Victor MAYAUD
Hichem KHETTAB

Repport MALIS
Project n°1
Understanding k-Nns

Part I – Implementing a kNN from scratch:

First I created a k-Nearest Neighbors (kNN) algorithm prototype, initially coding the necessary functions to handle initialization, train, distance calculation, and prediction. In the training section I store the training data X and labels Y. Following this, then I optimized the prototype to improve performance. By using the scipy.spatial.distance library, I made distance calculations more efficient. Furthermore, I changed the prediction method using the argsort function for faster identification of the nearest neighbors and streamlined label determination for more effective classification. The execution times recorded for the KNN algorithms were 6.28 seconds for the initial, 0.078 seconds for the optimized, and 0.028 seconds for the Scikit-learn implementation, revealing that the Scikit-learn version is substantially faster, outperforming the optimized algorithm by a factor of 3.5.(**A-1**)

Task 2:

After developing the kNN algorithm, I focused on optimizing the number of neighbors (k). For this, I used cross-validation to avoid overfitting and to ensure that the model generalizes well to new data. In cross-validation, the data is divided into a training set and a validation set. The training set is further partitioned into 'k' equal segments or folds. Each fold takes turns serving as the validation set with the remaining ones used for training. The model is trained and validated 'k' times, with each fold being used exactly once for validation, and performance metrics are recorded. The average performance across all folds is calculated, and the k value with the highest average performance is selected as the optimal number of neighbors for the kNN model (**A-2**). This method is advantageous because it uses data efficiently and provides a reliable performance estimate for unseen data (we avoid over fitting.). Using k=39, identified through cross-validation on the training set with 5 folds, resulted in an 81.46% accuracy on the validation dataset. The accuracy is the same when using the optimized algorithm and the Scikit-learn algorithm. The selection of k=39, although not optimal for this dataset, is expected since it aims to avoid overfitting and ensure generalizability. Choosing k=27 might improve results on this dataset but risks overfitting, potentially reducing performance on new data (**A-3**). In using utils.py we can display the frontier of the both classes with a trained dataset (**A-4**).

Part II – The curse of dimensionality:

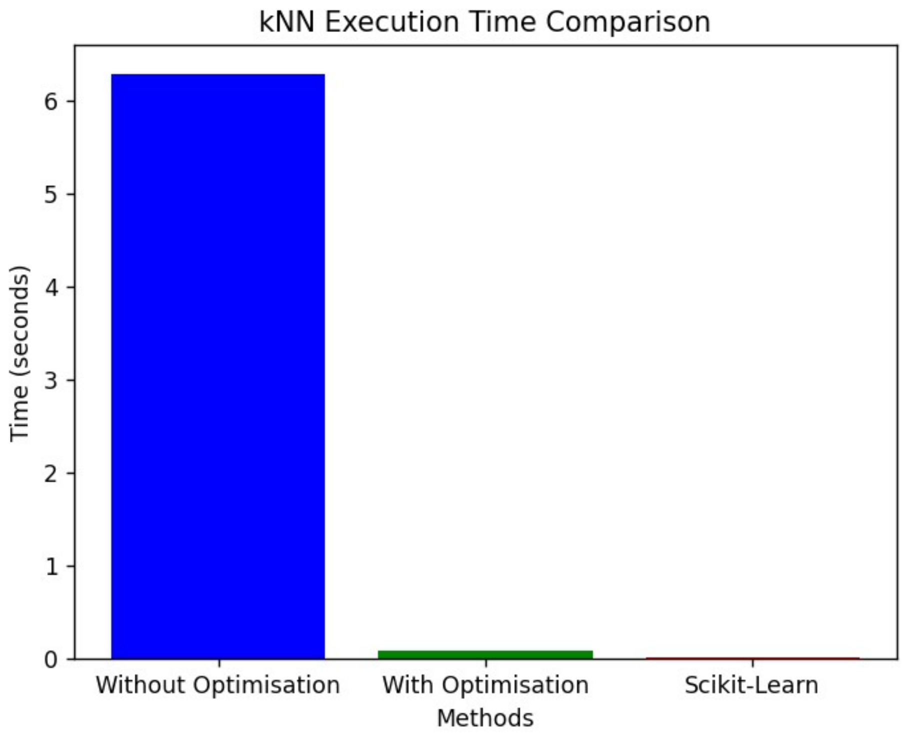
As presented in the book the curse of dimensionality is a recurrent problem in machine learning. Let us first briefly present what it is before making it in correlation with KNNs thanks to the exercises.

As explained in the book the curse of dimensionality has different manifestations. For example, for a p-dimensional unit ball sampling a small fraction of its volume demand to use a hypersphere neighborhood with a high radius. In the KNN method, we give to the algorithm a set of data and we attribute to them values. The algorithm will take the K (chosen wisely) nearest neighbors to determine the class of a new data. For example, if a majority of the nearest neighbors belong to Class 1 the new data will be put in Class 1.

The 2.4 exercise shows a manifestation of the curse of dimensionality since the equality we prove shows that for a high value of p, the median distance from the origin to the closest data point goes higher. So the data points are mainly spread on the surface of the p-dimensional sphere and so it will be harder for the KNN algorithm to “reach” these points. In the 2.5 exercise, we show that the zi, which are the projections of the data in the training set, are distributed uniformly around a prediction point x0. So it makes it difficult for the KNN algorithm to choose the appropriate neighborhood.

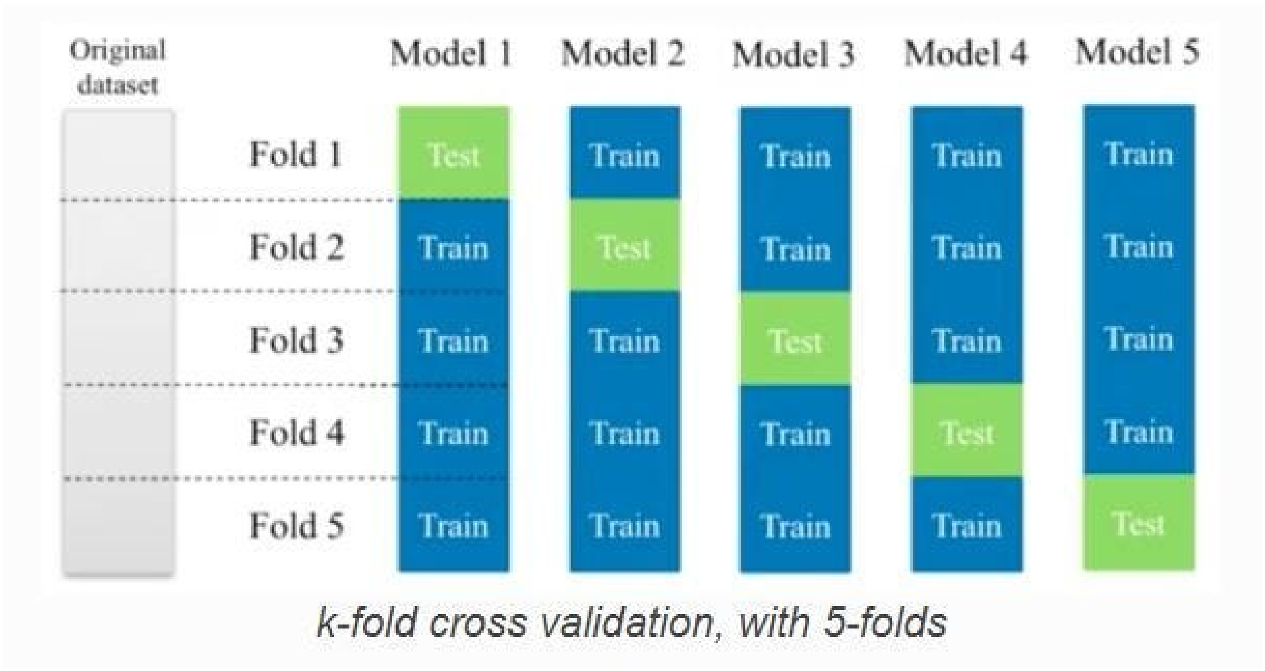
Annexe :

A-1:



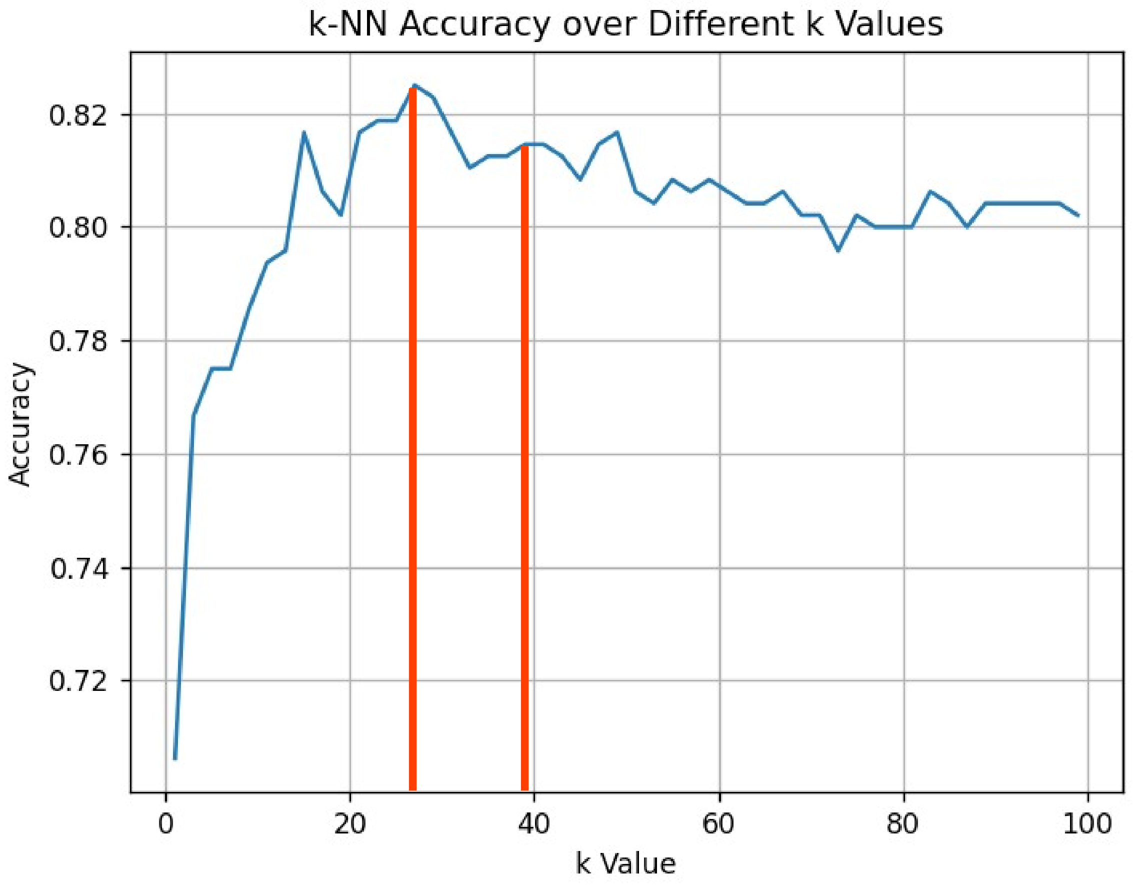
kNN Execution Time Comparison
Time (seconds)
0
1
2
3
4
5
6
Without Optimisation
With Optimisation
Scikit-Learn
Methods

A-2:

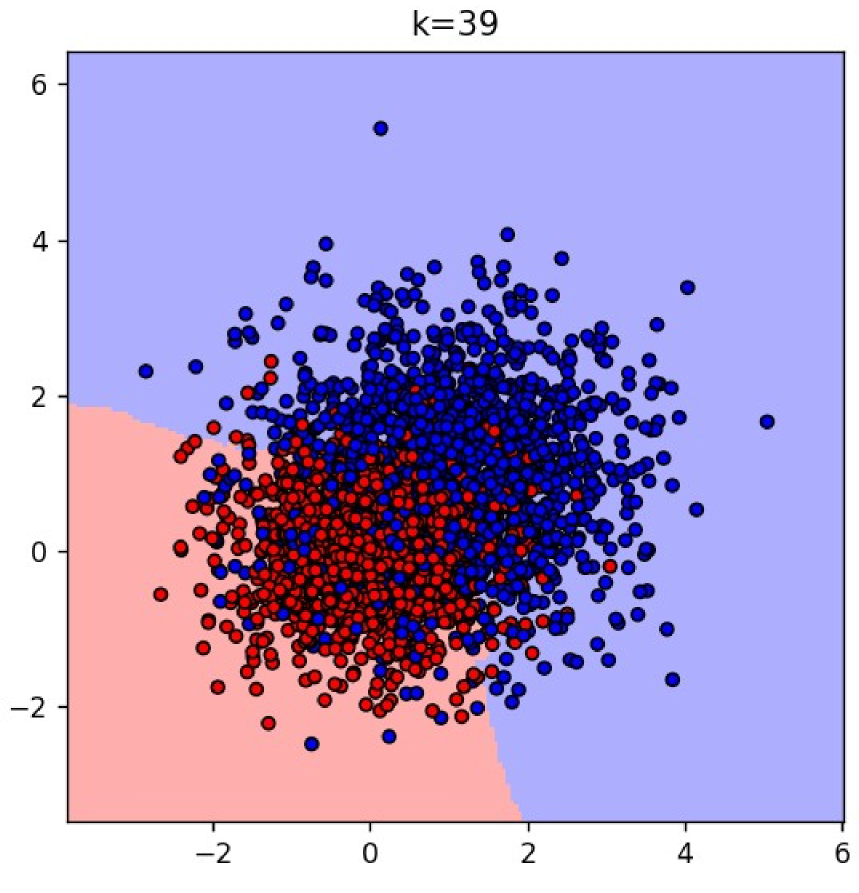


Original dataset
Model 1
Model 2
Model 3
Model 4
Model 5
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Test
Train
k-fold cross validation, with 5-folds

A-3:


k-NN Accuracy over Different k Values
Accuracy
k Value
0.72
0.74
0.76
0.78
0.80
0.82
0
20
40
60
80
100

A-4:


k=39
6
4
2
0
−2
−2
0
2
4
6