Practical Machine Learning Assignment 1

Part 1 – Development of k-NN Algorithm

The aim of this part is to develop the k-NN Algorithm in Python using as training and tests data the version of the Mammographic Mass Data Set from the UCI dataset (training and data located in the cancer.zip directory).

In this part, the k parameter (number of neighbors) of the k-Nearest Neighbor algorithm is 3 (k=3) and the distance calculation is on the standard Euclidean basis.

Let's now present the Python code of the k-NN Algorithm which has been implemented.

The code has 6 functions:

- ReadData:
  The aim of the function is to read the csv data format from the UCI dataset which is inside the compressed zip file cancer.zip.
- calculateDistances:
  It is used to make the calculation of the distance between on query request (one row of the test dataset) and each point of the data training.
- calculateDistanceskp:
  This is another version of the function "calculateDistances" which has the hyper parameters k, for the number of neighbors and p, for the Minkowski distance parameter setting. This function is used for the part 2 of the assignment
- queryResult:
  This function aims to return the diagnosis result based on the number of the nearest data training points.
- queryDistWeightVoteResult:
  This is another version of the function "queryResult" which returns the diagnosis result based on the distance 's weight from the data training points.
- AccuracyResult:
  It is used for calculating the number of the correct prediction in comparison to the real diagnosis data and returns the accuracy rate.

1) **ReadData:**

This function has 2 arguments “filename” and “typef”. The “filename” argument corresponds to the path and the name of the dataset file and the “typef” argument is used to specify the type of the dataset (either “training” or “test”).
First, a numpy array “dataArray” is created in order to store the complete cvs data format (It can be either the training data or the test data). The command “genfromtxt” from numpy is used:
dataArray = np.genfromtxt(filename, delimiter=",")

Then, a test on “typef” variable ensure if the dataset is a test dataset or a training dataset. If it is a training dataset, the outliners points are supressed.
Afterward a slice of the numpy array “dataArray” is made in order to separate the features data from the diagnosis result data. At the end, the function returns 2 arrays:

_ One array “dataInfo” which corresponds to the values of the data features. The data has 5 features which corresponds to the 5 first columns of the dataset.

_ Another array “dataResult” which is the result of the diagnosis of the cancer (benign or malignant). This arrow is in fact the last one of the datasets.

2) **calculateDistances:**

This function has also 2 arguments “data” and “query”. “data” is a 2D numpy array which corresponds to all training data of the features. Each row is a training points and each column is a feature. “query” is a 1D numpy array which is the query point (One row of the test data).
Based on the 2 arguments, the distance is calculated between the “query” and the list of points in “data”. However, depending the type of the feature, the distance cannot be processed in the same way. Typically, features of type ordinal or integer should get their distance calculated on the basis of the Euclidean standard (as per the specification); whereas, features of type nominal should get their distances calculated according to the Hamming metric. Here is the list of the features of the data and the way their distance calculation has been implemented in the code:

	BI-RADS	**Age**	**Shape**	**Margin**	**Density**
Type of feature	Ordinal	Integer	Nominal	Nominal	Ordinal
Range	1-5	0-max(data)	NA	NA	1-4
Distance Calculation	Euclidean	Euclidean	Hamming	Hamming	Euclidean

Therefore, the 2 first columns and the last column are calculated based on Euclidean distance (in the Python code, slice of array “*calcDistance_tmp[:,0:2]*” and “*calcDistance_tmp[:,4]*”). But the 3rd and 4th column are calculated based on the

Hamming distance metric (in the Python code, slice of array "*calcDistance_tmp[:,2:4]*").
To calculate the Hamming distance, we need to evaluate the inequality of the value: if the test value is different from the training value, then the distance is equal to 1. Otherwise, the distance is 0 (in the Python code, "*(data[:,2:4] != query[2:4])*". This returns a Boolean array; but it is converted into a float array when it is affected to the slice of numpy array "*calcDistance_tmp[:,2:4]*").
Another difficulty is the data range for each ordinal or integer feature: Each of them are at different scale. Typically, the "Age" feature range is very high compared to the "BI-RADS" feature and the "Density" feature. Therefore, we need to normalize the distance for each feature (in the Python code, 1D numpy array "*normalized_val*" and "*np.max(data[:,4], axis=0) - np.min(data[:,4], axis=0)*"). The Python code of the above explanation is as following:

```
normalized_val = np.array([BiRadMax-BiRadMin, AgeMax-AgeMin], dtype=float)
calcDistance_tmp[:,0:2] = (data[:,0:2] - query[0:2])/normalized_val
calcDistance_tmp[:,2:4] = (data[:,2:4] != query[2:4])
calcDistance_tmp[:,4] = (data[:,4] - query[4]) / (np.max(data[:,4], axis=0) - np.min(data[:,4], axis=0))
calcDistance_tmp = np.sum(calcDistance_tmp**2, axis=1)
euclDistance = np.sqrt(calcDistance_tmp)
```

At the end, the function returns 2 arrays:

_ One 1D numpy array "euclDistance" which corresponds to the mix distance (Euclidean and Hamming) between the test point and each training point (for each euclDistance index)

_ Another array "index_sort" which is the 3 nearest point indexes of the data training. As per specification, we select only the 3 nearest neighbors' points (k=3).

3) calculateDistanceskp:

This function is quite similar to the "calculateDistances" function; but it has 4 arguments "data", "query", "k" and "p". "data" and "query" are identical to those presented in "calculateDistances". "k" is an integer which corresponds to the number of nearest neighbors to consider; whereas "p" is a float which parameterizes the Minkowski distance.
The implementation is globally the same as "calculateDistances" except that all Euclidean distances are replaced by the Minkowski distance and at the end the "index_sort" variable does not return 3 neighbors; but k neighbors. The function returns 2 arrays "Distance" and "index_sort" in the same way as the "calculateDistances" function.

4) **queryResult:**

The function has 2 arguments “dataResult” and “index_sort”. the “dataResult” argument corresponds to a 1D numpy array extracted from the training diagnosis result data (last column of the training csv file) and the “index_sort” argument is a 1D numpy array which is the list of the k nearest neighbor’s indexes.

Then, the mean of the diagnosis result is calculated (sum of the diagnosis result, in the code, “dataResultSum” divided by the number of the k nearest neighbors, in the code, “len(index_sort)”).

At the end, an integer number “qResult” is returned. The value can be 1 (malignant) or 0 (benign). “qResult” is the prediction result based on the k-NN algorithm.

5) **queryDistWeightVoteResult:**

This function is a different way to predict the result of a test point, it replaces the “queryResult” function; but it is based on the weight distance voting system rather than the number of the k nearest neighbors. It has 3 arguments. “dataResult” and “index_sort” are identical to those presented in “queryResult”. The argument “Distance” is a 1D numpy array which contains the distance values between the test point and each training point.

The idea of the function is to sum the invert values of the distance for each category (malignant and benign). The invert values of the distance is obtained thanks to the numpy command “np.reciprocal”. In the code:

- *DistWeight = np.reciprocal(Distance[index_sort])*

Then, the 2 categories of the diagnosis data result of the nearest neighbors (benign or malignant) are identified. In the code, this result is converted into Booleans values:

- *dataResultBool = np.array(dataResult[index_sort], dtype=bool)*

Then the invert values of each category are summed. In the code:

- *dataResultMalignantSum = np.sum(DistWeight[dataResultBool])*
- *dataResultBeningnSum = np.sum(DistWeight[np.invert(dataResultBool)])*

At the end, a voting system is done: the category with the highest value is the query result “qResult” to be returned (1, for malignant and 0 for benign).

6) **AccuracyResult:**

The function has 2 arguments “testDataResult” and “testDataPrediction”. the “testDataResult” argument corresponds to a 1D numpy array extracted from the test diagnosis result data (last column of the test csv file) and the “testDataPrediction” argument is a list of the predictive result for each test point.

The idea of this function is to rate the accuracy of the predictive result.

For that, the number of correct predictions is counted and then divided by the number of the test points. In the code:

- *(correct/float(len(testDataResult))) * 100.0*

Result of Part 1:

The KNN Algorithm implemented in Python according to the above specification has shown the following result:

Acuracy of k=3, Euclidean distance, non-weighted dist: 77.34375

Acuracy of k=3, Euclidean distance, weighted dist : 75.0

The KNN Algorithm has achieved a better result (77.34%) with a non-weighted distance predictive selection in case of 3 nearest neighbors (k=3), and Euclidean distance.

Part 2 – Investigating k-NN variants and hyper-parameters:

In this part, we are going to see the impact of the variants and hyper parameters on the k-NN performance.

The 2 main hyper-parameters which can be used for a k-NN Algorithm is:

- The k parameter which corresponds to the number of the nearest neighbors to consider for predicting the diagnosis result of a test point
- The p (Mdist) parameter of the Minkowski distance calculation:

$$\left(\sum_{i=1}^{n} |X_i - Y_i|^p\right)^{1/p}$$

This parameter p has an impact on the distance calculation between the test point and the training points. With p=1, we have the same formula as the Manhatan distance and if p=2, we have the formula of the Euclidean distance.

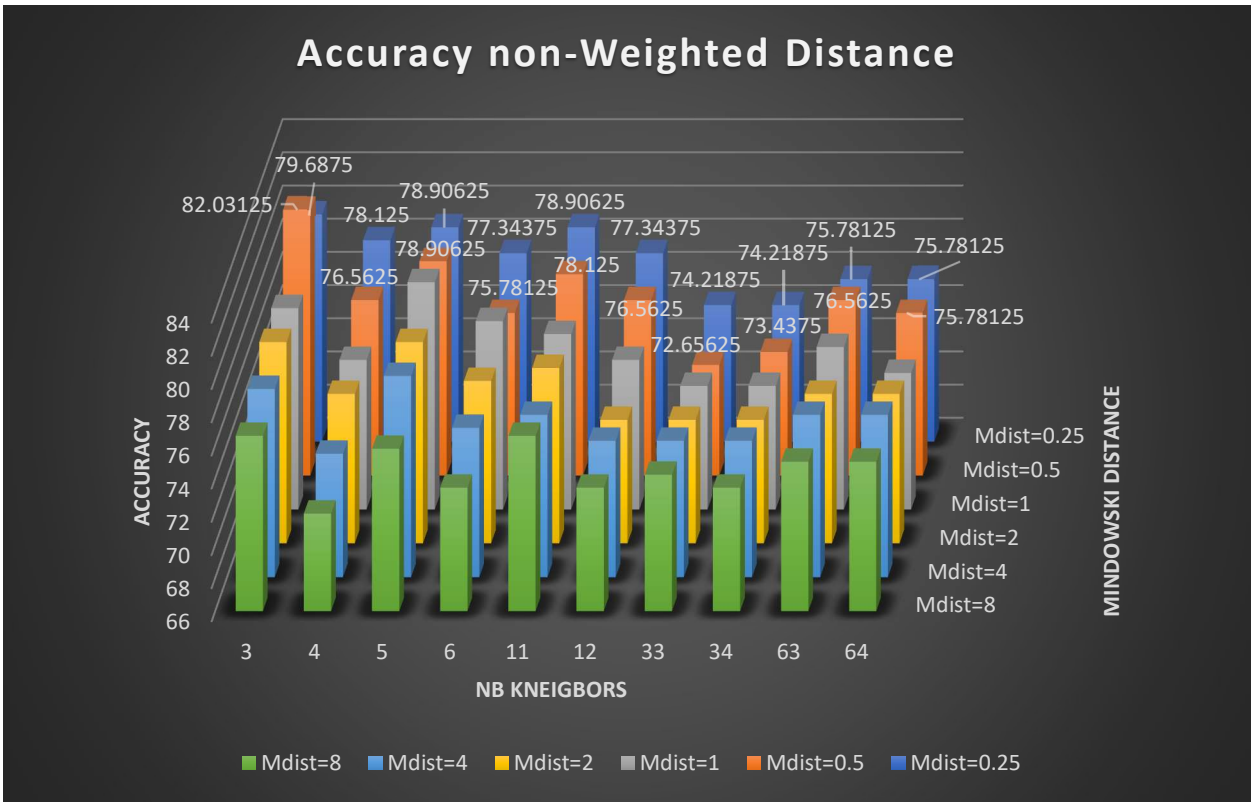
On the other side, the 2 main variants to be identified for the k-NN Algorithm are:

- The distance weighting: By applying distance weight or not, the prediction of the test point result may change.
- The distance normalization: for this, the distance implemented in the code is already normalized since the scale of the Age is high compared to the others scale (see Part 1 -2 calculateDistances). Therefore, this normalization process will not be treated in this part.

In this part, different combination of the hyper-parameters and variant have been tested:

- For k, the following values has been chosen: 3, 4, 5, 6, 11, 12, 33, 34, 63, 64
- For p, the following values has been selected: 0.25, 0.5, 1, 2, 4, 8
- And then a study on the impact of the weighting distance or not has been done for each combination.

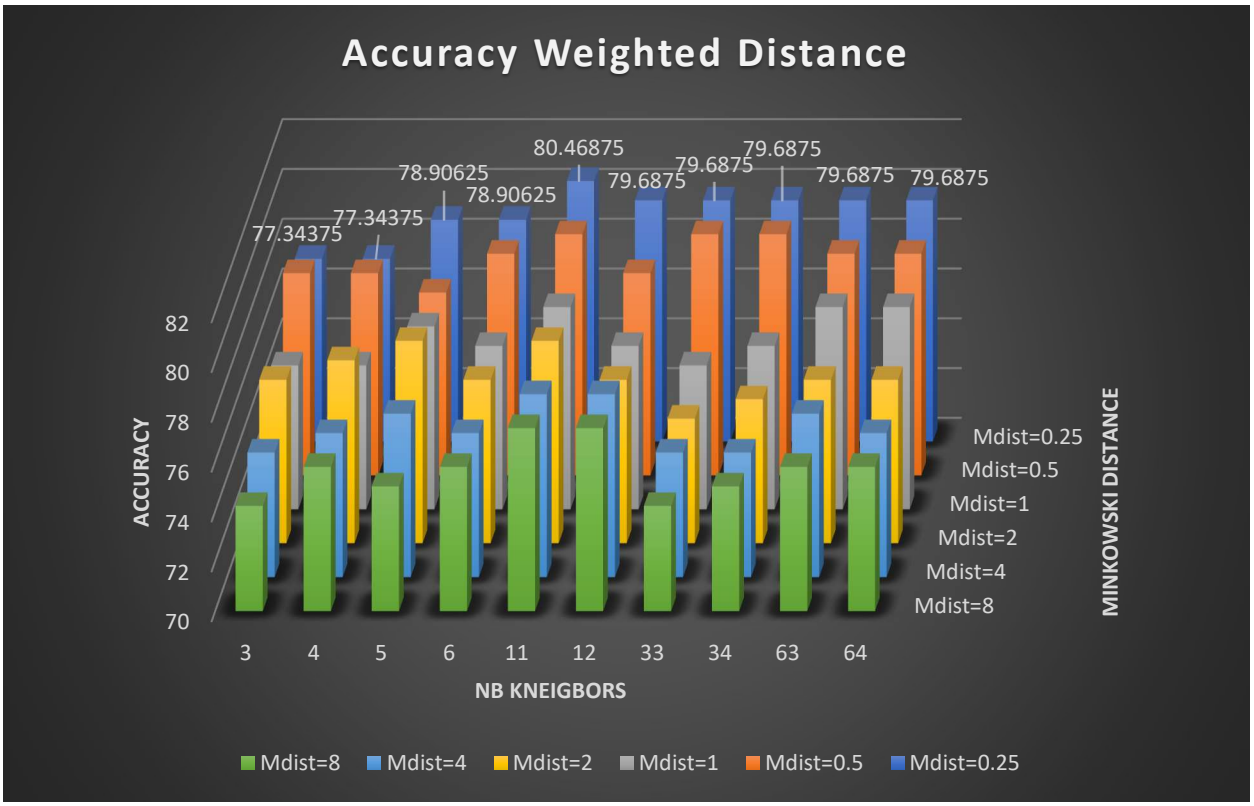
You can find below a 3D graph of the different combination which have been done for the case of a non-Weighted distance (X-axis = nb k neighbors, Y-axis = p (Mdist) parameter of Minkowski distance, and Z-axis = Accuracy):



In the case of the non-Weighted distance, we can notice the following points:

- The highest accuracy is reached when the parameter k is low. Indeed, the more we add the neighbors, the more risk, we get to include neighbors with the opposite diagnosis value.
- When k is even, the rate is also low. This is due to the fact that with even numbers the risk that half of the nearest point could be at one category and the other half at the other category.
- We observe a better result when the parameter p (Mdist) from the Minkowski distance is low.
- The best accuracy is reached with a value of 82.03% when k=3 (nb of neighbors) and Mdist=0.5

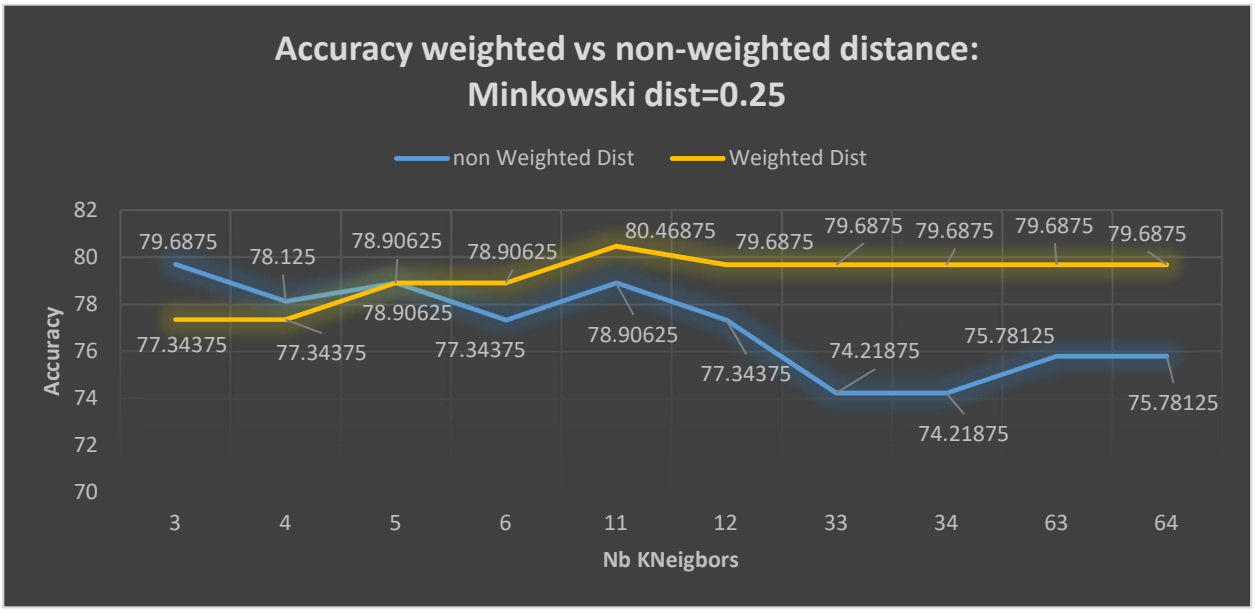
Now let's consider the 3D graph of the different combination for a Weighted distance (X-axis = nb k neighbors, Y-axis = p (Mdist) parameter of Minkowski distance, and Z-axis = Accuracy):



In the case of the Weighted distance, we can notice the following points:

- The highest accuracy is reached when the parameter k is relatively high. The peak is reached when k=11. Then, the accuracy is relatively stable when k > 11. Indeed, Contrary to the non-Weighted distance, the weighted distance gives less ponderation for neighbors which are far. Therefore, adding more k neighbors does not really much impact on the accuracy.
- We observe a better result when the parameter p (Mdist) from the Minkowski distance is low.
- The best accuracy is reached with a value of 80.47% when k=11 (nb of neighbors) and Mdist=0.25

You can find below a comparison between the weighted distance and the non-Weighted distance for p (Mdist)=0.25:



As mentioned previously, we have better result of accuracy when k is low for a non-weighted distance; whereas, for weighted distance, we have a better result when k is high.

However, between the weighted and the non-weighted distance, the best value of accuracy is observed for:

_ non-Weighted distance, with k=3 and Mdist=0.5 (not shown in the above graph) with an accuracy of 82.03%

Part 3 – Developing k-NN for Regression Problems:

In this last part, we will focus on the usage of the kNN Algorithm for a regression problem.

For that, the training and test dataset from regressionData.zip compressed folder is used.

In this section, the functions “calculateDistanceskp”, “queryDistWeightVoteResult” and “AccuracyResult”, presented in Part 1 and used in Part 2, have been modified, and all the functions have been saved in a new file” KNN_assignment_part3.py”.

- The function “calculateDistanceskp” does not use anymore the Hamming distance calculation since all the features in the new training database are float numbers (non-categorical features). Each distance is also normalized by using the min and max values of each column features.
- The function “queryDistWeightVoteResult” has now a new parameter n which increases the weighting distance.
- The function “AccuracyResult” does not count anymore the number of good results. Instead, it integrates the concept of the R squared values:

  $R2=1-SSR/TSS$

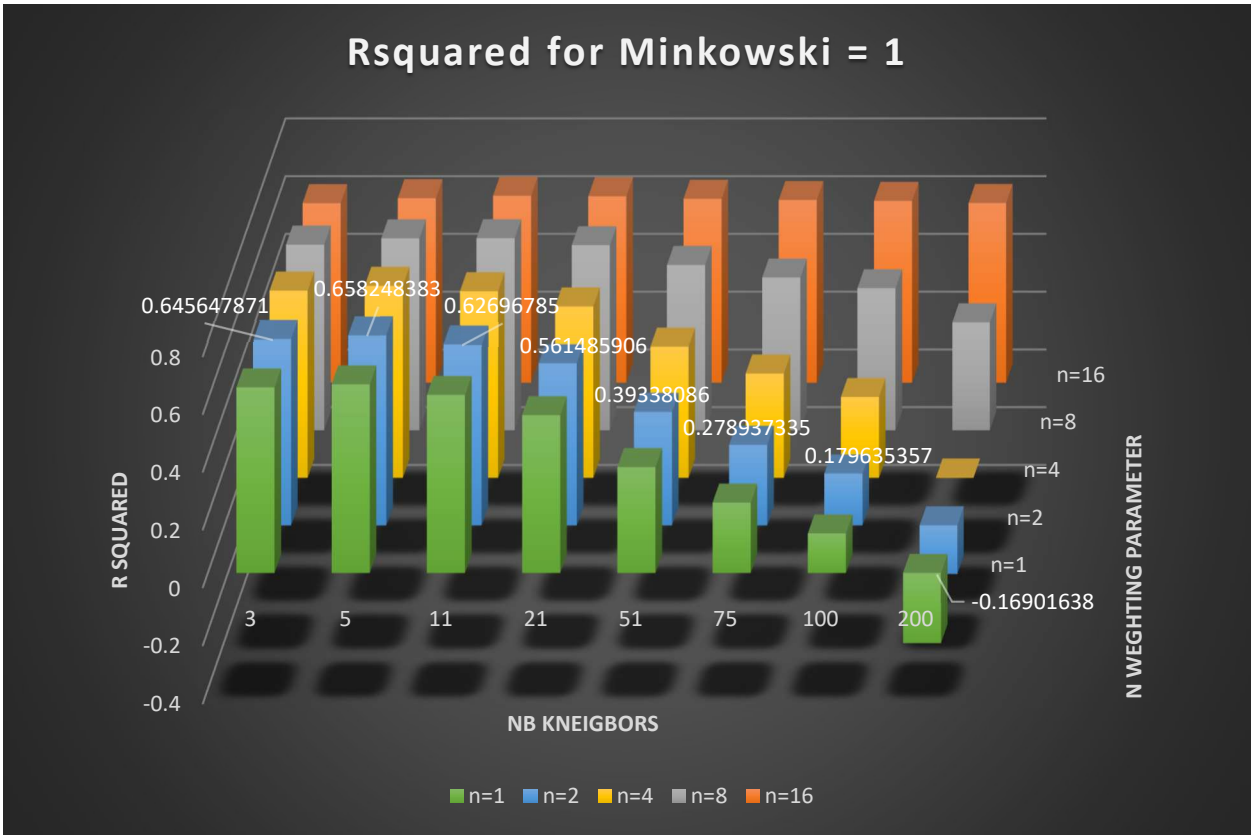
  with, $SSR=\sum i(f(Xi)-Yi)$

  and $TSS=\sum i(Y-Yi)$

Like in Part 2, different combination of the hyper-parameters and variant have been tested:

- For k, the following values has been chosen: 3, 5, 11, 21, 51, 75, 100, 200
- For p (Mdist), the following values has been selected: 1, 2, 4
- For n (weighting distance parameter), the considered values are: 1, 2, 4, 8, 16

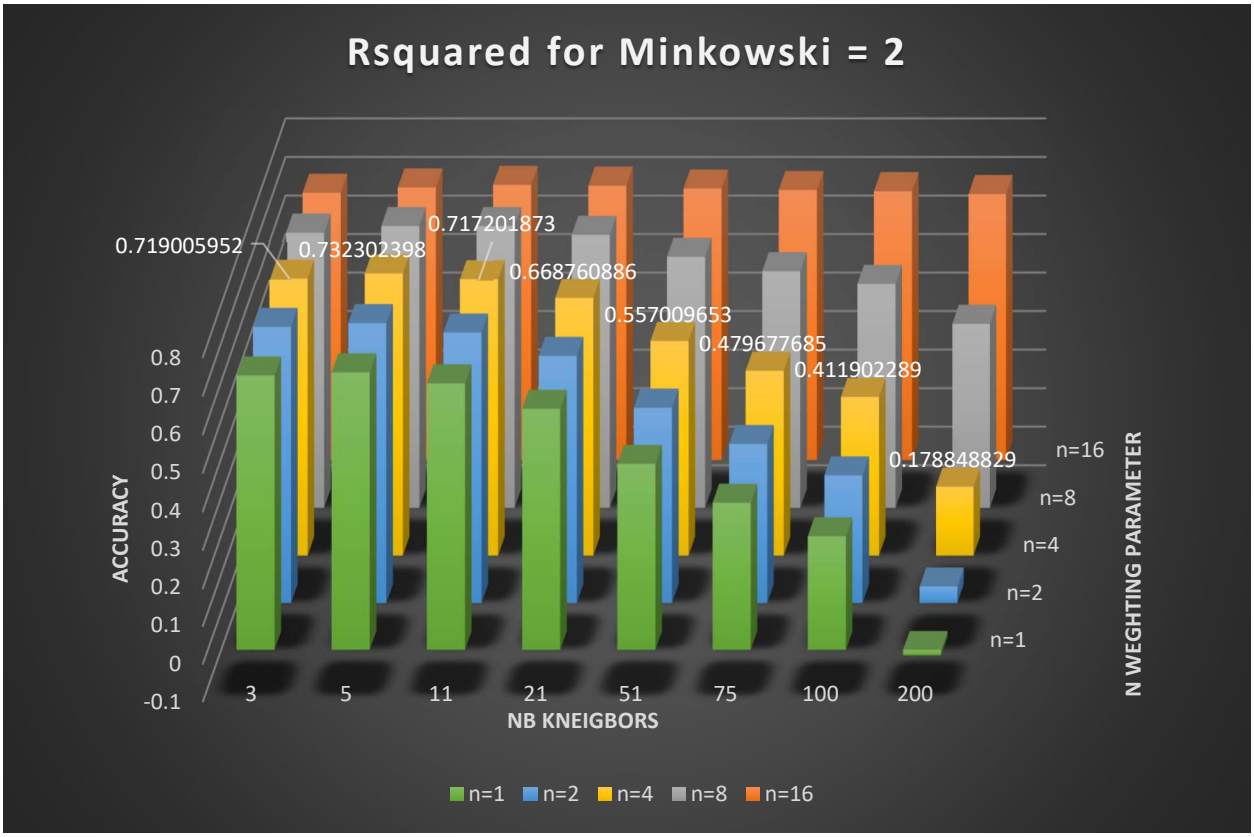
You can find below a 3D graph of the different combination which have been done for the case of a Minkowski distance of 1 (X-axis = nb k neighbors, Y-axis = n, weighting distance parameter, and Z-axis = R squared):



In the case of a Minkowski distance of 1, we can notice the following points:

- When n (weighting parameter distance) is low, a high value of k (nb of nearest neighbors) will lead to a very low R Squared value. This is due to the fact that the farer neighbors have still an influence on the test point prediction accuracy. In order to avoid, the weight of the farer neighbors, we should increase the weight of the distance parameter n.
- R Squared value here is maximal when k=5 and n=2 with a value of 0.658.

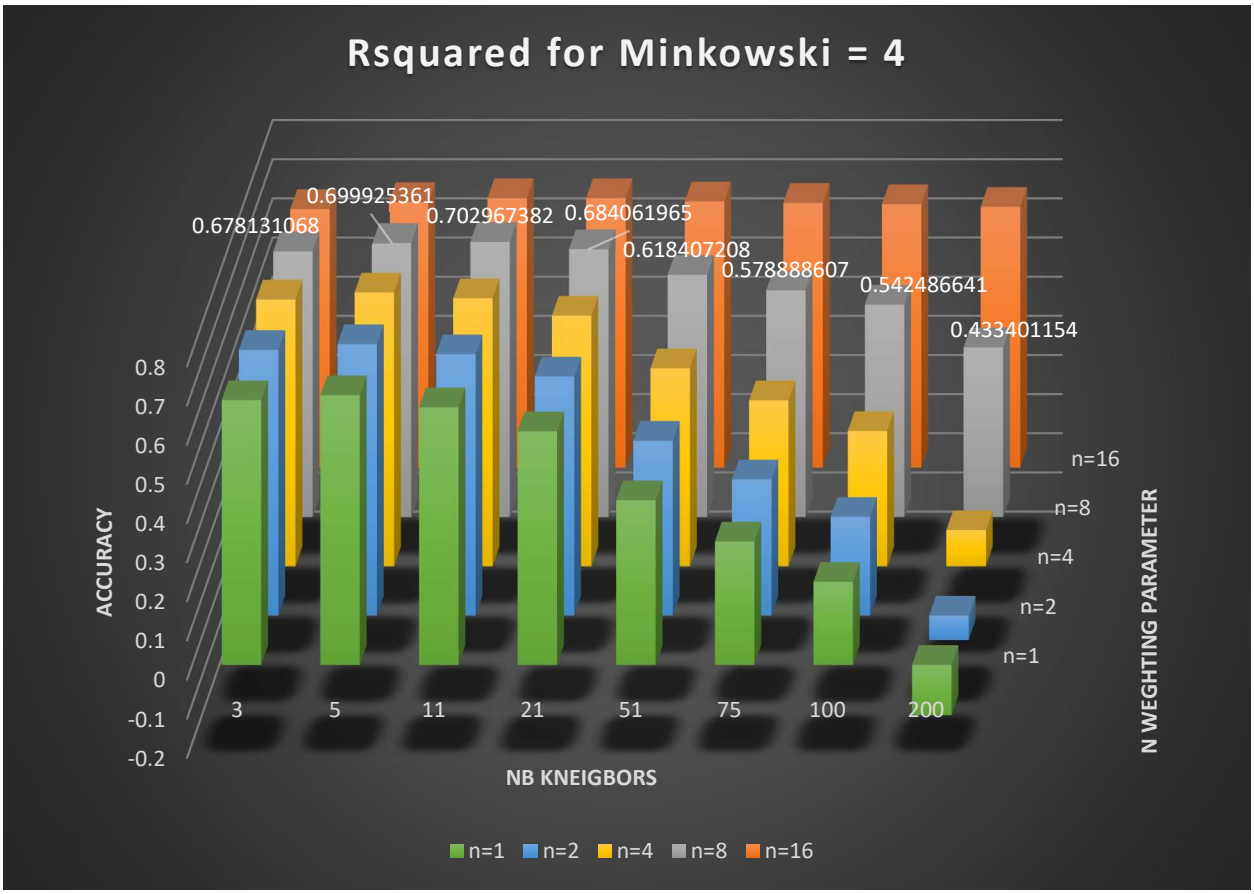
Now let's consider the 3D graph of the different combination with a Minkowski distance of 2 (Euclidean distance) (X-axis = nb k neighbors, Y-axis = n, weighting distance parameter, and Z-axis = R squared):



In the case of a Minkowski distance of 2, The same observation can be done for a low n, weighting distance parameter.

The R Squared value here is maximal when k=5 and n=4 with a value of 0.732.

Finally, we can see the 3D graph of the different combination with a Minkowski distance of 4 (X-axis = nb k neighbors, Y-axis = n, weighting distance parameter, and Z-axis = R squared):



The R Squared value here is maximal when k=11 and n=8 with a value of 0.703.

In conclusion, the maximum R Sqaured value has been observed with a value of 0.732 for an Euclidean distance (Minkowski = 2), with a number of nearest neighbor of k=5 and a weighting distance parameter of n=4. However, this study is missing the evaluation of the R squared when both parameters n and k are high. Indeed, the number selected for both parameters are quite low for a training database of more than 6000 points. It may be possible that we get a better Rsquared value when bigger value of k and n is considered.