Grenoble INP - Ense3

AI and Autonomous systems

Lab 1 - M2-MARS

Supervised Learning : The K-NN algorithm and model assesment

Author :
Souhaiel BEN SALEM

6 octobre 2021

Lab Objectives: The Objective of this Lab is to explore the K-Nearest Neighbor algorithm through examples, understand its advantages and disadvantages and be able to interpret the results obtained when applying this algorithm.

Introduction:

K-nearest neighbors (KNN) is a type of supervised learning algorithm used for both regression and classification. KNN tries to predict the correct class for the test data by calculating the distance between the test data and all the training points. Then select the K number of points which is closet to the test data. The KNN algorithm calculates the probability of the test data belonging to the classes of 'K' training data and class holds the highest probability will be selected. In the case of regression, the value is the mean of the 'K' selected training points.

1. **In view of the results obtained, which values of the number of neighbors (or range of values) seem optimal? Verify that these values give boundaries that seem coherent to you?**
   By looking at the error rates plots, we can see that the training error begins to stabilize around K=10. Moreover, for this stable phase (the phase between over-fitting and under-fitting) we get minimum error rates (training error and testing error) for K in range [10,12].
   The optimal K value that would give us the best classification result would be in this interval. For instance we could choose K to be equal to 11 (we note that the optimal K is almost equal to $\sqrt{N}$ where N=1000 stands for the number of samples).

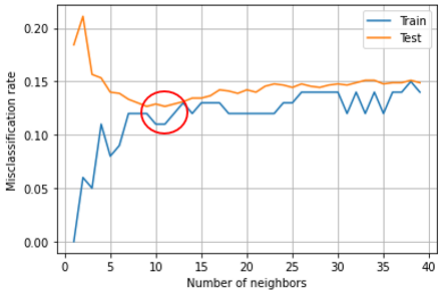

Figure 1: optimal K range

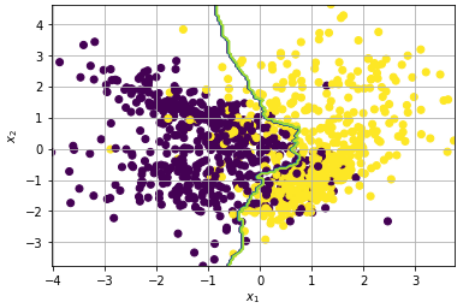

Figure 2: Classification results for k=11

2. **n this plot, which edges of the curve correspond to the over and under-fitting cases respectively?**
   If we increase the number of neighbors we could clearly see the two cases of over-fitting and under-fitting:
   - The zone highlighted in Red represents the over-fitting case which occurs for small values of K (notably for K=1) . In this case the bias will be 0 which means that training data will be perfectly predicted, however, when it comes to new data (in test set), we have a higher chance to get a wrong prediction , which causes high variance.
   - The zone highlighted in Green represents the Under-fitting case which occurs when we

choose a high value of K (this case becomes clear for K > 95 in our case). In this case the algorithm becomes too simple and fails to predict the training data due to its high bias/low variance nature.

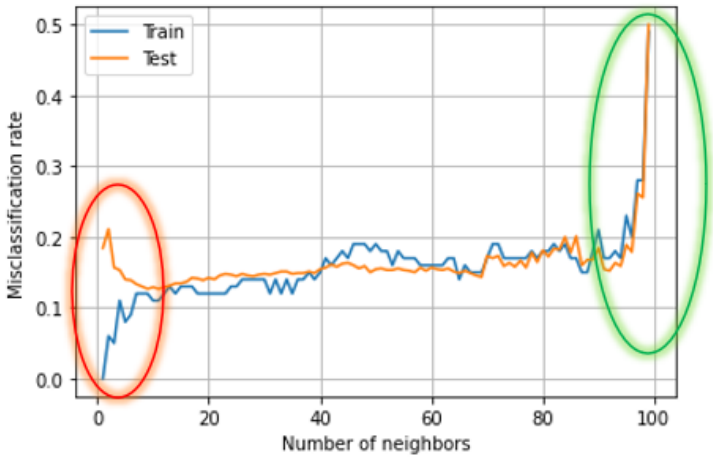

Figure 3: over-fitting and under-fitting

3. **Why the train error can not be used alone to select the method (here the value of K)?**
   Training error by itself is not a good metric to determine the performance of the algorithm and and select the number of neighbors K. For instance, training error by itself cannot spot the over-fitting as that bit is done by the validation data-set. The training phase can over-fit especially for complex models and that is when cross validation is very critical.That is why we need testing error to keep track of the algorithm's performance when it comes to new (unseen) data and whether the model is generalizable or not.
   In short, when using both training and testing errors, we can determine the best bias/variance tradeoff and thus the value of K to choose.

4. **What happens when you decrease gradually the class_sep parameter from 1 to 0.5 ? In particular, how can you explain the shape of the test error curve?**
   When we decrease the **class_sep** parameter from 1 to 0.5, the overlapping between the two classes

becomes gradually important and thus the learning process becomes more and more difficult because the prior probability of the two classes become almost equal.
The evolution of the error curve is presented byy the following figures:

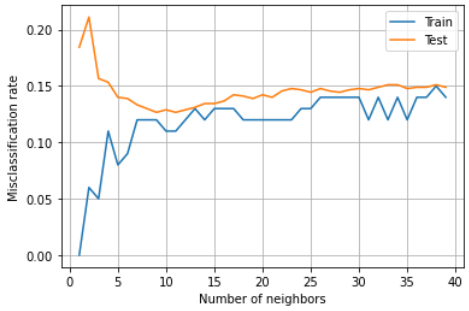

Figure 4: class_sep=1

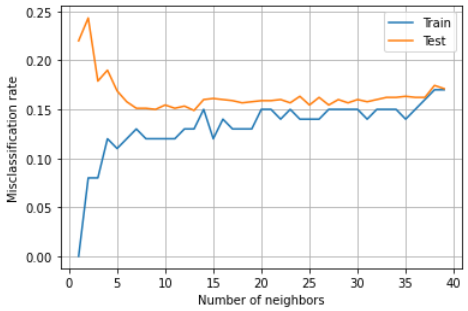

Figure 5: class_sep=0.9

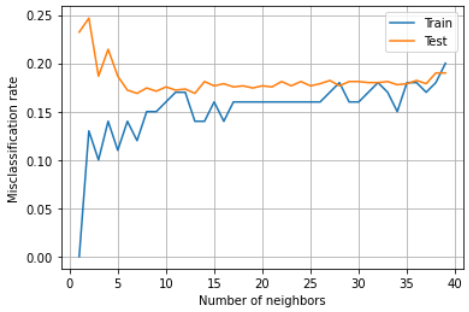

Figure 6: class_sep=0.8

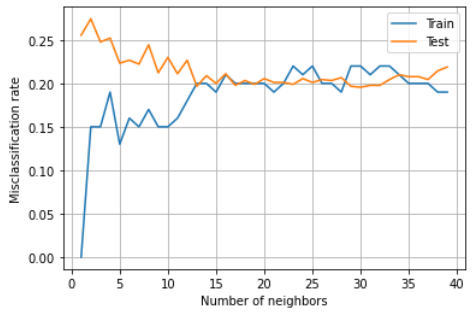

Figure 7: class_sep=0.7

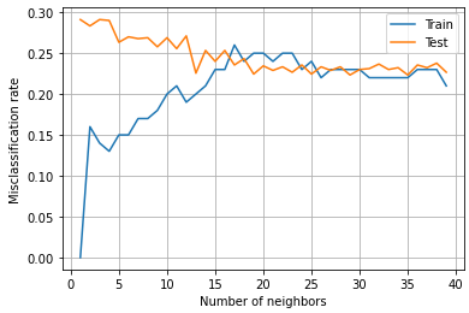

Figure 8: class_sep=0.6

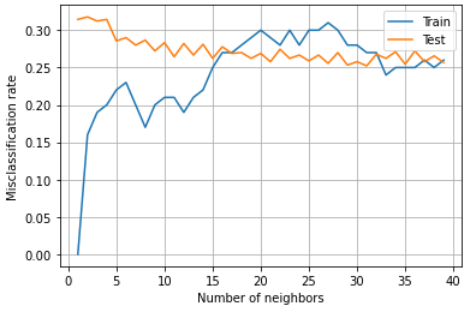

Figure 9: class_sep=0.5

As the overlapping between the two classes becomes more and more important we notice that the K-NN algorithm can't keep up with the data imbalance which is clearly shown by the fact that the training error keeps augmenting which means that the algorithm's accuracy keeps going down. This is because the overlapping is making both classes more and more inseparable to our K-NN algorithm.

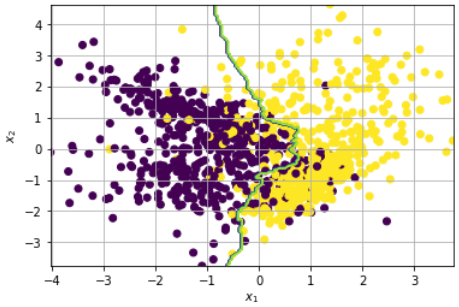

Figure 10: classification for class_sep=1

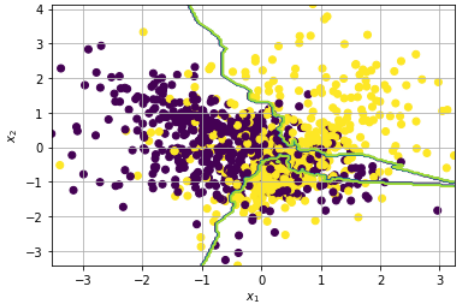

Figure 11: classification for class_sep=0.5

0.1 Iris_K-NN

Exercice:

1. **What is the (estimated) optimal value for the knn parameter?**
   By looking at the plot of the overall misclassication rate function of the hyperparameter, the optimal value of the K-NN algorithm is in the range of [10,15]. We can estimate K=13, which corresponds to a region where the prediction becomes smoother and both the training and validation errors are minimal.
2. **What is the optimal error rate?**
   The optimal error rates for the optimal K=13 are:
   - optimal training error rate: 0.028
   - optimal validation error rate: 0.015
3. **Replace the default euclidean metric used to determine the nearest neighbors by "chebyshev" one and compare your results**
   After changing the metric parameter from "euclidean" to "chebyshev" which corresponds to changing the way we compute the distance between a point and its neighbors, we notice that the misclassification rate went higher in general. We also notice that when using the chebyshev metric, the training and testing errors have almost the same shape which means a low variance/low bias situation.

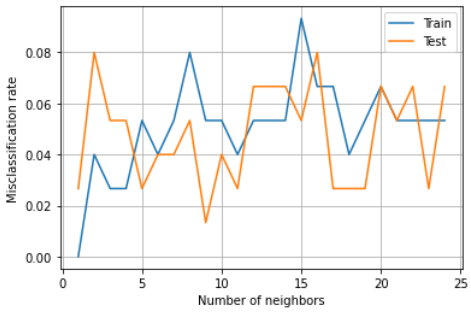

Figure 12: using the chebyshev metric

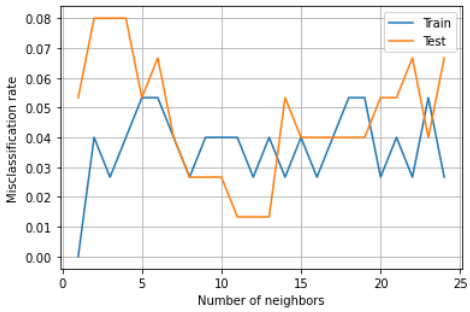

Figure 13: using the euclidean metric

4. **What happens when we change the test and train sets (e.g., try on several runs by changing the random_state seed)?**
   random_state number splits the test and training data-sets with a random manner. This means that eveytime we change the seed and run our code, a new random value is generated and the train and test data-sets would have different values each time which leads to different results (performance and error rates).
   random_state value can have significant effect on the quality of our model (by quality we essentially mean accuracy to predict). It is important to find the best random_state value to provide us with the most accurate model. We can do that by splitting and cross-validating our algorithm by assigning random numbers to random_state parameter.

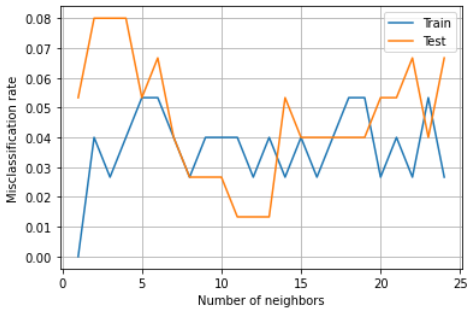

Figure 14: random_state=1

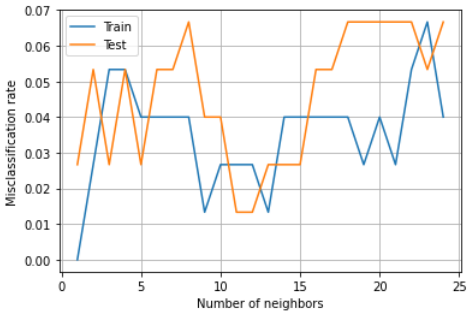

Figure 15: random_state=2

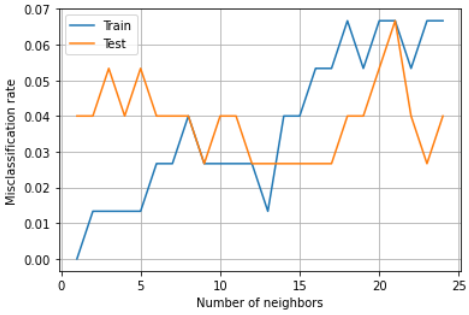

Figure 16: random_state=3

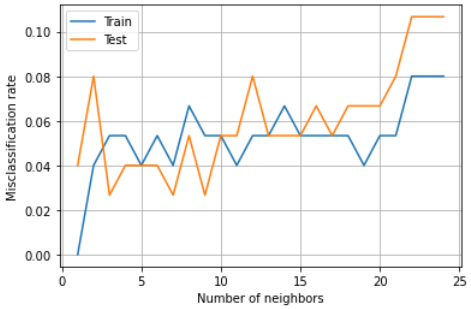

Figure 17: random_state=5

5. **How confident are you on these best scores/hyperparameter values?**
   The best scores depend highly on the training and testing sets and the optimal best hyperparameters are only estimated but not determined by using a sophisticated method.
   In order to be more confident of our choice of hyperparameters/scores, we need to apply more efficient methods in order to determine with high confidence the best representation (hyperparameter K) that maximises the performance of our algorithm.

1 Model assesment

1.1 Validation and model selection

Exercice:

1. **Recall what are the benefits of cross-validation w.r.t the simple validation approach**
   The holdout method's (test_train_split) score is dependent on how the data is split into train and test sets. Cross-validation on the other hand, allows us to test the model on multiple splits in order to obtain a better understanding of how it will perform on unknown data. In fact data is the fuel of any machine learning algorithm/model and it's highly improbable in real world applications that we'll be able to find a large dataset from which we can create an efficient model. If we proceed with conventional validation in the case of data scarcity (which is the normal circumstance), we will almost certainly reduce the dataset size by 20–30%. There will be no such reduction in dataset size when doing k-fold cross validation. Furthermore, by doing a k-fold cross validation, one can avoid overfitting.
2. **Is the CV accuracy estimate consitent with the accuracy estimate obtained on the test set?**
   Yes, the CV accuracy estimate is consistent with the accuracy estimate obtained on the test set (unseen data) , this is especially true for n (number of neighbors) in range [3,7] , in particular n=3 which was chosen by the **GridSearchCV** method as the best hyperparameter.
3. **Apply the same CV procedure to the iris data set (see and adapt notebook** $2_knn/N2_iris_knn.ipynb).Compareyourresultswiththesimplevalidationapproach(fixedtrainingandtestset)usedin2\_kn$

After applying the cross validation methon on the Iris dataset, we notice that we are able to build a mode accurate model and be more confident when choosing the optimal hyperparameter K.
The results for cross validation on the Iris dataset using the neighbor [1, 7, 9,10, 11,12, 13,14, 15,23,25,27,33,36,40] are as follow:

```
Results for the cross-validation:
{'n_neighbors': 10}
        accuracy = 0.942 (+/-0.041) for {'n_neighbors': 1}
        accuracy = 0.950 (+/-0.033) for {'n_neighbors': 7}
        accuracy = 0.942 (+/-0.041) for {'n_neighbors': 9}
        accuracy = 0.958 (+/-0.053) for {'n_neighbors': 10}
        accuracy = 0.958 (+/-0.053) for {'n_neighbors': 11}
        accuracy = 0.950 (+/-0.062) for {'n_neighbors': 12}
        accuracy = 0.950 (+/-0.062) for {'n_neighbors': 13}
        accuracy = 0.950 (+/-0.062) for {'n_neighbors': 14}
        accuracy = 0.958 (+/-0.053) for {'n_neighbors': 15}
        accuracy = 0.950 (+/-0.062) for {'n_neighbors': 23}
        accuracy = 0.933 (+/-0.041) for {'n_neighbors': 30}

Results for the Test data set: 0.967
```

Figure 18: Result of the cross validation on Iris dataset

For the same neighbor used in the previous notebook (k in range [1,25]), the mean scores and errors for training and testing are as follow:

```
plt.plot(test_scores, label='test_scores')
plt.plot(train_scores, label='train_scores')
plt.legend(loc='best')
plt.grid()
plt.show()
```

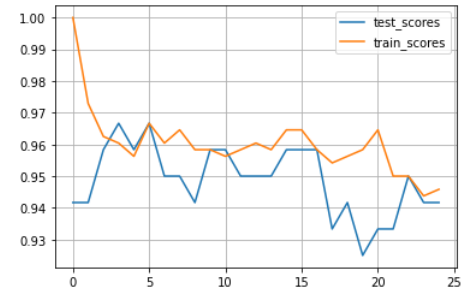

Figure 19: mean scores

```
plt.plot(test_error, label='test_error')
plt.plot(train_error, label='train_error')
plt.legend(loc='best')
plt.grid()
plt.show()
```

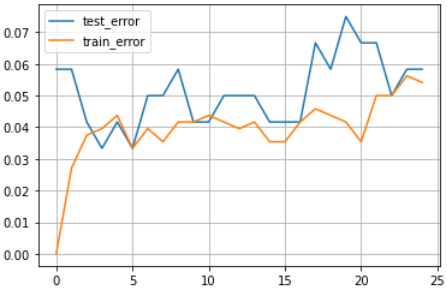

Figure 20: mean error

We notice that the variance is minimized compared to when we used the simple validation method, if fact the model we got here is more consistent and these graphs are more indicative of how well our K-NN would perform on unseen data than the graph we got when we used the simple validation approach.

1.2 Nested versus non-nested cross-validation

Exercice:

1. **Explain why the prediction performances estimated by non-nested CV is optimistic with respect to the nested CV ones.**
   The non-nested cross-validation method divides the data into training and testing parts, while nested cross-validation divides the data into training, validation and testing parts, forming two cross validation steps one happening in the outer loop (error estimation) within which the hyperparmeter optimization in nested (inner loop). Because non-nested CV uses the same data to tune model parameters and evaluate model performance, the model can become biased to the used dataset and may lead to overfitting or high optimism (overestimating the model's performance) compared to the nested approach which prevent data leaking between the training process and error estimation process.
2. **Which estimator do you think most reliable for the test error?**
   The nested CV is the most reliable for the test error because it gives a more realistic idea of the performance of the model by preventing it from being biased to the used dataset.
3. **Why use a separate test set in addition to the validation set?** The "validation dataset" is mainly used to describe model assessment when tuning hyperparameters and data preparation, whereas the "test dataset" is mainly used to describe model evaluation when comparing it to other final tuned models.
4. **What are the benefits (or disavantages) of nested cross-validation w.r.t. the "separate test set" split approach?**
   - **Advantages of nested cross validation w.r.t "separate test set" approach**: for small datasets, the nested CV approach would allow us to cover all the dataset. This important because as we stated earlier, data is very important for building the model and sometimes we can't afford to take away a part of our dataset for testing purposes only.
   - **Disadvantages of nested cross validation w.r.t "separate test set" approach**: A downside of nested cross-validation is the dramatic increase in the number of model evaluations performed (this is clearly apparent for when the number of models evaluated is large).

     For instance, if n * k models are fit and evaluated as part of a traditional cross-validation (with a separate test set in this case) hyperparameter search for a given model, then this is increased to k * n * k as the procedure is then performed k more times for each fold in the outer loop of nested cross-validation.
     This fact renders the computational cost of nested CV heavy.

Conclusion:

The K-NN algorithm is a simple, yet powerful algorithm for small to medium datasets with no imbalances and it can use for both classification and regression but it does not work with large datasets , needs feature scaling and it is sensitive to noisy data and outliers. During this lab, we saw how to implement and test the K-NN algorithm on different datasets using the Scikit-Learn library. We then explored different methods for model assesment and hyperparameter tuning (number of neighbors) suck as cross validation , nested cross validation and the ”separated test set approach” (leave-one-out cross validation) and compared the different result we got from each approach.