best SVM (`svm.svc_rbf_c_10.0_gamma-scale`) against the best k-IBL model (`euc_bc_5_ar`). Each one is evaluated in terms of classification accuracy (%) and total computation time (seconds).

f. SVM vs. SVM reduced dataset

Table 13 shows the Friedman and Nemenyi statistical tests comparison of SVM with different instance reduction techniques, at a significance level $\alpha = 0.1$.

Figure 8, shows the Nemenyi post-hoc significance matrix for SVM with different instance reduction techniques at a significance level $\alpha = 0.1$.

Table 14 shows the comparison of SVM with different instance reduction techniques, in terms of accuracy and total computation time, using as baseline the `svm.svc_rbf_c_10.0_gamma-scale` model.

Model	Average Rank
None	1.50
All-kNN	2.16
ICF	3.00
MCNN	3.34
Friedman Test Summary	
Statistic	53.44
p-value	1.48×10^{-11}
Reject H_0 at $\alpha = 0.1$?	**Yes**

Table 13: Results of the Friedman and Nemenyi statistical tests for the SVM (RBF kernel, $C = 10$, γ =scale) with instance reduction techniques. Average ranks correspond to the Nemenyi post-hoc analysis ($\alpha = 0.1$).

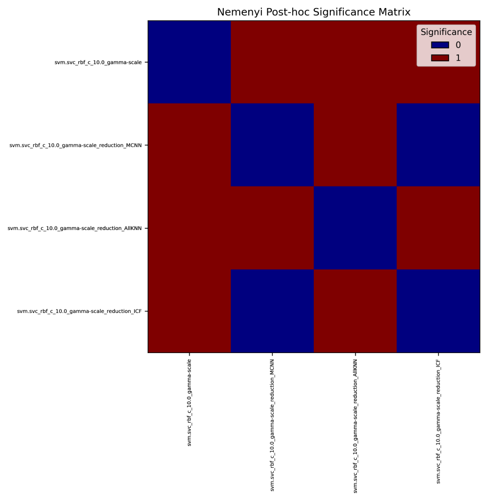

Figure 8: Nemenyi post-hoc significance matrix indicating statistically significant differences (1) and non-significant differences (0) between various reduction pairs.

Reduction	Time (s)	Accuracy (%)	# Samples
		credit-a	
None	**0.14**	**87.33**	613
All-kNN	0.75	87.32	458
ICF	0.77	86.30	143
MCNN	2.83	85.11	210
		pen-based	
None	**1.23**	**99.63**	9894
All-kNN	72.59	99.52	9799
ICF	100.94	99.06	538
MCNN	35.18	99.26	383
		vowel	
None	**0.25**	**98.18**	891
All-kNN	1.22	91.51	834
ICF	1.74	94.94	763
MCNN	0.97	86.86	212
		grid	
None	**0.22**	**92.37**	1700
All-kNN	2.21	92.15	1615
ICF	2.58	79.71	197
MCNN	3.96	85.11	149

Table 14: Performance comparison of SVM (RBF kernel, $C = 10$, γ =scale) with different instance reduction methods. Each block reports total computation time (seconds), accuracy (%), and number of selected samples.

V. Discussion

a. Choosing the best model

To definitively select the optimal model from our pairwise comparison set, we first employed the Nemenyi post-hoc test, following the initial rejection of the null hypothesis by the Friedman test as it can be seen in Table 4. Small confidence level (α) values (such as 0.05 or 0.01) did not yield statistically significant differences among the models. Therefore, to ensure that the test was sensitive enough to identify meaningful distinctions within the full set of models, we utilized an α value of 0.1, corresponding to a 90% confidence level.

The resulting Nemenyi significance matrix (see Figure 6) indicated that significant differences ($\alpha = 0.1$) exist between the top-performing models and the worst-performing variants. However, no single model demonstrated a clear and statistically significant advantage over all other models.

As there was not a significantly superior model, we decided to analyze the average ranks provided by the Friedman test. The overall rank differences across most algorithms were small, which is expected as the majority of the models share a common structure, differing only in minor factors such as the number of nearest neighbors (k), voting, or retention policies.

Focusing on the top five ranked algorithms (where the average ranks ranged tightly from 22.75 to 23.98), a pattern was observed regarding the most effective configurations. The majority of these top-tier models (specifically four out of five) consistently utilized the Euclidean distance metric coupled with the borda count aggregation method. Since the differences introduced by varying k and the specific retention policies were negligible among the top group, we decided to select the two highest-ranked models to represent the best configurations. The final model chosen based on the highest rank and consistency was `euc_bc_5_ar`, which uses the Euclidean distance metric, borda count aggregation, a neighborhood size of $k = 5$, and the 'always retain' policy (model shown in yellow in Figure 6).

The fact that `euc_bc_5_ar` is a really optimal configuration for our datasets can be better understood by observing the performance changes across the top variants (see Table 5). The superior performance of the Euclidean metric, even after feature normalization, indicates that the simple, direct geometric separation in the scaled space is the most effective measure of dissimilarity, as it had superior results to both angle-based (cosine) and complexity-weighted (IVDM) distance measures. Next, the choice of $k = 5$ (compared to $k = 3$ or $k = 7$) provides an optimal balance, offering sufficient neighborhood context to suppress local noise while still maintaining the integrity of the local class boundaries. A lower value like $k = 3$ would be highly sensitive to local outliers, whereas $k = 7$ might look at neighbors that are 'too far away', potentially blurring decision boundaries. Furthermore, the selection of the borda count aggregation method suggests that the proximity of the neighbors is highly informative in our dataset, as borda count gives a weighted importance to the ranked neighbors based on distance. Finally, the 'always retain' retention policy proved most effective. By retaining every instance, this policy ensures that every local piece of information is kept available for the classifier, maximizing the resolution of the decision space.

b. Feature weighting vs. best model

After selecting the best configuration for the baseline k-IBL algorithm, we are left with just 3 models to compare regarding feature weighting. This is a substantial decrease from the 72 models from the previous case, which significantly simplifies the comparisons. Performing the statistical tests with only the original 2 datasets would be enough. However, to maintain consistency with the previous section, we still present the results using all 4 datasets, each divided into 10 folds.

At significance level $\alpha = 0.01$ we cannot reject the null hypothesis of the Friedman test. Therefore, we do not perform any post-hoc tests. On the other hand, both for significance levels $\alpha = 0.05$ and $\alpha = 0.1$, we are able to reject the null hypothesis, and therefore compute the corresponding post-hoc test. The post-hoc test for significance level $\alpha = 0.05$ does not report significant differences, but at $\alpha = 0.1$ we observe a significant difference between the top and bottom performing feature weighting techniques (SFS and Relief, respectively). We present the results for the Friedman and Nemenyi tests (in terms of accuracy) in Table 6.

Although the statistical test shows a significant difference between SFS and Relief, if we take a look at their average ranks we'll notice that actually they are very similar. This indicates that while SFS has a slightly better accuracy than Relief, the improvement is not big enough to have a noticeable impact when applying these techniques in practice. Considering how these algorithms work, this is not a surprise.

On the one hand, SFS is a wrapper-based method that iteratively evaluates subsets of features to try and find a combination that performs well. However, this procedure is costly in terms of computation time. On the other hand, Relief is a filter-based method, which is computationally lighter, but features are not optimized directly for the final model so it may obtain slightly worse accuracy than SFS [33]. The model with no feature weighting uses all features equally and, in our experiment, it performs almost as good as SFS. This would indicate that most of the features are already meaningful, and adding feature weighting does not result in considerable improvements.

The feature weighting step adds computational overhead, so we decided to perform a statistical analysis in terms of execution time too. The results of this test are shown in Table 7. Results of this test show that all pairwise comparison are significantly different in terms of execution time.

There is a significant difference between SFS and the model without feature weighting. So, we can say that while the difference in accuracy between the models is small, their execution times differ significantly.

Table 8 presents the execution time and accuracy metrics for both SFS and no feature weighting for all datasets. In most cases, average performance for SFS is slightly better than the model with no feature weighting, but we reckon this difference would be not significant in practice. However, we notice a big difference in time. This is explained by how SFS works, as it performs model evaluations to determine the best subset of features, increasing the computational effort. In contrast, the model without feature weighting does the classification directly, without any previous steps that would increase its execution time.

Considering that IBL is already slow during inference, as it is a lazy learner, adding more overhead with feature weighting could make the model not suitable for large datasets. Relief would provide more balanced results in terms of accuracy and execution time, but we believe that the baseline model, without feature weighing, is the most efficient approach, as it offers practically the same accuracy with much better execution time.

c. Reduced vs. best model

Based on the findings provided in the previous two sections, the configuration selected as the optimal for our k-IBL model uses an Euclidean distance metric, borda count voting policy, a neighborhood size of $k = 5$ and an always-retain retention policy. This combination (`euc_bc_5_ar`) has showed a higher performance across all datasets, serving as the benchmark for comparison against various instance reduction techniques described in Section 5.

The Friedman test comparing the best k-IBL configuration trained on the full dataset against the same configuration trained on datasets reduced through different instance reduction techniques (see Table 9) allows us to reject the null hypothesis at $\alpha = 0.01$. This indicates that training set size, as affected by the reduction methods, lead to statistically significant performance differences. The subsequent Nemenyi post-hoc analysis identifies three significant pairwise differences, specifically between the configuration trained on the complete dataset and those trained with the MCNN and ICF reduction techniques. According to the average ranks, the model trained without instance reduction achieves the lowest rank, followed by All-kNN, ICF and MCNN. Although the difference between the full dataset and All-kNN is not statistically significant, selecting the configuration without reduction remains the best option, as it performs significantly better than most reduced variants with a confidence level of 99%.

The average ranking of each reduction technique appears to be strongly influenced by the degree of data reduction achieved, as shown in Table 9. Methods that retain a larger portion of the original training dataset, such as All-kNN, tend to have a higher predictive performance, whereas more aggressive reduction strategies like ICF and MCNN, which remove a higher number of instances, generally lead to a lower accuracy. This relationship suggests that the k-IBL model benefits from having access to the complete set of training instances, as the availability of more data points allows for better generalization and more reliable neighborhood-based decision.

Another relevant aspect to consider is the computational efficiency introduced by instance reduction. Reducing the number of training samples can potentially decrease the overall runtime of the model, as fewer comparisons are required during prediction. However, the reduction process itself has also an associated computational cost, meaning that a balance must be achieved between preprocessing time and inference efficiency. As observed in Table 10, the total execution time is substantially higher when applying reduction techniques, specially for larger datasets such as *pen-based* or *grid*. This suggests that the additional overhead of performing instance reduction outweights the potential runtime benefits during classification. Given the accuracy of the model trained on the complete datasets remains consistently higher across all of them, we have not included any further time-related trade-offs in this discussion. Regarding storage efficiency, the datasets employed here are relatively moderate in size, so the reduction in memory usage is not critical. Nonetheless, in real-world scenarios, storage constraints could make a more relevant strategy to explore, even at the cost of a downgrade in the performance of the resulting models.

Overall, based on the analysis presented in these discussions sections, we conclude that the best configuration for our k-IBL algorithm uses an Euclidean distance metric, a borda count voting policy, a neirghborhood size of $k = 5$ and an always retain retention policy. Additionally, no feature weighting or instance reduction is applied, as this setup consistently yield the highest and statistically significant accuracy and overall performance across all the evaluated datasets. In the following sections, we will evaluate how this optimal configuration performs when compares against an SVM model.

d. Selecting the best SVM configuration

The results of the Friedman test comparing different SVM configurations are summarized in Table 11. The test statistic allows us to reject the null hypothesis at $\alpha = 0.1$, confirming that not all SVM configurations perform equally. The subsequent Nemenyi post-hoc analysis further reveals the presence of significant pairwise differences among some configurations.

The top-performing model corresponds to the `rbf_c_10.0_`γ`-scale` configuration, which achieves the lowest average rank, followed by several polynomial kernels with varying degrees and regularization parameters. Although the RBF configuration has obtained the best overall ranking, the post-hoc analysis indicates that its performance is not statistically different from the other top models within the 10% significance threshold. This does not imply that RBF model is not superior, but rather that there is a 10% probability that the observed advantage is due to random variation rather than a true performance difference.

The higher performance of the RBF kernel can be attributed to its flexibility in handling non-linear decision boundaries. This is is particularly noticeable in the *grid* dataset, which shows highly non-linear class separations that cannot be properly captured by linear or low-degree polynomial kernels as shown in Figure 9. By mapping

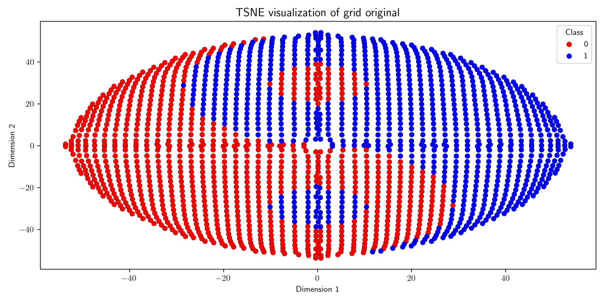

Figure 9: Two-dimensional visualization of the *grid* dataset.

data into an infinite dimensional feature space, the RBF kernel can effectively separate complex patterns that the linear and polynomial kernels may fail to capture. Furthermore, the relatively high regularization parameters ($C = 10$) allows the model to fit the training data more closely.

Based on these findings the the `rbf_c_10.0_`γ`-scale` configuration is selected as the optimal SVM setup for subsequent comparisons with the best-performing k-IBL model and the SVM trained on a reduced dataset.

e. SVM best vs. k-IBL best

As can be seen in Table 12, SVM consistently outperforms the k-IBL model in terms of accuracy for the *credit-a*, *pen-based*, and vowel datasets, while k-IBL performs better on the *grid* dataset. In the *credit-a* (see Figure 1), *pen-based* (see Figure 2) and vowel datasets, the samples form non-linear and in some cases, overlapping clusters. These conditions, difficult the definition of the boundaries in the k-IBL methods. However, SVM using its RBF kernel, is able to effectively capture these complex decision boundaries, resulting in superior generalization.

Regarding the *grid* dataset, it presents a regular and structured pattern of the Yin-Yang shape, which facilitates class separation based on geometric proximity. In this scenario, local instance relationships dominate, and the k-IBL model can effectively classify the samples, achieving higher accuracy than SVM.

Overall, SVM demonstrates stronger performance in terms of accuracy in datasets with complex and overlapping class boundaries, while k-IBL outperforms in geometrical structured organized data. However, it should be mentioned, that the difference in accuracy between k-IBL and SVM for the *credit-a*, *pen-based* and vowel datasets remains relatively small, below than 2.5.

In terms of total time, Table 12 shows that k-IBL model is consistently faster, in datasets with a relatively small number of features. However, it is seen that the total time for large datasets such as *pen-based* is much higher for k-IBL than for SVM. This could be due to the fact that k-IBL is a lazy learning method, which performs minimal computation during training, but requires extensive comparisons during testing by evaluating each sample against CD instances. Consequently, the model presents a very low training time, but a high testing time, which increases substantially when the dataset presents an increased number of samples. Conversely, SVM is an eager learning, that performs most computations during training, but then it makes classifications fast based on the trained model. This makes SVM suitable for large databases.

In conclusion, SVM offers better metrics in terms of time and accuracy in complex datasets and large sample sizes, while k-IBL is effective for small datasets with structured data.

f. SVM vs. SVM reduced dataset

As can be seen in Table 14, the baseline model, without reduction, achieved the highest accuracy across all datasets. Moreover, it is observed that although, the number of instances is reduced and therefore the training time of the algorithm is lower, when the reduction processing time is included, the total computational time becomes considerably higher than the baseline model. Among the reduction methods, *credit-a* and *pen-based* datasets exhibited similar accuracy across methods, while ICF achieved the best accuracy for the vowel dataset, and All-kNN obtained the best accuracy for the *grid* dataset.

To extract more informed conclusions, the Friedman and Nemenyi post-hoc test were computed. As can be observed in Table 13, the Friedman test produced a statistic of 53.44, with a p-value much more lower than the significance level ($\alpha = 0.1$). This result leads to the rejection of the null hypothesis confirming that there are statistical significant differences among the SVM models. The baseline SVM, obtained an average rank of 1.5 followed by a 2.16 from All-kNN. The ranking results show that performance decreases as more instances are removed (see Table 14). Regarding the Nemenyi significance matrix (see Figure 8), it is observed a strong presence of red cells (statistically significant differences) between the baseline SVM and the reduced models, confirming that the baseline is statistically superior.

It should be mentioned, that although the resulting Friedman statistic was identical for all significance levels (0.01, 0.05 and 0.1), the Nemenyi post-hoc significance matrices differed, due to the different critical differences. At $\alpha = 0.1$, the matrix shows a clear hierarchy among algorithms (see Figure 8), therefore adopting 0.1 for alpha is reasonable as it provides a higher discrimination between models.

In summary, it is seen that in terms of accuracy and total time instance reduction offers limited benefit. However, these techniques could be useful in resource-constrained scenarios such as memory limits.

VI. Conclusions

With this work, we conducted a comprehensive analysis of different configurations of the k-IBL algorithm and SVM. The results showed that the most effective tested configuration uses Euclidean as distance metric, Borda Count as voting policy, a number of neighbors equal to 5, and always retain as retention policy. For the SVM analysis, the best accuracy was obtained with an RBF kernel, with a regularization parameter $C = 10$ and a γ set to scale. When comparing both models, it was found that SVM outperforms k-IBL in scenarios with complex and overlapping class boundaries and large datasets, while k-IBL exhibits a higher accuracy on geometrically structured datasets with fewer number of samples, such as the *grid* dataset.

Feature weighting made with SFS for the k-IBL algorithm provided a minor improvement in terms of accuracy, but it introduced more computational load, which led to higher execution times. Instance reduction techniques used, introduced performance degradation, with lower accuracy values and higher computational times. Consequently, the baseline configurations proved to be the most balanced in terms of performance and computational time. Moreover, although it is not part of the analysis, since the overall results did not improve with feature weighting and neither with instance reduction, it is expected that the combination of both also does not improve the performance.

The main limitations of this study include the limited number of parameters and methods tested. Incorporating additional distance metrics, a broader set of SVM parameters or more instance reduction or feature weighting techniques could further enrich the analysis. Moreover, the k-IBL algorithm is a lazy learning model that requires significant computational time and resources during inference time. In cases in which memory is limited and the dataset is large, saving all the training samples during the testing time could not be feasible, and therefore the use of an eager algorithm such as SVM would be a better option. Finally, when performing the Friedman and Nemenyi post-hoc tests, the folds were used as different datasets although they were sampled from the same database. This could devalue the statistical robustness of the results.

References

[1] D. W. Aha, D. Kibler, and M. K. Albert. “Instance-based learning algorithms”. In: *Machine Learning* 6.1 (1991), pp. 37–66.

[2] C. J. Burges. “A tutorial on support vector machines for pattern recognition”. In: *Data Mining and Knowledge Discovery* 2.2 (1998), pp. 121–167.

[3] UCI Machine Learning Repository. *Credit Approval Data Set*. Accessed: 2025-11-01. n.d. URL: https://archive.ics.uci.edu/dataset/27/credit+approval.

[4] R. Kuhn. *Analysis of Credit Approval Data*. Accessed: 2025-11-01. n.d. URL: https://rstudio-pubs-static.s3.amazonaws.com/73039_9946de135c0a49daa7a0a9eda4a67a72.html.

[5] UCI Machine Learning Repository. *Pen-Based Recognition of Handwritten Digits Data Set*. Accessed: 2025-11-01. n.d. URL: https://archive.ics.uci.edu/dataset/81/pen+based+recognition+of+handwritten+digits.

[6] OpenML. *Vowel Dataset*. https://www.openml.org/search?type=data&sort=runs&id=307&status=active. Accessed: 2025-11-02. n.d.

[7] Scikit-learn. *KNNImputer*. Accessed: 2025-11-01. n.d. URL: https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html.

[8] Scikit-learn. *SimpleImputer*. Accessed: 2025-11-01. n.d. URL: https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html.

[9] Scikit-learn. *MinMaxScaler*. http://scikit-learn.org/sklearn.preprocessing.MinMaxScaler.html. Accessed: 2025-11-02. n.d.

[10] Scikit-learn. *OneHotEncoder*. Accessed: 2025-11-01. n.d. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html.

[11] Scikit-learn. *LabelEncoder*. Accessed: 2025-11-01. n.d. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html.

[12] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[13] David W Aha, Dennis Kibler, and Marc K Albert. "Instance-based learning algorithms". In: *Machine learning* 6.1 (1991), pp. 37–66.

[14] GeeksforGeeks. *Euclidean Distance*. Accessed: 2025-11-01. GeeksforGeeks. July 2025. URL: https://www.geeksforgeeks.org/maths/euclidean-distance/.

[15] NumPy Developers. *numpy.linalg.norm — NumPy v2.4.dev0 Manual*. Accessed: 2025-11-01. n.d. URL: https://numpy.org/devdocs/reference/generated/numpy.linalg.norm.html.

[16] NumPy Developers. *numpy.einsum — NumPy v2.3 Manual*. Accessed: 2025-11-01. n.d. URL: https://numpy.org/doc/stable/reference/generated/numpy.einsum.html.

[17] D Randall Wilson and Tony R Martinez. "Improved heterogeneous distance functions". In: *Journal of artificial intelligence research* 6 (1997), pp. 1–34.

[18] Maria Salamó Llorente. *Introduction to Machine Learning: Slides Support Work 2 (Session 2) (course 2025-2026)*. Practical slides. Master in Artificial Intelligence, UPC, UB, URV. 2025. URL: https://campusvirtual.ub.edu/pluginfile.php/9896033/mod_resource/content/9/SuportWork2_2025_Session2.pdf.

[19] Client Challenge. *sklearn-relief*. https://pypi.org/project/sklearn-relief/. Accessed: 2025-11-02. n.d.

[20] Padraig Cunningham, Bahavathy Kathirgamanathan, and Sarah Jane Delany. "Feature selection tutorial with python examples". In: *arXiv preprint arXiv:2106.06437* (2021).

[21] Sahameh Shafiee et al. "Sequential forward selection and support vector regression in comparison to LASSO regression for spring wheat yield prediction based on UAV imagery". In: *Computers and Electronics in Agriculture* 183 (2021), p. 106036.

[22] S Raschka. *mlxtend.feature_selection — Mlxtend*. Accessed: 2025-11-01. n.d. URL: https://rasbt.github.io/mlxtend/api_subpackages/mlxtend.feature_selection/.

[23] V. S. Devi and M. N. Murty. "An incremental prototype set building technique". In: *Pattern Recognition* 35.2 (2002), pp. 505–513. DOI: 10.1016/S0031-3203(01)00058-5.

[24] Ivan Tomek. "An experiment with the edited nearest-nieghbor rule." In: (1976).

[25] Henry Brighton and Chris Mellish. "Advances in instance selection for instance-based learning algorithms". In: *Data mining and knowledge discovery* 6.2 (2002), pp. 153–172.

[26] Soft Computing and Intelligent Information Systems Group. *Prototype Reduction in Nearest Neighbor Classification: Prototype Selection and Prototype Generation | Soft Computing and Intelligent Information Systems*. Accessed: 2025-11-01. n.d. URL: https://sci2s.ugr.es/pr.

[27] Zong-Yao Chen et al. "Instance selection by genetic-based biological algorithm". In: *Soft Computing* 19.5 (2015), pp. 1269–1282.

[28] scikit-learn Developers. *Plot classification boundaries with different SVM Kernels*. https://scikit-learn.org/stable/auto_examples/svm/plot_svm_kernels.html. Accessed: 2025-11-01. 2025.

[29] Google for Developers. *Classification: Accuracy, Recall, Precision, and Related Metrics*. Accessed: 2025-11-01. n.d. URL: https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall#accuracy.

[30] Python Software Foundation. *time — Time Access and Conversions*. Accessed: 2025-11-01. n.d. URL: https://docs.python.org/3/library/time.html.

[31] Janez Demšar. "Statistical comparisons of classifiers over multiple data sets". In: *Journal of Machine learning research* 7.Jan (2006), pp. 1–30.

[32] *SciPy v1.16.2 Manual: Statistical functions, friedmanchisquare*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.friedmanchisquare.html. [Accessed 02-11-2025].

[33] Maria Salamó Llorente. *Introduction to Machine Learning: Theory 6. Feature Selection*. Lecture slides. Master in Artificial Intelligence, UPC, UB, URV. 2025. URL: `https://campusvirtual.ub.edu/pluginfile.php/9895961/mod_resource/content/10/T6_2025_students.pdf`.