LAZY LEARNING

INTRODUCTION TO MACHINE LEARNING
UNIVERSITAT DE BARCELONA

MASTER IN ARTIFICIAL INTELLIGENCE

Members:
Joël Haupt
Hajar Lachheb
Saurabh Nagarkar
Miquel Sugrañes

Professor:
Maria Salamó

Course:
2022/2023

Contents

1. Introduction

The aim of this work is to study instance reduction performed on data used for a Lazy Learning classification approach and the performance obtained before and after the removal of examples.

More concretely, we will study the k-Nearest Neighbors as a Lazy Learning algorithm, that is a model that doesn't perform many steps during training time but deferes the majority of them to the consultation time, on the data already used in previous works.

For this work, several k-Nearest Neighbors models will be created and tested, in order to find the one that performs better on each dataset used in a total of 3. Later, the best model will be used to predict class labels for test data using a reduced number of instances as training data.

In this document we first introduce the practical case in section 2, and we define the different metrics used to build our k-NN approaches. Then, in section 3 we discuss the results obtained for each combination in each dataset, and how we have chosen the best k-NN model. In section 4 we introduce and explain the several instance-reduction techniques used. Finally, a comparison between the use of the whole dataset and the reduced subset is done is section 5.

For the analysis different calculations have been performed, but basically we studied the accuracy of the model and the run-time required to perform classification.

We would like to emphasize that the computational cost of this work has resulted in high run-times, making it difficult to finish all the required experiments. In order to be able to obtain results in the time given, we have performed instance reduction by keeping a defined number of the first instances of the biggest datasets. Moreover, the fact that different and limited computational resources have been used by the members of the team has made it difficult to compare results especially in terms of run times values.

2. Metrics implemented for the k-NN Algorithm

In this work we were asked, as explained in the previous introduction, to perform several experiments related to the k-Nearest Neighbors model for classification given a set of datasets and then perform different instance reduction techniques on this given data in order to compare the performance of the model when reducing the amount of samples used.

Before implementing our k-NN approach, we had to perform some previous steps in order to preprocess our data and normalize it, so it was suitable for the algorithm. More concretely, we had to load our input data and normalize the values for each instance in a [0, 1] range. This rescaling was performed with the min-max normalization technique. Moreover, the classes for each instance were stored in a different variable, so our training data was separated into features (X) and class labels (y) indexed in the same way. By doing this, we could keep the original label values for each class and only rescale the features' values. We must also say that categorical variables have been transformed into numerical ones in order to correctly preprocess all data.

In order to compare and contrast the performance of different k-Nearest Neighbors algorithms we were asked to compare different metrics to define different k-NN combinations and evaluate their performance on three different datasets. The different metrics to be evaluated for this work are:

- Distance metrics (Minkowski, Cosine and Hassanat)
- Weighting metrics (Majority class, Sheppard's work, Inverse distance)
- Voting metrics (Equal weights, ReliefF, Information Gain)
- *k* values (1, 3, 5, 7)

For each one of the combinations, we computed the accuracy of the model to predict class labels on test data, as a measure of how well it performed. Furthermore, we also studied the run time for each combination to both fit the training data and predict classes for test data, as a measure of computational efficiency when combined with the accuracy obtained. Related to the accuracy, we also stored for each combination of metrics the number of correctly and incorrectly predicted instances.

All these calculations were saved in a text file making use of python libraries such as *numpy*, in order to be able to recover afterwards to make some analysis.

We also have to keep in mind that each combination of k-NN metrics was used to perform classification on each of the 10 folds of each dataset separately, so at the end we had for the same combination 10 different results for one dataset. In order to have a common measure for the whole 10 folds for each combination, we calculated the mean of both the accuracy and the run time.

These final mean values were used to compare the different k-Nearest Neighbors approaches and choose, for each one of the datasets, the combination that better fitted the data.

The several metrics we have used in order to build our different k-NN models can be found in the following subsections.

a. Distance Functions

For the distance calculation between data points, that is instances from our data, we made use of three different functions: the Minkowski Distance with order = 2 (also known as Euclidean distance), the Cosine Distance and finally the Hassanat Distance.

Minkowski Distance

Firstly, in our work we used the Minkowski distance. In fact, the Minkowski Distance is a metric that can be used to measure the distance between two points in a Euclidean space. It is a generalization of the Euclidean distance. The Minkowski distance is defined as follows:

Given two points x and y in an N-dimensional space, the Minkowski distance between x and y is:

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Figure 1 - Minkowski Distance

where x_i and y_i are the i-th components of the points x and y, respectively, and p is a positive real number. When p = 2, which is our case, the Minkowski distance becomes the Euclidean distance. When p = 1, the Minkowski distance is known as the Manhattan distance, which is the sum of the absolute differences of the coordinates.

The Minkowski distance is a metric because it satisfies the following three properties:

1. Non-negativity: d(x, y) ≥ 0 for all x and y, and d(x, y) = 0 if and only if x = y.
2. Symmetry: d(x, y) = d(y, x) for all x and y.
3. Traingle inequality: d(x, z) ≤ d(x, y) + d(y, z) for all z, y and z.

Finally, the Minkowski distance is often used in machine learning and data mining algorithms, such as k-nearest neighbors and clustering, to measure the similarity between data points.

i. Cosine Distance

To define the Cosine Distance, we need to first define the Cosine Similarity, since both of the notions are highly related. This to say, Cosine Similarity is a metric used to measure how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space.

Therefore, we can add the conclusion that similarity decreases when the cosine distance between two vectors increases. This leads us to define the Cosine Distance.

Cosine_Distance = 1 - Cosine_Similarity & Cosine Similarity = $\cos(\theta)$

So calculating the Cosine Distance depends on calculating the Cosine Similarity, that we can calculate using the following formula:

$$\text{cosine similarity} = S_C(A,B) := \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Figure 2 - Cosine Distance

This distance is really important. Since it will help us find the optimal number of neighbors. We can also use it in the KNN model that is fitted and can be evaluated against different classification machine learning algorithms and the KNN classifier alone that is fitted with cosine similarity as a metric can be used to evaluate various performance parameters like the accuracy score, AUC score, and the classification report can also be obtained to evaluate other parameters like precision and recall.

Hassanat Distance

Hassanat is a distance that we personally chose since as has been seen in the literature, it outperforms other distances results, and so we wanted to incorporate it in our work. After doing a search through the given literature on the evaluation of distance metrics applied for KNN, we found out that it performed the best out of all distance metrics. (Abu Alfeilat et. al., 2019)

In fact, Hassanat Distance was introduced by Hassanat in 2014, in Dimensionality invariant similarity measure. Journal of American Science, 10.

$$HasD(x,y) = \sum_{i=1}^{n} D(x_i, y_i)$$

Figure 3 - Hassanat distance

Hassanat distance is bounded by [0,1]. It reaches 1 when the maximum value approaches infinity assuming the minimum is finite, or when the minimum value approaches minus infinity assuming the maximum is finite. We can see more about the mathematical borders that defines the Hassanat Distance in the following limits.

$$\lim_{\max(A_i,B_i)\to\infty} D(A_i,B_i) = \lim_{\max(A_i,B_i)\to-\infty} D(A_i,B_i) = 1,$$

Figure 4 - Limits for the Hassanat distance function

By satisfying all the metric properties this distance was proved to be a metric by Hassanat (2014). In this metric no matter what the difference between two values is, the distance will be in the range of 0 to 1. so the maximum distance approaches the dimension of the tested vectors, therefore the increase in dimensions increases the distance linearly in the worst case.

b. Voting Methods

After defining the different distance functions, we will now define the different voting methods. There are a variety of ways in which k-Nearest Neighbors can be used to determine the class of the query q. In our work, we will consider these three following policies :

i. Majority Class Voting

The majority class voting technique basically defines and assigns the class label to an instance based on the most frequent class label of its neighbors. In some cases, it can happen that a tie occurs, and there is no label defined as more frequent as the rest. For this reason, we also introduced in our work a method to break ties, which will be explained in further details in the following paragraphs.

Let's put in mind that a tie can occur when two or more neighbors from an unclassified observation have the same label, thereby making it difficult to predict a concrete class. With our solution, we need to emphasize the fact that we are working with distance. This means that solving ties and choosing the right label (to solve the tie) will be following how high its distance can be or can reach.

In our approach, we are defining labels and sorting their respective mean distance in a decreasing order so as to get, by the end of the function, the highest mean distance related to the respective label. In fact, sorting the mean distances helps us a lot to avoid any more ties and extract only one label, which is in our case the label whose mean distance is the highest.

ii. Inverse Distance Weighted votes

For our second voting method, we are actually assigning more weight to the nearer neighbors in deciding the class of q. In fact, we will be taking a neighbor's vote to be the inverse of its distance to q. The voting mathematical formula is defined as follows:

$$Vote(y_j) = \sum_{c=1}^{k} \frac{1}{d(q, x_c)^p} \, 1(y_j, y_c)$$

Figure 5 - Vote weight for inverse distance

The vote assigned to class yi by neighbor yc is 1 divided by the distance to that neighbor, for example 1(yi,yc) returns 1 if the class labels match and 0 otherwise. In the equation, p would normally be 1 but values greater than 1 can be used to further reduce the influence of more distant neighbors.

To explain more in depth how we developed the algorithm for our work. We consider that the inverse_distance function takes two arguments: k_labels: a list of labels (integers) and k_distances: a list of distances (floats). The function, then, does the following:

It converts the list of distances to a numpy array, and divides 1 by each element which effectively inverts the distances. Then, it calls np.argmax on the result of np.bincount, which takes the list of labels and the list of inverted distances as inputs. np.bincount counts the number of occurrences of each label, and assigns a weight to each count based on the corresponding inverted distance.

Actually, np.argmax returns the label with the highest weighted count. And the inverse_distance function returns the label that appears the most in the list of labels, but with the weights of each label adjusted based on the corresponding distance. The label with the highest weighted count is the one that is returned.

iii. Sheppard's work

Sheppard's work approach is very similar to the previous voting metric. As explained before, we assign weights to each one of the neighbors depending on the distance between them and the instance we have to predict the label for. In the previous case an inverted distance approach was used, and similarly to that, in Sheppard's work an exponential function is implemented. The idea remains the same, and the vote for each neighbor is defined as the exponential function of the negative distance of each neighbor to the data point to be predicted. The following formula defines this voting metric:

$$Vote(y_j) = \sum_{c=1}^{k} e^{-d(q,x_c)}\, 1(y_j, y_c)$$

Figure 6 - Vote weight for Sheppard's work

Note the use of the negative distance in the exponential function. This is used because like this we can obtain a behavior where the higher the distance is, the lower the importance of the vote will be. So, by this, we can give more relevance to the nearest neighbors than to those that are further from the new data point.

c. Feature Weighting

Feature weighting seeks to estimate the relative importance of each feature (with respect to the classification task), and assign it a corresponding weight (value). When properly weighted, an important feature would receive a larger weight than less important or irrelevant features, which are weighted with a value of 0. In our work, we make use of three major feature weighting methods: equal weighting, information gain and RelieF.

i. Equal Weighting

For the equal weighting method, and as the name indicates, all the features are equally important for the algorithm. By this, we want to give the same value for each feature weight, and this is done by assigning a value of 1 to all instances. As can be understood, by doing so all distances have the same importance, regardless of how close they are to the unseen data point.

ii. Information Gain

In feature weighting, information gain is a measure of how much a feature contributes to the classification of a dataset. It is calculated by comparing the entropy of the target variable before and after splitting the data on a particular feature. Entropy is a measure of the impurity of a set of examples. If the examples are completely pure (i.e., all examples belong to the same class), the entropy is zero. On the other hand, if the examples are equally divided among different classes, the entropy is at its maximum. Information gain is calculated as the difference between the entropy before the split and the weighted average entropy after the split, where the weight is the proportion of examples that fall into each branch.

In a simpler way, Information Gain or also called Mutual Information is a method that measures the amount of information one can obtain from a random variable given another.

For our implementation, we have made use of an already-implemented approach, belonging to the *sklearn* python library, called Mutual Information Classification.

iii. ReliefF

To start with, Relief is a family of machine learning algorithms that uses nearest-neighbors to select features whose association with an outcome may be due to epistasis or statistical interactions with other features in high-dimensional data.

In fact, there is a big difference between Relief and ReliefF. The original Relief can deal with nominal and numerical attributes, and basically calculates a feature score for each feature which can then be applied to rank and select top scoring features for feature selection. However, it cannot deal with incomplete data and is limited to two-class problems. Its extension, which solves these and other problems, is called ReliefF.

In any case, the algorithm that defines Relief is closely related to the one defining ReliefF, and as it is quite simpler, we will briefly introduce it as follows:

Take a data set with n instances of p features, belonging to two known classes. Within the data set, each feature should be scaled to the interval [0 1] (binary data should remain as 0 and 1). The algorithm will be repeated m times. Start with a p-long weight vector (W) of zeros.

At each iteration, take the feature vector (X) belonging to one random instance, and the feature vectors of the instance closest to X (by Euclidean distance) from each class. The closest same-class instance is called 'near-hit', and the closest different-class instance is called 'near-miss'. Update the weight vector such that

$$W_i = W_i - (x_i - \text{nearHit}_i)^2 + (x_i - \text{nearMiss}_i)^2$$

Thus, the weight of any given feature decreases if it differs from that feature in nearby instances of the same class more than nearby instances of the other class and increases in the reverse case.

3. Analysis of the best k-NN Evaluation (Accuracy and Efficiency)

To evaluate the performance of our different methods in combination, we will be using, as explained before, both accuracy and efficiency. These evaluation results will help us choose the best KNN Algorithm and what combination is used to get the best KNN Algorithm.

a. Finding the best Combination

After running several tests and storing the results for each method and for each dataset, we performed a search through results taking **accuracy** and **efficiency** into consideration. In order to achieve this, the results for the mean accuracy as well as the corresponding mean runtime were normalized. For each value in the results the normalized runtime was subtracted from the normalized accuracy. To account for their importance we gave the accuracy a slightly higher weighting (60%) to prioritize the accuracy over the runtime (40%).

We chose this approach because by doing so we first consider both calculations to choose the best model, so we give importance to both the accuracy and the computational time required, and second we weigh more the accuracy, so getting good results have higher importance than the time required, since none of them when executed only one time lasted long enough to be discarded. It must be taken into account that running tests in different environments may lead to different results, but all in all, the behavior is the same for all the cases if they are run in the same machine, so as it is the difference defined before.

The results for 10 folds are averaged and plotted below, showing per each dataset both the accuracies received for each combination as well as the run time values.

In Figure 7 the results on the vowel dataset show that with an accuracy of 96.97% and a runtime of 0.016 seconds the best model combination for this dataset applied Minkowski distance, 1-Nearest Neighbour, Sheppard's work as a voting method and ReliefF for feature weighting.

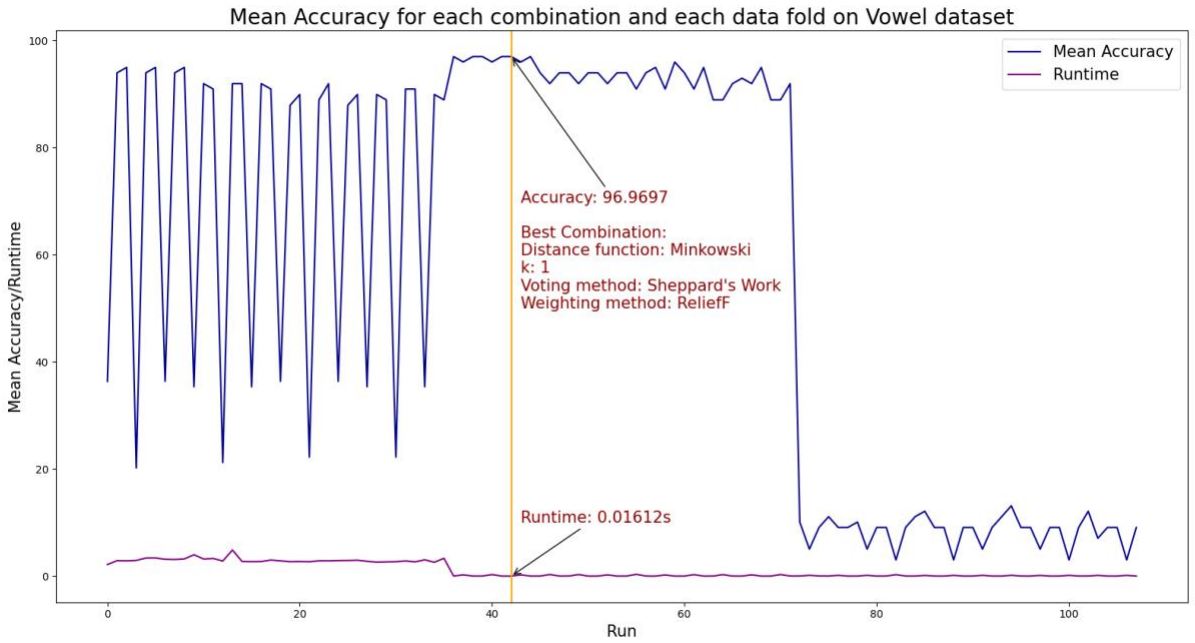

Figure 7 - Results for the k-NN approaches for the Vowel dataset

Figure 8 shows the results of the test results on the Adult dataset. Due to computational limitations we only iterated over the first 1000 instances of this dataset. The combination of methods with highest accuracy/efficiency trade off is Cosine Distance, K=3 for K-Nearest Neighbours, the inverse distance as the voting method and equal weights for all features, which achieves a 77.1% accuracy and ran for 0.436 seconds.

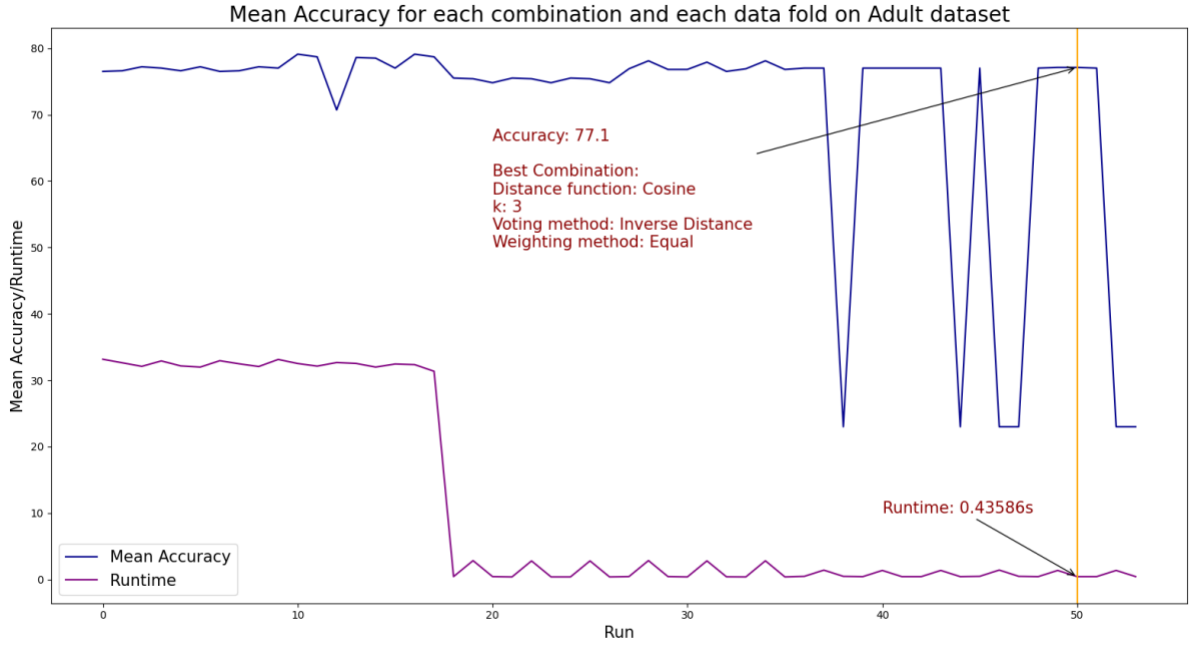

Figure 8 - Results for the k-NN approaches for the Adult dataset

Figure 9 shows the mean results for the Pen-based dataset. With an accuracy of 99.46% and a runtime of 5.32 seconds the classifier using Minkowski distance, k=1, Sheppard's work for voting and equal weights was the best tradeoff between accuracy and runtime.

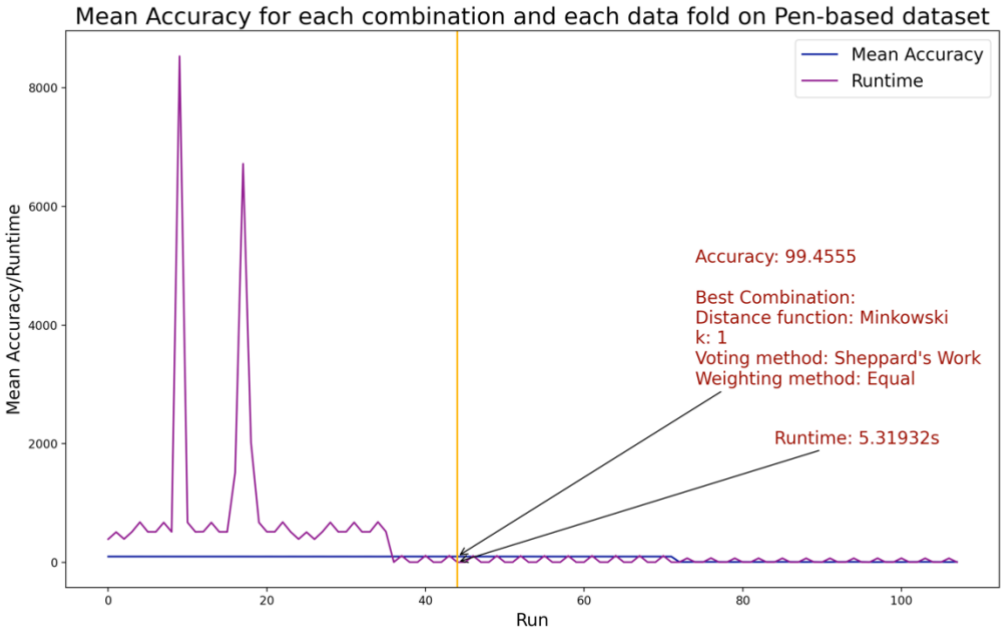

Figure 9 - Results for the k-NN approaches for the Pen-based dataset

The classifier on the Vowel dataset achieved the highest accuracy and ran the fastest.

The Minkowski metric was chosen as the best distance function in two of the combinations, as well as k=1 was applied in two combinations. Inverse Distance was the most popular voting method and keeping the features equally weighted was applied in two of the three best classifiers.

It turns out that none of the models with Hassanat distance were kept. This is probably because its runtime took much longer with the additional computation of minimum and maximum which led to a punishment for a long runtime. This is also indicated by the abrupt decrease of the runtime after the first third of runs, where the Hassanat distance is used.

4. Instance reduction techniques

The aim of this section is to apply to our original datasets several instance reduction techniques in order to reduce the number of examples they contain and so, reduce their size. The idea in which this approach is based is that some instances may contain irrelevant information, or may be adding noise to the data. By removing these examples, we can keep a good performance for the trained model but reduce the computational complexity and cost, since a reduction in size will also mean a reduction in the time and the resources required to run the algorithms.

For our work, we have implemented the Reduced Nearest Neighbor (RNN), the Repeated Edited Nearest Neighbor (RENN) and the Decremental Reduction Optimization Procedure 3 (DROP3) algorithm.

a. Reduced Nearest Neighbor

The first technique (RNN) is a filter approach based on condensation which has been implemented as a first step using the Condensed Nearest Neighbor Rule (incremental) and a second one applying the RNN rule (decremental).

The basic idea of the CNN case is the following, being T_{NN} the whole dataset and T_{CNN} a reduced subside obtained after the algorithm's procedure:

1. Copy the first instance from T_{NN} to T_{CNN}

   Until termination do:

2. Use T_{CNN} to classify all instances from T_{NN}

Where termination is accomplished when all examples in T_{NN} are correctly classified by T_{CNN}.

Then, RNN is applied in the inverted way, as follows:

1. Copy T_{CNN} to T_{RNN} (a new subset of samples)

   Until termination do:

2. Remove one instance from T_{RNN}
3. Use T_{RNN} to classify all instances from T_{CNN}
   a. If there is any incorrectly classified instance return the removed one

Where termination is accomplished when all examples in T_{RNN} has been removed once and possibly replaced.

So we can see that the first approach increases the subset until obtaining an optimal one, while the second approach does it the other way around: it decreases the subset until obtaining the optimal one.

b. Repeated Edited Nearest Neighbor

The second approach (RENN) is a filter approach based on edition with a decremental direction of search. It is based on the ENN (Edited Nearest Neighbor) algorithm, and basically repeats its procedure for a given number of iterations. By doing so, the reduction of instances may increase depending on the case, obtaining better results in terms of storage as the algorithm advances. Even so, it has been seen that after a finite number of iterations or repetitions the set becomes immune to further eliminations.

c. Decremental Reduction Optimization Procedure 3

The third and final approach (DROP3) is the more consistent one as has been seen in the literature. It is a consistent approach that uses both ENN and DROP2 techniques. Firstly, it uses a noise-filtering rule based on ENN, and then uses the DROP2 algorithm to both sort instances by their distance to their nearest enemy, that is, the nearest neighbor labeled with a different class, and eliminate unnecessary instances. The DROP2 algorithm is also based on a previous version of DROP technique, the DROP1, which is the simplest one but at the same time easier to understand. The following pseudocode is an example of how the DROP1 behaves in order to reduce instances:

```
1.  DROP1(Training set T): Instance set S.
2.      Let S = T.
3.      For each instance P in S:
4.          Find P.N_{1..k+1}, the k + 1 nearest neighbors of P in S.
5.          Add P to each of its neighbors' lists of associates.
6.      For each instance P in S:
7.          Let with = # of associates of P classified correctly with P as a neighbor.
8.          Let without = # of associates of P classified correctly without P.
9.          If without ≥ with
10.             Remove P from S.
11.             For each associate A of P
12.                 Remove P from A's list of nearest neighbors.
13.                 Find a new nearest neighbor for A.
14.                 Add A to its new neighbor's list of associates.
15.             For each neighbor N of P
16.                 Remove P from N's lists of associates.
17.         Endif
18.     Return S.
```

Figure 10 - DROP1 pseudocode

In order to analyze the performance of each one of the reduction techniques we computed a classification with our k-Nearest Neighbors algorithm using the best combination of metrics obtained from the previous experiments, and fitting the reduced data into it. By doing so, we can compare the accuracy obtained with and without the instance reduction on the training data, as well as the computational time needed to run the experiments. We also calculated the storage space that we were saving by reducing the size of the datasets, so as a metric to also compare computational cost reduction in terms of storage space saving.

5. Analysis of the Instance Reduction Evaluation (Accuracy, Efficiency and Storage)

Before advancing to the results, we would like to emphasize that due to the high complexity of the DROP3 algorithm, it also had an extremely high run time, and so the computational cost to run the different experiments with this technique was high. In our work, we had to resign ourselves to using only the other two metrics to run experiments, as we couldn't accomplish any test with the DROP3 algorithm. Even though, an implemented code is presented with this

document and could be used for instance reduction for the given datasets. Again, for the Adult dataset we only used the first 1000 instances.

In order to calculate the storage space gained by performing reduction, we divided the length of the reduced training dataset by the length of the original dataset. This resulted in a percentage indicating the proportion of the original training data that the reduced data set actually represented.

To account for the reduction technique for each dataset we compared the accuracy, efficiency and storage achieved on the best KNN-classifier for each reduction method. As in the previous section, the scores were normalized and weighted. Once more, the accuracy got a slightly higher rate (40%) than the other methods (30% each). All normalized scores are plotted below, showing each one of them the values obtained for both RNN (in blue) and RENN (in red) per each dataset. Later in this section the results are discussed comparing accuracy, run time values and percentage of data set reduction.

As can be seen in the following figure, for the Adult dataset the RENN reduction method worked best, however only with a slight difference.

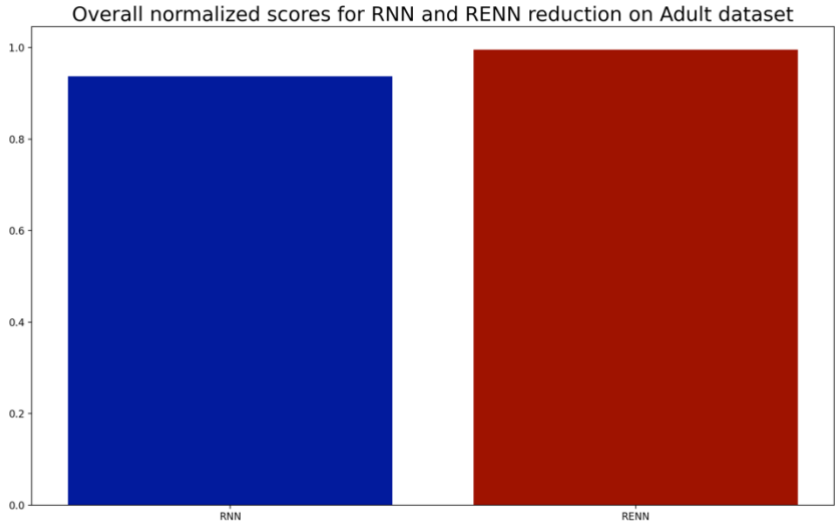

Figure 11 – Adult dataset instance reduction results

Table 1 - Adult dataset instance reduction results

Technique	Accuracy	Runtime	Data reduction
RNN	75%	0.1 seconds	33%
RENN	77%	0.04 seconds	77.7%

The normalized scores for the Vowel dataset was higher for the RNN-reduction compared to the RENN-reduction with a much higher accuracy and better runtime (Figure 12).

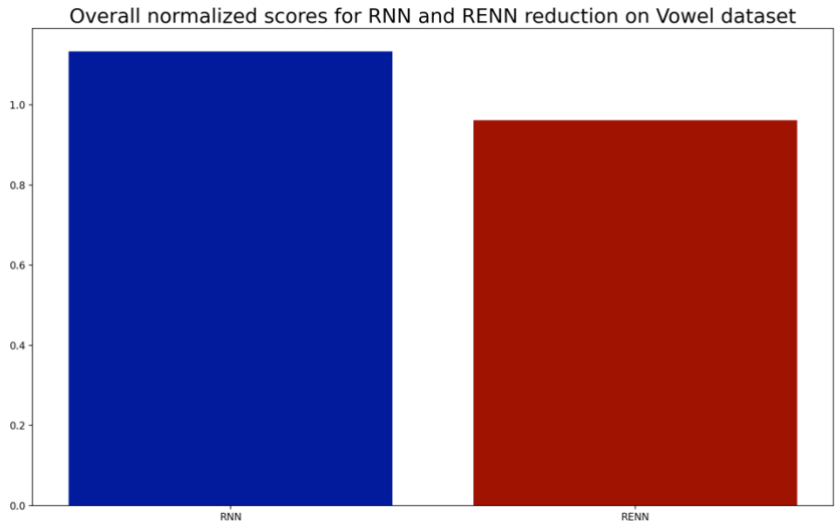

Figure 12 - Vowel dataset instance reduction results

Table 2 - Vowel dataset instance reduction results

Technique	Accuracy	Runtime	Data reduction
RNN	95.1%	0.058 seconds	90%
RENN	79.3%	0.151 seconds	91.2%

Finally, as shown in Figure 13 the best reduction method for the pen-based dataset was RNN with a better accuracy and runtime.

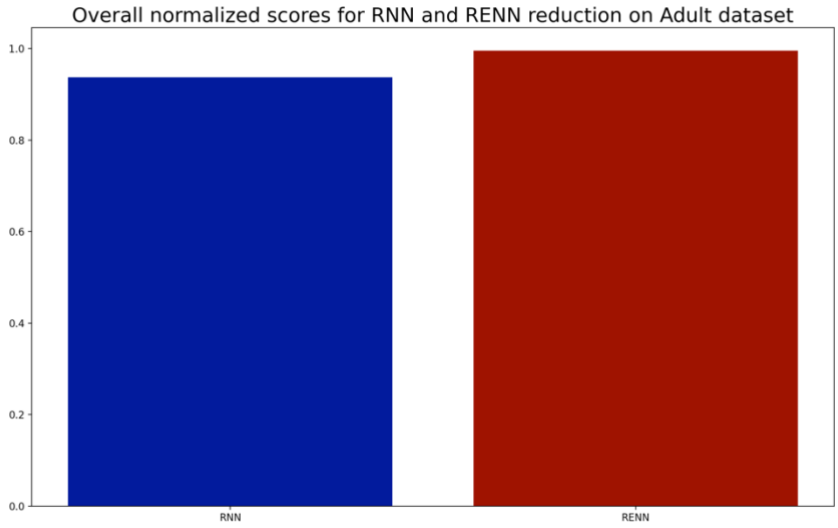

Figure 13 - Pen-based instance reduction results

Table 3 - Pen-based dataset instance reduction results

Technique	Accuracy	Runtime	Data reduction
RNN	98.33%	0.1244 seconds	96.51%
RENN	84.09%	0.069 seconds	98.59%

Now to compare the classifiers on the original dataset and then on the reduced dataset we found that the Adult dataset had almost the same performance with 77.1% accuracy on the original data compared to a maximum of 77.0% of the reduced data with RENN. On the other hand, the runtime was more than ten times higher on the original dataset than in the reduced one, and the storage was reduced by 77.7%, also both using RENN method. As a result, in the case of the adult dataset the RENN reduction greatly helped to improve the runtime and reduce storage without losing accuracy and would therefore be the preferable approach.

For the Vowel dataset we found a higher performance when trained on the original dataset and even a lower runtime than on the reduced version. As stated before some runtimes might have turned out to be wrong due to runs on different machines, and for this reason it is difficult to draw conclusions from this one. The reduction was very high for the vowel dataset by RNN, still in this case the preferred training would probably be with the original dataset in order to not lose on performance, since in any case the time required was too high.

For the Pen-based dataset the accuracy was slightly higher (by 1% - 99.46%) for the original training dataset compared to the maximum value obtained with the RNN technique for the reduced data set (98.33%). However, the runtime was around 42 times faster for the RNN-reduced classifier, which is really significant. Storage reduction was higher than in the previous cases, reaching a maximum above 98.5% for the RENN algorithm. As a result, in this case is really notorious the reduction obtained, while keeping a really high accuracy and reducing a lor the runtime.

6. Conclusions

After this work the first thing that we would like to emphasize is the importance of the computational cost and resources required to perform Machine Learning research with data containing high numbers of instances. In our case we have experienced with this work the relevance of instance reduction in order to reduce the amount of required storage as well as the run times for an algorithm to perform a classification prediction on unseen data. Furthermore, and far from reducing instances without any previous analysis, we have been able to understand the significance of finding an optimal procedure to remove those irrelevant instances, or at least those that don't contribute much to the performance of a Lazy Learning algorithm such as the k-Nearest Neighbors.

The gain in storage space and the reduction in run time is really valuable when working with large data sets, and we know that in Machine Learning and Artificial Intelligence research it is common to work with much bigger data than the used for this work. It is also true that usually the computational resources are higher for these research fields.

In any case, the reduction of the size of a data set is always interesting to be studied because, as it has been seen in this work, it can increase the efficiency of the algorithms used to predict outcomes without giving up on accuracy. In fact, it may be the case that the model can generalize equally or even better than using the whole dataset, since some overfitting is also reduced. Even the time to perform instance reduction may be high sometimes, in some cases it will be worth to compute it since the algorithm's training time will be much lower for the experiments to be performed.

Other techniques to reduce the size of a dataset have been seen during the course of this subject. Concretely, some feature-reduction algorithms such as PCA have been applied in previous works. We think that probably a combination of both approaches, instance reduction and feature selection could actually improve the performance of a Machine Learning algorithm even losing less accuracy on predictions than using only one approach, since relevant features would be kept and consequently the amount of data would be reduced even more. Even so, that is left for further experiments, and in this one feature weighting approaches have been used to also define and value more those features than are more significant.

Based on the report, the aim was to use Lazy Learning and apply it to different datasets using a K-Nearest Neighbor (KNN) algorithm, and analyze if instance-reduction techniques could increase accuracy, efficiency, and storage in the KNN algorithm. To do this, the report implemented three distance functions (Minkowski, Cosine, and Hassanat), three voting methods (majority class voting, inverse distance weighted votes, and Sheppard's work), and three feature weighting methods (equal weighting, information gain, and ReliefF). The results of the evaluation metrics, accuracy and efficiency, were then analyzed to determine the best combination of methods for the KNN algorithm. The report found that the best combination for the Adult dataset was the Cosine distance function, using 3 neighbors, inverted distance voting method, and equal feature weighting method. The best combination for the Pen-based dataset was the Minkowski distance function using 1 neighbor, Sheppard's work method, and also equal feature weighting method. Finally, the best combination for the Vowel dataset was the Minkowski distance function with again 1 neighbor, majority class voting method, and ReliefF feature weighting method.

Overall, the report concludes that using reduction techniques can increase efficiency without losing accuracy in a KNN algorithm, and that the best combination of methods for each dataset varied depending on the specific characteristics of the dataset. This is also a relevant point, since a model defined by different metrics can perform best on a given set of data and worse on another, since not all data explain the same. The underlying information that can be extracted from data is also dependent on the algorithm used. To see it from a different perspective, we can say that using different algorithms is similar to looking at the data from different points of view, which will also lead to different results.

With this being our last work in this Introduction to Machine Learning course, we can say that it is not so easy to actually develop an optimal model to perform any work, and that we also have to take into consideration that it might be the case where more than one model can actually act as the optimal one. To understand the data, finding the relationship between features and instances, and to define the right metrics for a model is a time consuming task as well as it requires computational skills and resources.

All in all, we want to conclude by saying that we enjoyed this course, and we hope and expect to keep learning from this knowledge field that now starts to style down in our minds.

7. Resources

Breaking Ties in K-NN Classification | LinkedIn [WWW Document], n.d. URL https://www.linkedin.com/pulse/breaking-ties-k-nn-classification-nicholas-pylypiw/ (accessed 12.22.22).

Abu Alfeilat, H. A., Hassanat, A. B., Lasassmeh, O., Tarawneh, A. S., Alhasanat, M. B., Eyal Salman, H. S., & Prasath, V. S. (2019). Effects of distance measure choice on k-nearest neighbor classifier performance: a review. *Big data*, *7*(4), 221-248.

Wilson, D. R., & Martinez, T. R. (2000). Reduction techniques for instance-based learning algorithms. *Machine learning*, *38*(3), 257-286.

Tomek, I. (1976). An experiment with the edited nearest-neighbor rule.

Gates, G. (1972). The reduced nearest neighbor rule (corresp.). *IEEE transactions on information theory*, *18*(3), 431-433.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research*, *7*, 1-30.