Implementing Various k-NN Algorithms and Optimisations using the Waveform Dataset

Patrick Barry Buu Dinh Ha

Abstract

This report will discuss the process of applying various machine learning techniques on the waveform dataset. The techniques and algorithms that will be covered during this report are: k-nearest neighbours (k-NN), data reduction techniques, using the triangle inequality to speed up calculations, and analysing the effect of sampling methods on the dataset.

1. Introduction

This report investigates the application of the k-Nearest Neighbors (k-NN) algorithm and several optimization techniques to the waveform dataset. k-NN is a non-parametric, instance-based learning algorithm widely used for classification and regression tasks. Its simplicity makes it a popular choice for various machine learning applications.

This project explores four different methods to improve the performance and efficiency of k-NN: 1) tune the number of neighbours for k using k-fold cross-validation, 2) implementing data reduction techniques to improve efficiency, 3) implement the triangle inequality to improve performance, 4) analyse the effects of oversampling and undersampling.

The waveform dataset contains 5000 examples, 3 classes (equally represented), 20 continuous variables and a optimal Bayesion classification rate of 86%.

The report is structured as follows: Section 2 describes the experimental setup, including data exploration and methodology, Section 3 discusses the results; and Section 4 concludes, summarizing findings, acknowledging limitations, and suggesting future work.

2. Experimental Set-up

2.1. Data Exploration

To get a better understanding of the dataset, we first explored the data to identify potential features for removal and any outlying examples that might skew our results.

The dataset contains three classes, each representing approximately one-third of the data. The number of examples (represented as n) in the dataset is 5000.

All input features are represented as floating-point numbers. The target variable, located in column 21, consists of integer values: 0, 1, and 2.

Initially, the dataset was split into training and test sets, with an 80/20 split. Subsequently, we separated the input features (represented as X) and the target variable (represented as y) for both the training and test sets.

Figure 1 shows the correlation matrix for the features in the training set, including both X and y. The visualisation shows some distinct areas of higher positive and negative correlations, especially surrounding the diagonal. Several features have a positive or negative correlation of more than 0.7, which suggests that dimensionality reduction techniques, such as Principal Component Analysis (PCA), could be effectively applied. It is also interesting that some of features are correlated to the target variable. Based solely on the correlation matrix, a classification algorithm would perform well on this dataset.

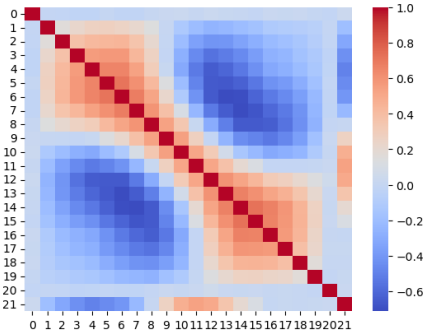

Figure 1. Correlation Matrix for Features in Waveform Training Set

2.2. Data Manipulation

To allow comparative analysis and simplify Euclidean distance calculations for kNN, we standardised the data using the z-score, defined as $z_i = \frac{x_i - \mu_i}{\sigma_i}$, where x_i represents the column i, μ_i it's corresponding mean and σ_i the standard deviation for i. It is important to standardise the test data

using the mean and standard deviation from the training data. This prevents data leakage, which is when test set data influences the training set, leading to unreliable estimates of generalised error.

With the standardised data, we can use box-plots to visually search for outliers. As illustrated in Figure 2, a box-plot shows the interquartile range (IQR) within the box, where 50% of the data lies. The whiskers, extending from the box, show the values within 1.5 times the IQR for both lower and upper quartiles. Therefore the values outside the whiskers are the potential outliers. In the waveform dataset, features 5 to 16 have fewer outliers, indicating more stability, while the other features have higher variability in their data, with more outliers above and below the mean. It is likely that these features with more variation have a larger impact in differentiating the examples.

It is difficult to determine if there are significant outliers in the data from this plot alone. An example may have an outlier value for one feature but lie in the typical range for all other features.

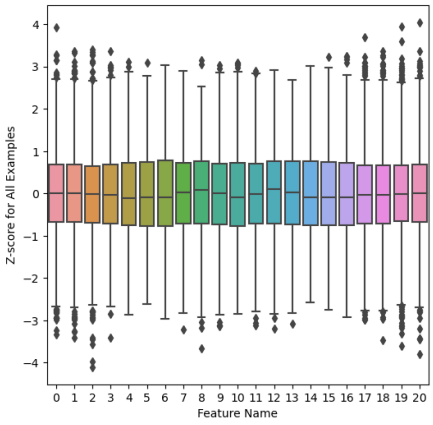

Figure 2. Correlation Matrix for Features in Waveform Training Set

2.3. Tuning k of k-NN by Cross Validation

To prevent the recomputation of distance when performing k-NN, first we computed the Euclidean distance matrix for the set of points in the train set with the function `ComputeDistanceMatrix`. Next, we used broadcasting to compute the pairwise differences and with it, we computed its Euclidean distances. The function returns a distance matrix of shape (n, n), when each entry $D[i, j]$ is the distance between point i and j.

We chose to perform 10 folds cross validation. `KFoldIndices` function takes an input train set and splits it into 10 folds. Initially, it creates 10 folds of data. In each fold, the data is split into 2 parts, one as the validation set and the other as the train set. The function returns the indices of the train and validation sets, and stores them to $\mathcal{F}$. An empty array $\mathcal{A}$ is used for storing the accuracy of each fold.

Now we have the folds $\mathcal{F}$, we did a loop for each fold. First, we computed the sub-matrix of distances for the validation set versus all datasets $\mathcal{D}_{\text{val}}$. An empty array $\mathcal{P}$ is used for storing the prediction, for each data point x in $\mathcal{X}_{\text{val}}$, we calculated the k-NN with `GetNeighbors`. This function returns the k nearest neighbors indices, store it to $\mathcal{N}$. Then, we ran `KnnPrediction` with $\mathcal{N}$ and $\mathcal{Y}_{\text{train}}$ as input. With $\mathcal{N}$ and $\mathcal{Y}_{\text{train}}$, we could retrieve the labels of the neighbors then perform majority voting to get the prediction, store it to y_{pred}. We added y_{pred} to array $\mathcal{P}$. With $\mathcal{Y}_{\text{train}}$ and $\mathcal{X}_{\text{val}}$, we got the actual labels for the validation data, store it to $\mathcal{Y}_{\text{val}}$. Finally, we performed `AccuracyMetric` with the actual list $\mathcal{Y}_{\text{val}}$ and predicted list $\mathcal{P}$. For each index i, if the predicted label matches the actual label, a counter is incremented. After all predictions are checked, the accuracy is calculated as $accuracy = \frac{N_{\text{correct}}}{N_{\text{total}}} \times 100$, then we added it to $\mathcal{A}$. So after performing k-folds Cross Validation, we got a list of accuracies for k-folds $\mathcal{A}$.

Pseudo-code for the implementation is Algorithm 1.

Next, we tuned k (number of neighbors) to find the optimal value, which maximize the accuracy of the model. Since we trained on the train data, $k = \sqrt{n} = \sqrt{4000} \approx 63$ is a good starting point. Thus, we looped over k from 1 to 100. For each k, we ran 10-folds Cross Validation to get the list of accuracies for 10 folds $\mathcal{A}$ and computed the mean accuracy. We tracked the best k that gives the highest accuracy, which is the optimal value that we are looking for.

2.4. Data Reduction Implementation

To increase the performance of the k-NN algorithm, especially in terms of computation time, it is necessary to perform data reduction on the dataset. The data reduction technique we implemented is executed in two steps as presented in Hart 1968 (Hart, 1968). The first step removes the outliers from the data, outlined in the algorithm 2. This algorithm uses the 1-NN classifier for outlier detection based on the fact that outliers are often misclassified by their single nearest neighbor.

The second step removes all points that are not necessary to classify new examples, outlined in the algorithm 3. This algorithm starts with an empty set, $STORAGE$, and iteratively adds misclassified points to it. In doing this, it keeps only the points that are required to define the decision boundaries between classes.

2.5. Speeding up the calculation

The Brute Force 1-NN logic is that, given a test point, it computes its distance to every point in the train set and selects the closest point as its nearest neighbor. We implemented a function that takes an input 3 datasets $\mathcal{X}_{\text{train}}$, $\mathcal{Y}_{\text{train}}$, $\mathcal{X}_{\text{test}}$ and returns the list of predicted labels $\mathcal{P}$ for the test set. First, we initialized an empty list $\mathcal{P}$ to store the predictions. For each test instance x_{test} in the test set X_{test}, we calculated the Euclidean distance from x_{test} to each train instance in train set X_{train}, store it in $\mathcal{D}$. The smallest value in $\mathcal{D}$ corresponds to the closest training point to the test instance x_{test}. Thus, we retrieved the index of nearest neighbor i_{nearest} by sorting $\mathcal{D}$ and taking the smallest value. With the training dataset $\mathcal{Y}_{\text{train}}$ and i_{nearest}, we got the label of the nearest neighbor y_{pred} and added it to $\mathcal{P}$. The pseudocode of Brute Force 1-NN is Algorithm 4.

The Triangle Inequality 1-NN algorithm computes the distance from each test point to every training point to find the nearest neighbor. It helps to reduce the number of distance computations by using the triangle inequality to eliminate points that cannot be the nearest neighbor. We implemented a function that takes an input 3 datasets $\mathcal{X}_{\text{train}}$, $\mathcal{Y}_{\text{train}}$, $\mathcal{X}_{\text{test}}$ and returns the list of predicted labels $\mathcal{P}$ for the test set. First, we initialized an empty list $\mathcal{P}$ to store the predictions. For each test instance x_{test} in the test set X_{test}, we initialized the minimum distance d_{min}, the label of the nearest neighbor and the indices of all candidate neighbors $\mathcal{C}$ (initially containing all training points). We did a loop through every candidate index, selected the first candidate i from $\mathcal{C}$. Then, we computed the Euclidean distance $d_{\text{candidate}}$ between the current test instance x_{test} and the i-th train instance $x_{\text{train}}[i]$. The sphere bounds are defined by a lower bound and lower bound. Then we filtered candidates using the Triangle Inequality. To do this, first we computed the distance between the current candidate and the remaining candidate using the distance matrix $\mathcal{D}$. Then, we just kept the candidates that its distances to the remaining candidate is larger than the lower bound of the sphere and smaller than the upper bound of the sphere. And if $d_{\text{candidate}}$ is smaller than d_{min}, we will update the value of d_{min} and set the label of the nearest neighbor to the label of $x_{\text{train}}[i]$. After all candidates are processed, we appended the label of the nearest neighbor to $\mathcal{P}$. Finally, after processing all test points, we returned the list $\mathcal{P}$. The pseudocode of Triangle Inequality 1-NN is Algorithm 5.

2.6. Sampling Methods

Sampling is commonly used in imbalanced datasets to ensure all classes have an equal representation of examples (Mohammed et al., 2020). The waveform dataset is already balanced, with each class representing approximately one-third of the data, so we are implementing sampling methods to artificially create imbalance and investigate the effects on k-NN's performance. We will evaluate using the best k found in Section 2.3.

We implemented two sampling methods: undersampling and oversampling. Undersampling a dataset takes the majority class and randomly removes examples until the dataset is balanced. We tested k-NN using a training and validation set with various levels of imbalanced datasets. We also compared the effect of undersampling each class separately.

Oversampling involves randomly duplicating examples from the minority class. Each example in the minority class can be duplicated more than once, leading to a high risk of overfitting. We performed the same tests on the oversampled datasets as on the undersampled ones.

Another sampling method that will not be covered in this report is called SMOTE. This is based on oversampling and aims to reduce the risk of overfitting by generating artifical data points.

The results of these experiments are presented in Section 3.4, where we compare the performance of k-NN across the various undersampled and oversampled datasets using metrics such as accuracy and F1-score.

3. Result Analysis

3.1. Tuning k of k-NN by Cross Validation

We plot the accuracy of the implemented k-NN classifier in figure 3. The accuracy steadily increases as k rises from $k = 1$ to larger values, reaching a peak when $k = 86$, achieving 85.67%. This result suggests that the implementation performed as expected, since it is very close to the theoretical Bayes optimal rate (86%). For small k ($k < 10$), the accuracy fluctuates drastically, which shows that the classifier has high variance at this point, a sign of overfitting. This result can be expected since for small k, the classifier closely follows the train data, leading to poor generalization. The curve becomes stable and slightly fluctuates after $k = 30$. This implies that the model's variance is decreasing, and it is reaching the limit because the noise in the dataset prevents it from reaching the optimal Bayes rate.

3.2. Data Reduction

Here, the data reduction algorithms outlined in 2.4 are implemented. The first step, removing outliers, reduced the dataset size from 4000 to 3176, identifying 824 outliers, which represents approximately 20% of the original data. This aligns with the box-plots from Section 2.2 which showed many features had a large amount of outliers.

The second step of the algorithm further reduced the dataset by 671 examples, making the final reduced size 2511. The data reduction algorithm has therefore removed approxi-

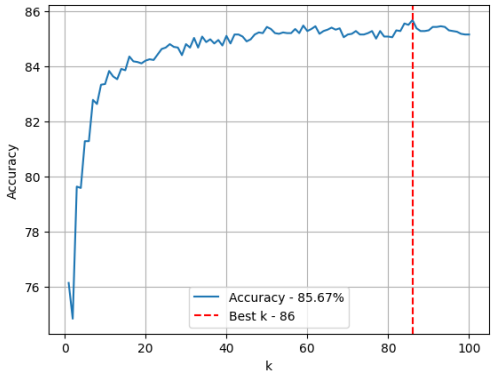

Figure 3. Accuracy vs Number of neighbors (k) for k-NN

DATA SET	1-NN	K-NN
ORIGINAL TRAINING SET	75.1%	84.3%
REDUCED TRAINING SET	75.7%	83.7%

Table 1. Classification accuracies for 1-NN and k-NN before and after data reduction.

mately 37% of the training set. To evaluate the impact of this reduction on the performance of the k-NN classifier, we compared the accuracy of 1-NN and k-NN using the best k from Section 2.3 on the test set, using both the original and reduced training set.

Table 3.2 shows the results of this comparison. There has been no significant loss of information that leads to a decrease in accuracy. The slight improvement in the accuracy of 1-NN on the reduced dataset could be explained by the lack of outliers in the reduced dataset, which 1-NN can be particularly affected by. Meanwhile the slight decrease in accuracy for k-NN could be due to the best k being tuned for the original dataset. It is possible that tuning a new k for the reduced dataset would improve the test accuracy.

3.3. Speeding Up the Calculation

After performing Brute Force 1-NN and Triangle Inequality 1-NN, we got the computation time for Brute Force around 0.1 seconds and for Triangle Inequality it is around 7 seconds, which is much higher than Brute Force. The computation time for each iteration is shown in figure 4, which supports this result. This is because, at each step, Triangle Inequality filters the list of candidate points by checking if they lie within the sphere bounds or not. This requires logical operations and array indexing on the set of candidate points, which takes more time than a simple computation of distances, as in Brute Force. Another reason is that, when we printed out the filtered candidates (those lie inside the upper bound and lower bound), actually Triangle Inequality does not filter too many candidates. Thus, the algorithm reverts to Brute Force performance but with additional computation.

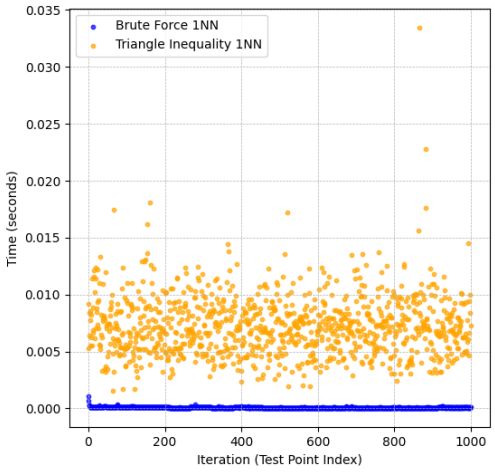

Figure 4. Computation time for each test point (iteration)

3.4. Sampling Methods

3.4.1. SAMPLING WITH BEST k

We implemented the sampling methods outlined in Section 2.6 to investigate their impact on k-NN classifier performance. The dataset was initially split into a training set (3000 examples), a validation set (1000 examples), and a test set (1000 examples). Before applying k-NN, we standardized the test set using the mean and standard deviation calculated from the training set.

To test undersampling, we tested the accuracy on varying percentages of removal, from 0% (no removal) up to 100% (complete removal of the class). The results are shown in Figure 5. The removal of class 2 from the dataset has the biggest impact on the test accuracy, which could suggest that there is a slightly higher representation of class 2 in the test set. Meanwhile removing class 1 had the least effect on the k-NN accuracy, suggesting that it may have a lower representation in the test set.

We then tested the effects of oversampling on classifier accuracy. We increased the number of examples in each class by varying percentages, from 0% (no oversampling) up to 100% (twice as many examples of class). The percentage corresponds to the increase in the number of examples relative to the original count of that class in the training set. The results are shown in Figure 6. These results corroborate with those of undersampling, with the oversampling of class 1 leading to a decrease in accuracy. Notably, oversampling

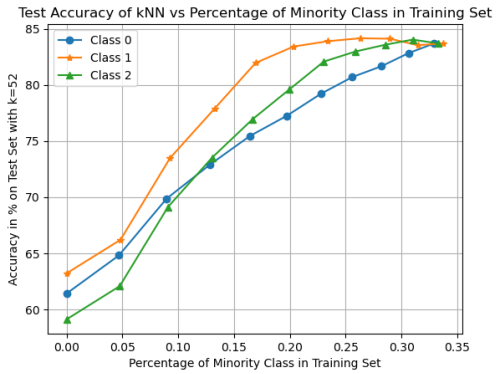

Figure 5. Accuracy on Test Set vs Percentage of Class in Training Set

both class 0 and class 2 temporarily increases the accuracy on the test set. This could suggest that both class 0 and class 2 are have a higher representation in the test set than class 1.

To further investigate the class distribution in the test set, we visualized the representation of each class, as shown in Figure 7.

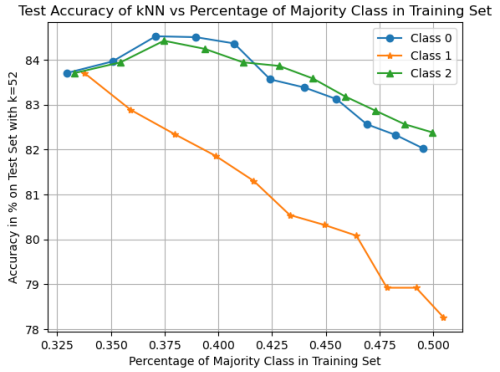

Figure 6. Accuracy on Test Set vs Percentage of Class in Training Set

The test set distribution confirms our hypothesis that class 2 has the lowest representation. It also reveals that class 0 and and class 1 have roughly an equal number of examples in the test set, which is consistent with the oversampling plot.

Ideally, the test set distribution should mirror the class distribution of the original dataset. A method to implement this is called stratified sampling, which ensures all splits maintain the same distribution. However, this has not been implemented in the project.

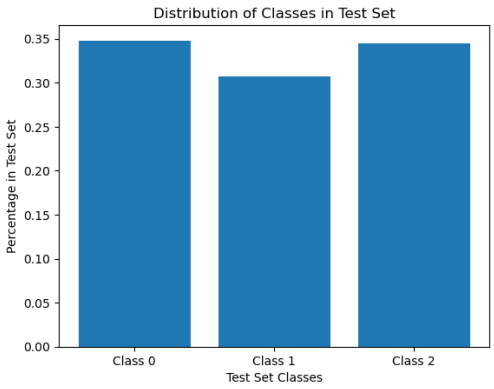

Figure 7. Representation of each class in the test set

3.4.2. Sampling using F-Measure to Tune k

In this section, we use the F-measure to tune the best k for each sampling rate. First, we must apply each sampling rate from 0% to 100% in increments of 10% to each class individually, creating a range of different datasets. Next, we can tune k by testing a range of k values from 1 to 60 against the F-measure on the validation set. The k value with the highest F-measure is selected as the best performer.

Figure 8 shows the results of this process for undersampling. Unexpectedly, using the F-measure to tune k does not improve the accuracy of the classifier on the test set. The highest test accuracy achieved is approximately 81%, and the lowest is worse than the results obtained in the initial undersampling (Figure 5).

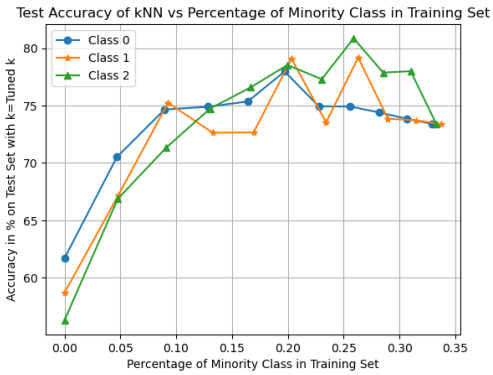

Figure 8. Representation of each class in the test set

Figure 9 shows the graph for the oversampling method. The results are consistent with the undersampling graph: using the F-measure to tune k does not improve the classification accuracy on the test set and generally decreases it.

While the F-measure is a useful metric for imbalanced

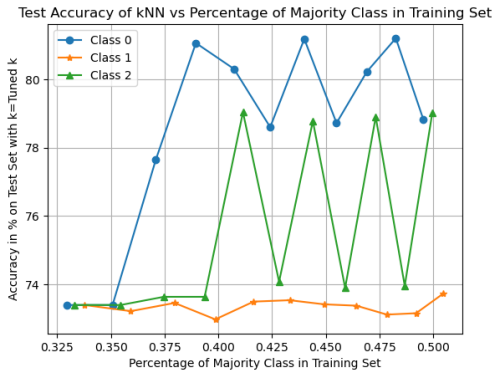

Figure 9. Representation of each class in the test set

datasets, it does not seem to translate well to our problem. It is possible that it is not suitable when the imbalance is artificially introduced, as in our case. The standard classification accuracy might be a more reliable metric for evaluating model performance. By removing or adding points, we are likely not reflecting the true underlying distribution, leading to skewed results using the F-measure.

4. Conclusion

This project investigated the application of k-NN and various optimisation methods on the waveform dataset. Tuning the best value of k results in a generalised accuracy of 84.3%, which is quite close to the optimal Bayesian classification accuracy of 86%, suggesting k-NN is a good algorithm for this classification problem.

As expected, k-fold cross-validation proved to be an effective method for tuning the best value of k. Data reduction techniques were also successfully applied, removing outliers and redundant points, which increased the speed of k-NN predictions while maintaining a similar classification accuracy to that achieved on the original training set. Outlier removal also likely contributed to the improved accuracy of 1-NN on the reduced dataset.

Using the triangle inequality to speed up k-NN proved to be unsuccessful. This is likely because our implementation used nested for loops, which is much less efficient than NumPy's highly optimised vectorised operations used in the brute force 1-NN. Future work could investigate an implementation of the triangle inequality using NumPy's vectorised arrays.

Implementing sampling methods showed an imbalanced representation of the data in the test set, with one class having fewer examples than the others. Undersampling led to the expected result of the k-NN classification accuracy decreasing as the percentage of each class in the training set tends to 0. However, oversampling showed a temporary increase in accuracy when examples from the two over-represented classes were duplicated. This is likely attributed to the fact there is more examples in the training set for k-NN to match with. Using the F-measure to tune the best value of k for each sampling rate did not yield the results expected, as it further reduced the test accuracy. Our conclusion is that using F-measure for artificially imbalanced datasets is not as accurate a metric as using the classification accuracy, This is possible because sampling the data alters the underlying data distribution, which deems the F-measure ineffective.

Due to time limitations, we were unable to implement other techniques, such as PCA for dimensionality reduction. Given the correlation matrix in 1, PCA could likely retain a lot of the variance in the data with a reduced number of dimensions, potentially improving both efficiency and accuracy of k-NN.

Future work could also include implementing more sampling methods such as SMOTE, and investigating more into the unexpected results of the triangle inequality and F-measure tuning.

Overall, the project was a valuable insight into implementing the k-NN algorithm and applying various optimisations to it. The results show that a simple algorithm like k-NN can achieve close to the optimal Bayesian accuracy on datasets such as the waveform one. It is also important to consider and evaluate various optimisation techniques to improve the efficiency and accuracy of k-NN.

5. Contributions

5.1. Patrick Barry

- Abstract, Introduction
- Data Exploration, Data Manipulation
- Task 2: Data Reduction
- Task 4: Sampling Methods
- Conclusion

5.2. Buu Dinh Ha

- Task 1: Tuning k of k-NN by Cross Validation
- Task 3: Speeding up the calculation

References

Hart, P. The condensed nearest neighbor rule (corresp.). *IEEE transactions on information theory*, 14(3):515–516, 1968.

Mohammed, R., Rawashdeh, J., and Abdullah, M. Machine learning with oversampling and undersampling techniques: Overview study and experimental results. In *2020 11th International Conference on Information and Communication Systems (ICICS)*, pp. 243–248, 2020. doi: 10.1109/ICICS49469.2020.239556.

Algorithm 1 k-folds Cross Validation

Input: Dataset $\mathcal{X}_{\text{train}}$, $\mathcal{Y}_{\text{train}}$, Distance Matrix D, Number of Folds k_{folds}, Number of Neighbors k
Output: List of accuracies for k folds $\mathcal{A}$
$\mathcal{F} \leftarrow \texttt{KFoldIndices}(\mathcal{X}_{\text{train}}, k_{\text{folds}})$
$\mathcal{A} \leftarrow []$
for each fold $(\mathcal{X}_{\text{train}}, \mathcal{X}_{\text{val}})$ **in** $\mathcal{F}$ **do**
  $D_{\text{val}} \leftarrow \texttt{ExtractDistances}(D, \mathcal{X}_{\text{val}})$
  $\mathcal{P} \leftarrow []$
  for each x **in** $\mathcal{X}_{\text{val}}$ **do**
    $\mathcal{N} \leftarrow \texttt{GetNeighbors}(D_{\text{val}}, \mathcal{X}_{\text{train}}, k)$
    $y_{\text{pred}} \leftarrow \texttt{KnnPrediction}(\mathcal{N}, \mathcal{Y}_{\text{train}})$
    Append y_{pred} to $\mathcal{P}$
  end for
  $\mathcal{Y}_{\text{val}} \leftarrow \texttt{GetActualLabels}(\mathcal{Y}_{\text{train}}, \mathcal{X}_{\text{val}})$
  $a \leftarrow \texttt{AccuracyMetric}(\mathcal{Y}_{\text{val}}, \mathcal{P})$
  Append a to $\mathcal{A}$
end for
Return: $\mathcal{A}$

Algorithm 2 Remove Outliers Algorithm

Input: X_{train}
Output: $X_{train\_cleaned}$
Randomly split X_{train} into two subsets S_1 and S_2
while S_1 and S_2 not stabilised **do**
  Classify S_1 with S_2 using 1-NN
  Remove from S_1 the misclassified instances
  Classify S_2 with updated S_1 using 1-NN
  Remove from S_2 the misclassified instances
end while
$X_{train\_cleaned} = S_1 \cup S_2$

A. Appendices

Algorithm 3 Remove Redundant Points Algorithm

Input: X_{train}
Output: $STORAGE$
$STORAGE \leftarrow \emptyset; BIN \leftarrow \emptyset$
Put a random example from X_{train} in $STORAGE$
while $STORAGE$ not stabilised **do**
  for all x_i in X_{train} **do**
    y_{pred} = classify x_i using $STORAGE$ with 1-NN
    if y_{pred} is correct **then**
      Add x_i to BIN
    else
      Add x_i to $STORAGE$
    end if
  end for
end while

Algorithm 4 Brute Force 1-NN

Input: Dataset $\mathcal{X}_{\text{train}}$, $\mathcal{Y}_{\text{train}}$, $\mathcal{X}_{\text{test}}$
Output: List of predicted labels $\mathcal{P}$ for the test set
$\mathcal{P} \leftarrow []$
for x_{test} **in** $\mathcal{X}_{\text{test}}$ **do**
  $\mathcal{D} \leftarrow \texttt{ComputeDistances}(\mathcal{X}_{\text{train}}, x_{\text{test}})$
  $i_{\text{nearest}} \leftarrow \texttt{FindNearestNeighbor}(\mathcal{D})$
  $y_{\text{pred}} \leftarrow \texttt{GetLabel}(\mathcal{Y}_{\text{train}}, i_{\text{nearest}})$
  Append y_{pred} to $\mathcal{P}$
end for
Return: $\mathcal{P}$

Algorithm 5 Triangle Inequality 1-NN

Input: Dataset $\mathcal{X}_{\text{train}}$, $\mathcal{Y}_{\text{train}}$, $\mathcal{X}_{\text{test}}$, distance matrix $\mathcal{D}$
Output: List of predicted labels $\mathcal{P}$ for the test set
$\mathcal{P} \leftarrow []$
for x_{test} **in** $\mathcal{X}_{\text{test}}$ **do**
  $d_{\text{min}} \leftarrow \infty$
  nearest_neighbor_label $\leftarrow$ None
  $\mathcal{C} \leftarrow \{0, 1, \ldots, n-1\}$
  while $\mathcal{C} \neq \emptyset$ **do**
    Select and remove the first index i from $\mathcal{C}$
    $d_{\text{candidate}} \leftarrow \|x_{\text{test}} - x_{\text{train}}[i]\|_2$
    lower_bound $\leftarrow d_{\text{candidate}} - d_{\text{min}}$
    upper_bound $\leftarrow d_{\text{candidate}} + d_{\text{min}}$
    $d_{\text{candidate\_to\_others}} \leftarrow \mathcal{D}[i, \mathcal{C}]$
    $\mathcal{C} \leftarrow \{j \in \mathcal{C} \mid \text{lower\_bound} \leq d_{\text{candidate\_to\_others}}[j] \leq \text{upper\_bound}\}$
    if $d_{\text{candidate}} < d_{\text{min}}$ **then**
      $d_{\text{min}} \leftarrow d_{\text{candidate}}$
      nearest_neighbor_label $\leftarrow \mathcal{Y}_{\text{train}}[i]$
    end if
  end while
  Append nearest_neighbor_label to $\mathcal{P}$
end for
Return: $\mathcal{P}$