Birds of a feather flock together

A review of the k-nearest-neighbors algorithm on the waveform dataset

Baptiste Mathevon

Abstract

The K-nearest-neighbors algorithm is among the earliest machine learning algorithms. Its simplicity makes it very popular among various domains and applications. Some major drawbacks of this latter are its computational complexity, its poor performance when faced with imbalanced data, or even its weakness against the curse of dimensionality. Here, we will present solutions to see how it is possible to handle these drawbacks.

1. Introduction

The KNN algorithm can be seen as the famous saying *"birds of a feather flock together"*. In its basic version, the classification of a new sample consists in assigning it the majority class among its k nearest neighbors. However, this process presents multiple issues. First, the choice of the number of neighbors is crucial: too many k's can lead to under-fitting while too few can lead to over-fitting. Second, finding the nearest neighbors requires a lot of calculations. Third, there is a risk of curse of dimensionality since distance measurements lose their meaning as dimensions increase. Fourth, KNN is sensitive to imbalanced data as it affects the decision boundaries. Finally, the KNN classification strongly depends on the choice of the distance metric as this distance impacts how close two points are.

Here we systematically test a range of methods aimed at dealing with these issues. First, we test if using cross-validation to find the k best suited to the data could help to find the right compromise of the number of neighbors. Then, we evaluate whether reducing the size of the dataset or modifying the way the KNN finds the k nearest neighbors are viable methods to reduce the computational cost. Thirdly, we test whether reducing the dataset's dimensions can help curing the curse of dimensionality. Next, we analyze if it is possible to overcome imbalanced data using well-known machine learning's techniques. Finally, we assess the impact of the distance metric on the algorithm.

2. Experimental set-up

The waveform dataset is made up of 5000 examples. Each example is composed of 21 features and belongs to one of the three classes. The dataset is split into two sets : the training set and the testing set. The training set is composed of 4000 randomly drawn examples and the testing set is composed of the 1000 remaining examples. The optimal Bayes classification rate of the dataset is 86% accuracy. Python 3.12.3 is used and the random seed is fixed at the start to get reproducible results.

3. Tuning k by cross-validation

The value of k is the keystone of an efficient KNN algorithm. Small values of k makes the decision boundary very sensitive to noise or outliers, it has a high variance. It tends to overfits the training data and make the decision boundaries very sharp. In contrast, a large value of k can lead to underfitting as it smooths the decision boundaries (fig. 1). Choosing the right value of k balancing between these two aspects is thus essential for having a low generalization error.

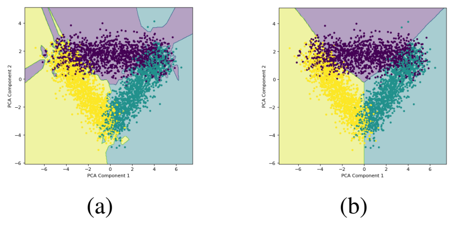

Figure 1: Decision Boundaries of the KNN. (a) With $k = 1$. (b) With $k = 1000$.

Cross-validation divides the training set into n folds, training the model on $n-1$ folds and validating on the remaining fold to tune hyperparameters. Here, 5-fold cross-validation evaluates different k values by averaging classification accuracy across folds, selecting the k that maximizes mean accuracy.

Figure 2 shows the results of the cross-validation. Initially, accuracy increases as k grows, reducing noise sensitivity and over-fitting, until it reaches an optimal value k^*. Beyond this point, accuracy decreases due to under-fitting, eventually dropping to random guessing as k approaches the training set size. Cross-validation successfully identifies k^*, balancing over-fitting and under-fitting, and this value also performs well on the test set, validating its effectiveness for unseen data.

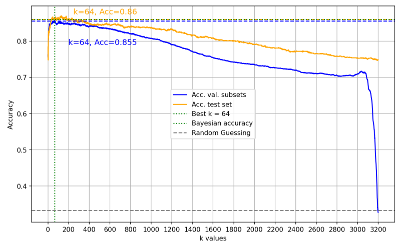

Figure 2: Accuracies over the validation subsets (blue) and over the test set (orange) with respect to k.

4. A Computational Problem

Finding the k-nearest-neighbors may pose a time complexity problem. Reducing the number of examples in the training set can be a solution as it will reduce the number of distances to be calculated. We must not modify the decision boundaries of our classifier. Another technique is to reduce the number of calculations required by taking advantage of the known distances between the training data.

4.1. Data Reduction

Here, we show that reducing the number of examples in the training set does not greatly affect decision boundaries. Some examples are not interesting for our classifier, such as outliers or those inside the Bayesian region. The Bayesian region being the area where distributions of the three classes overlap and the examples are then unclassifiable, which boils down to random guessing. So we first remove those examples. Decision boundaries are also affected by examples located on the outer border of the clusters. Thus, it is possible to remove the examples inside without modifying the boundaries (fig. 3).

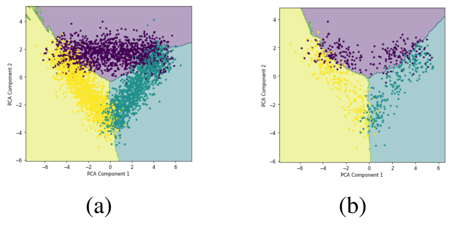

Figure 3: Decision boundaries with $k = k^*$. (a) Using the original training set S. (b) Using the condensed training set $S_{condensed}$.

We observe that the decision boundaries did not change much, whereas $|S_{condensed}| \ll |S|$, therefore saving a lot of computations.

Figure 4 shows that the difference between the accuracy using the original dataset and the accuracy using the condensed dataset for $k = k^*$ using $S_{condensed}$ is small. Confirming that our decision boundaries are still correct.

We also showed that the calculation time of the nearest-neighbors can be ~ 4 times faster for the condensed data compared to the original data. It can therefore be a good trade-off for reducing the computation time.

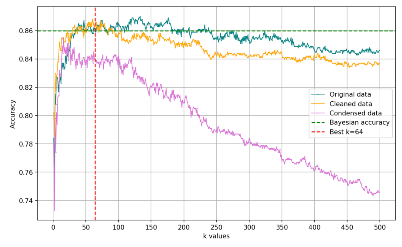

Figure 4: Accuracies over the test with respect to k using original, cleaned, and condensed training dataset.

4.2. Reducing Calculations

Taking advantage of the known distances between the training points can reduce the number of calculations needed. The goal is to find the nearest-neighbor, $k = 1$ (the strategy could be extended to $k > 1$, left for future work). The idea here is to take advantage of the triangle inequality for pruning the points that cannot be closer to the actual nearest neighbor.

By doing so, figure 5 shows that the speed version of the algorithm was able to find the nearest neighbor of the test set ~ 1.4 times faster compared to the normal algorithm using the full training set for predictions. This version can thus be very useful as the execution time gap between the two algorithms increases as the size of the training set increases.

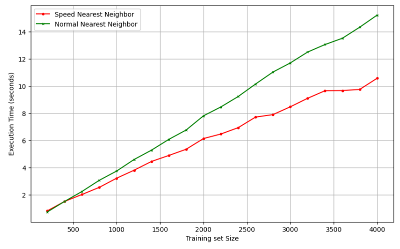

Figure 5: Execution time for finding the nearest neighbor of the test set samples w.r.t. different sizes of training subset (randomly drawn from the original training set).

5. Overcoming the Curse of Dimensionality

As the number of dimensions increases, the examples tend to be uniformly distant from each other. In an algorithm where we assume that ”closer” points are more relevant than ”farther” ones, this is an issue. This curse can sometimes be addressed by reducing the dimensions of the dataset without losing too much information. Using PCA and LDA, we reduce the dimensions of our dataset from the original size to 2. We then compute the mean accuracy of the predictions after reduction of the dimensions. Figure 6 shows that the results follow the theory, the mean accuracy decreases when the number of dimensions increases. We also observe that the accuracy of the LDA is slightly above, due to LDA’s ability to optimize class separation.

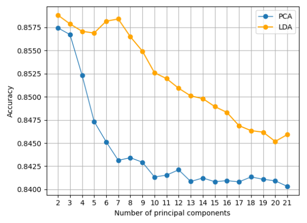

Figure 6: Accuracy after PCA / LDA with respect to the number of Principal Components.

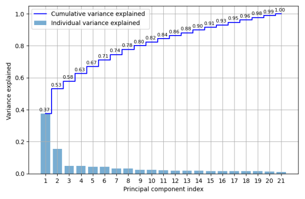

Figure 7: Explained Variance of the PCA (does not matter much as the accuracy is our main goal)

6. Imbalanced Data

Data imbalance poses a problem for machine learning models as we are often interested in the minority class. The issue lies in the score metric: by always guessing the majority classes the classifier will have good accuracy. We have to resort to another metric that better reflects the classifier's ability to predict correctly even with imbalance. This metric is the so-called f-measure which aims at balancing between the precision (proportion of real positives among the positive predictions) and the recall (proportion of real positives among the actual positives). We generate artificial imbalance in the training set and we re-tune the k using cross-validation but choosing the k maximizing the f-measure. Figure 8 shows that the new k found performs better when we keep less than 30% of the undersampled class. Using f-measure thus helps perform better when dealing with imbalance.

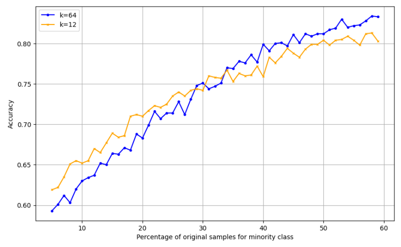

Figure 8: Accuracy comparison between the best k found earlier (blue) and the new one found using the f-measure (orange).

6.1. Other methods

We present here three techniques to deal with imbalance data as shown in the figure 9a where 20% of the minority class was kept.
First, undersampling the majority classes. We see in figure 9b that the decision boundaries after undersampling are quite similar to the ones from the original data (fig. 3).
We show in figure 9c that we can also oversample the minority class using the so-called method SMOTE to get correct decision boundaries.

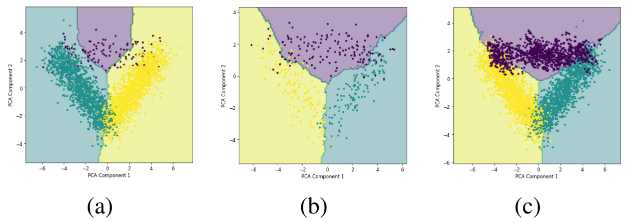

Figure 9: Decision boundaries. (a) Imbalanced Data. (b) Undersampling. (c) SMOTE.

7. Choosing the Distance Metric

In a context where the classifier decision is based on the distance metric used, it is essential to choose the right one for our dataset. Figure 10a shows the accuracy with respect to k using different distance metrics after reducing the dataset to 2 dimensions. We find that all distance metrics give more or less the same results. In 2 dimensions, this dataset does not seem to be sensitive to the measure used. However, figure 10b shows that when using the original number of dimensions, the variance between the accuracy of the different metrics is greater. It is therefore sensitive to the distance metric used, as some of them perform less well in higher dimensions. Indeed, the Euclidean distance and the Manhattan distance are the ones that perform the best and seem less affected than the others by the curse of dimensionality.

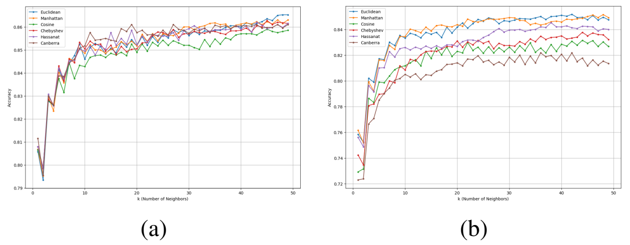

Figure 10: Accuracy w.r.t. k using various distance metrics. (a) After reducing to 2 dimensions using PCA. (b) Using the original dataset in 21 dimensions.

8. Conclusion

First, we have shown that cross-validation is indeed a good way to find the number of neighbors. Second, we have shown that reducing the size of the dataset or the number of distance calculations can improve the computation time of KNN. Third, we have seen that reducing the dimensionality can help overcoming curse of dimensionality. Then, we showed that the use of the f-measure or the use of undersampling and oversampling on the imbalance class can counteract imbalanced data. Finally, we illustrated how the distance metric impacts the accuracy of the classifier. Therefore, for its simplicity and efficiency, the KNN algorithm will probably continue to be widely used as many methods exists to overcome its issues.

References

[1] M. Sebban, "Course on introduction to machine learning." Lecture notes, Université Jean Monnet, Saint-Etienne, 2024.

[2] H. A. Abu Alfeilat, A. B. Hassanat, O. Lasassmeh, A. S. Tarawneh, M. B. Alhasanat, H. S. Eyal Salman, and V. S. Prasath, "Effects of distance measure choice on k-nearest neighbor classifier performance: A review," *Big Data*, 2019.

[3] A. A. Tokuç, "K-nearest neighbors." https://www.baeldung.com/cs/k-nearest-neighbors, 2024.