UNIVERSITA' DEGLI STUDI DELL'INSUBRIA

DIPARTIMENTO DI SCIENZE TEORICHE E APPLICATE (DiSTA)

CORSO DI LAUREA MAGISTRALE IN INFORMATICA

INTELLIGENT SYSTEM PROJECT

RELAZIONE DI:
Vladi VALSECCHI matr 730030
Gianluca MOLTENI matr 730113

ANNO ACCADEMICO 2020/2021

Summary

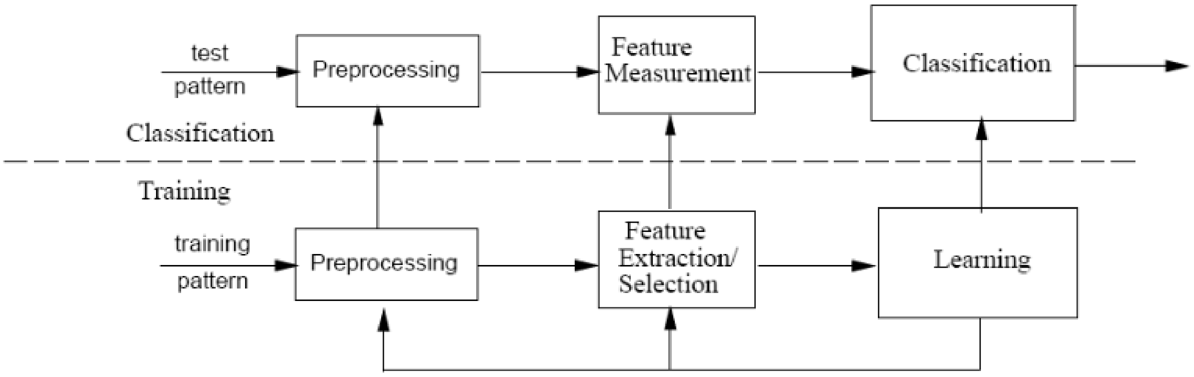
Introduction

In this project we will proceed to the classification of two datasets, using two classification models and varying their metrics. We will then carry out an analysis of the results obtained for the single algorithm and an overall analysis of the performances.
Our goal is to understand which classification algorithm performs better on the inputs considered. The project includes a main file and a function file made on Jupyter Notebook in Python language (.ipynb format) using a Python 3.6 environment generated in Anaconda in which we implemented methods for choosing the features to be used for the classification and subsequently the actual classification.

Part A

What are supervised methods?

Supervised learning is a type of machine learning. Its goal is to classify data based on samples provided for reference. Each sample is associated with a label (represents the class). Consequently, the expression <pattern-class label> is used to indicate the association.
The characteristic of supervised methods is the separation of the data provided in input into two subsets called training set and test set. The first is the set used for learning and creating the model, the second is used for validating the model.



Classification and training procedure

A mapping function is used to associate the input data with the most suitable class.
Supervised methods can be divided in two categories:

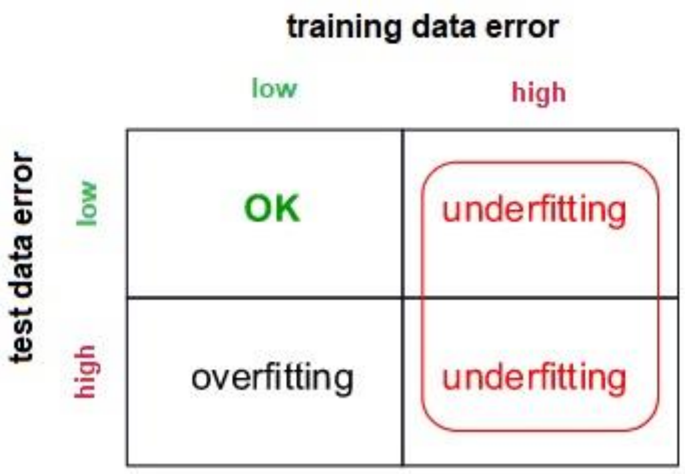
- Classification: is the process of prediction of the class based on the provided data. Classes are called targets/labels or categories;
- Regression: provides us with a simple linear relationship of two or more variables within a dataset.

The classification must be performed with a consistent set of data, this is because the performance varies according to the goodness of the training data provided in input. Another factor affecting performance is the amount of data provided. In this case, may happen a situation of:

- Overfitting;
- Underfitting.

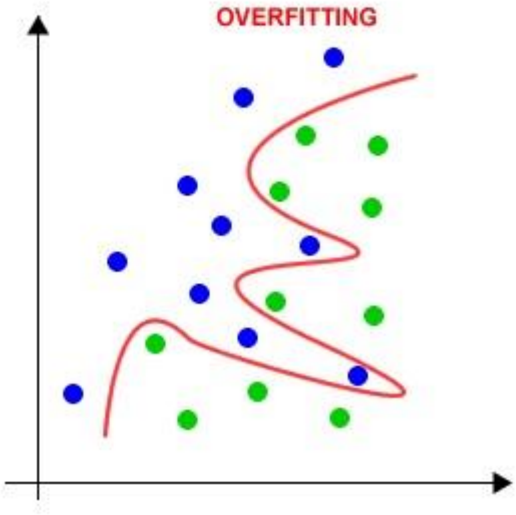
Problems with supervised methods

Depending on the amount of data provided, supervised methods can present situations of overfitting and underfitting.



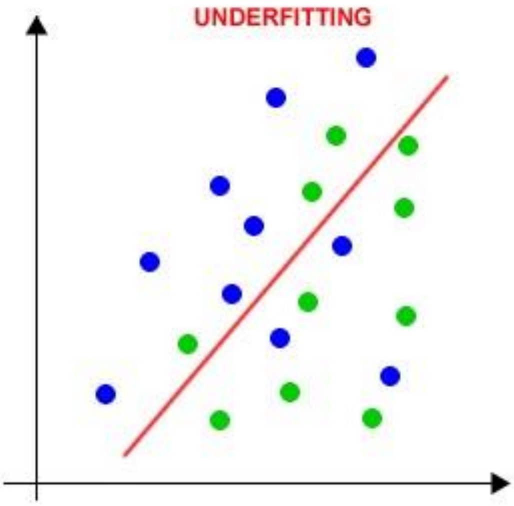
test data error \ training data error	low	high
low	OK	underfitting
high	overfitting	underfitting

Overfitting is a situation that arises when too many parameters are provided during the training phase, in which the classifier separates the data provided with high precision but fails in the generalization process (eg. when new, never seen, models are classified).



Overfitting example

Underfitting is a situation that arises when few parameters are provided during the training phase, in which the classifiers suffer from an excessive discrepancy.



Underfitting example

Differences with unsupervised methods

Unsupervised methods don’t have a separation phase. These methods don’t have a set of correct answers unlike the supervised ones, which have a set of labels assigned to the patterns, but processing the single data there is a possible direct classification done using clustering because the class assignment is carried out by similarity.

Part B

Introduction

For the realization of the project, we relied on two supervised classification models.
With classification we mean the processes of prediction of data classes of points. The classes are also called target/labels or categories. Classification models require a set of input variables (X) and a mapping function (f) which associate each input with an output. The result (Y) is a discrete variable. Given an input $x \in X$ and W classes $\omega_1,...,\omega_W$ we must find W decision functions $d_1(x),...,d_W(x)$ with the property that a given x belongs to the class ω_I iff $d_i(x)>d_j(x)$ with $j=1,...,W$ and $j \neq i$. The function f is also called the discriminant function and it helps us to understand where two classes are separated. Such a decision boundary between two classes (ω_i, ω_j) is given by the value of x for which $d_i(x)=d_j(x)$.
The difference between two classes is given by:

$d_{ij}(x) = d_i(x) - d_j(x) = 0$

$d_{ij}(x) > 0 \rightarrow$ x belongs to ω_i

$d_{ij}(x) < 0 \rightarrow$ x belongs to ω_j

The supervised classifiers chosen are Minimum Distance Classifier (MDC) and K-Nearest Neighbor (KNN):

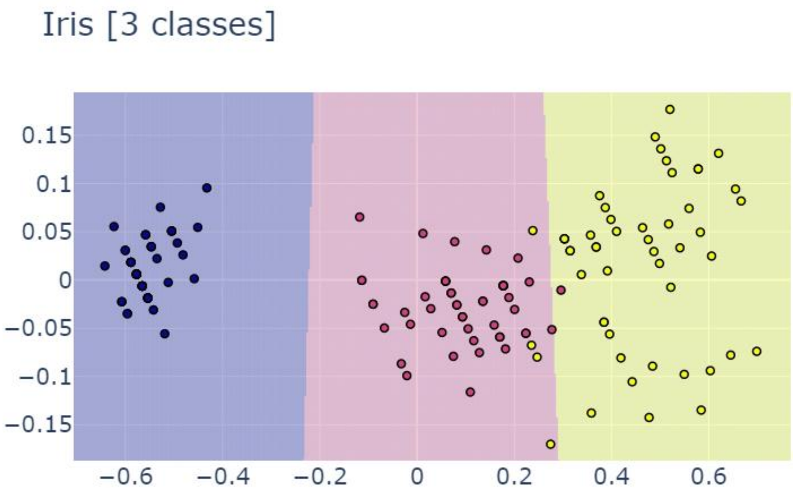
Minimum Distance Classifier (MDC)

The Minimum Distance Classifier or MDC is a supervised classification algorithm that associates a given x to the class based on the minimum distance. The classifier makes use of Minkowski distances:

$$d(x,y)=(\Sigma_{i=1,N}(x_i-y_i)^{\lambda})^{1/\lambda}$$

where λ is an integer

- λ=2: Euclidean distance;
- λ=1: Cityblock/Manhattan distance;
- $\lambda \rightarrow \infty$: Chessboard;



Example of classification using MDC

How it works

1. For each class is computed the prototype m_j;
2. The pattern x is compared with the prototype m_j;
3. The pattern x is assigned to the class with the shortest prototype-pattern distance.

The discriminating function is based on the minimum distance. In the case of using the Euclidean distance we have

$$D_j(x)=||x-m_j||=[(x-m_j)^T(x-m_j)]^{½}$$

Since D_j is a non-negative quantity, we can consider D^2_j equivalent in the comparison for class assignment

$$D^2{}_j=(x-m_j)^T(x-m_j)=x^Tx-2[x^Tm_j-½(m_j{}^Tm_j)]$$

Since the first term depends only on the pattern and not on the class, we can simplify it so that

$$d_j=x^Tm_j-½(m_j{}^Tm_j)$$

This is our discriminating function for each class and will be applied on the pattern x to find the class it belongs to.
A pattern x belongs to the j-th class if:

$$d_j(x)\geq d_i \ i=1,...,W \ i\neq j$$

The decision frontier is:

$$d_{ij}(x) = d_i(x) - d_j(x) = x^T(m_i-m_j)-½ \ (m_i{}^Tm_i-m_j{}^Tm_j)$$

The Minimum Distance Classifier decision frontier is the perpendicular bisector of the line segment joining m_i and m_j.

Advantages and disadvantages

Advantages:

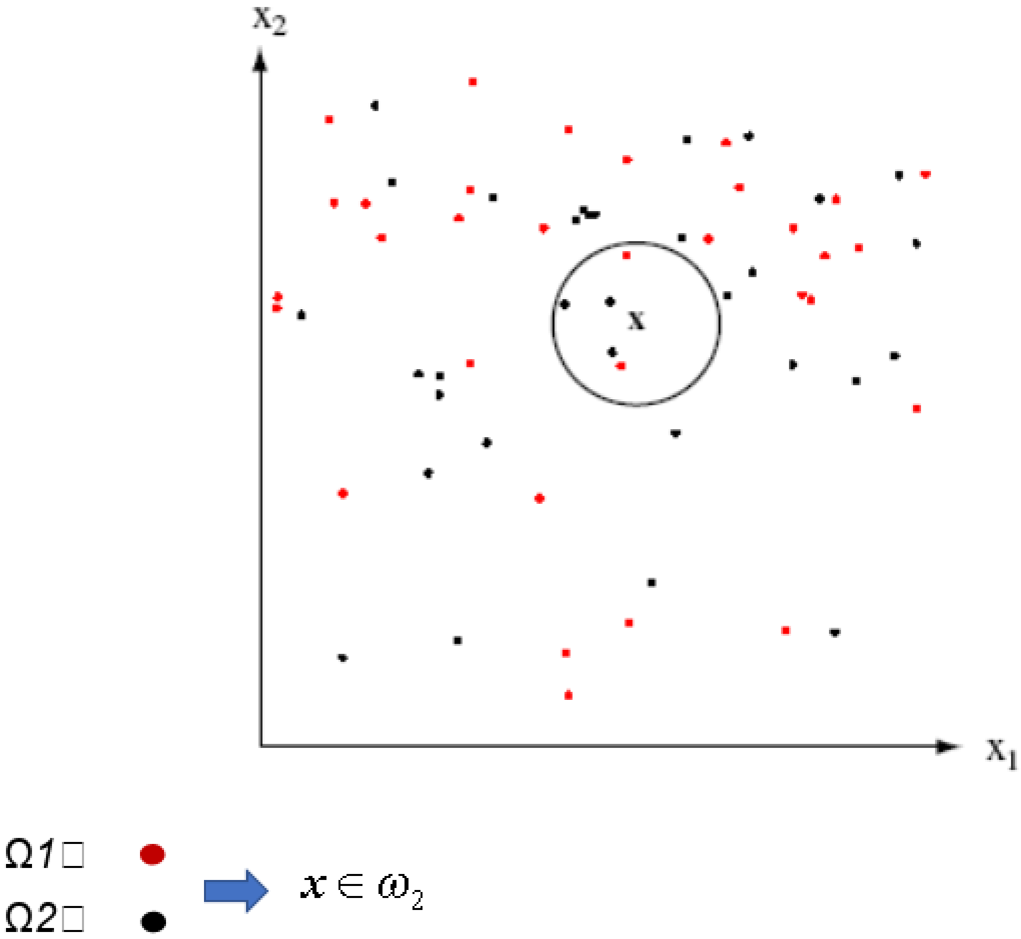
- Low demand for training data
- Heuristic rule, minimum number of training samples = 10*N con N=number of features;
- Easy to design;
- Low computational complexity.

Disadvantages:

- Statistical information such as a-priori probability and variance are ignored;
- In practice the Minimum Distance Classifier works well when the distance between means is large compared to the spread of each class with respect to its means.

K-Nearest Neighbor (KNN)

The K Nearest Neighbor or KNN is a supervised classification algorithm that associates a given x to the class based on the quantity of elements of a given class in a given range. KNN is also based on Minkowski distances and the most used for this algorithm is the Euclidean distance. The classifier does not require a separate training phase and does not make statistical assumptions on the distribution of classes. KNN is a lazy learner because it does not learn a discriminative function from the training data but memorizes the training dataset instead. For example, the logistic regression algorithm learns the model weights during training time. In contrast, there is no training time in KNN.



Example of classification using KNN

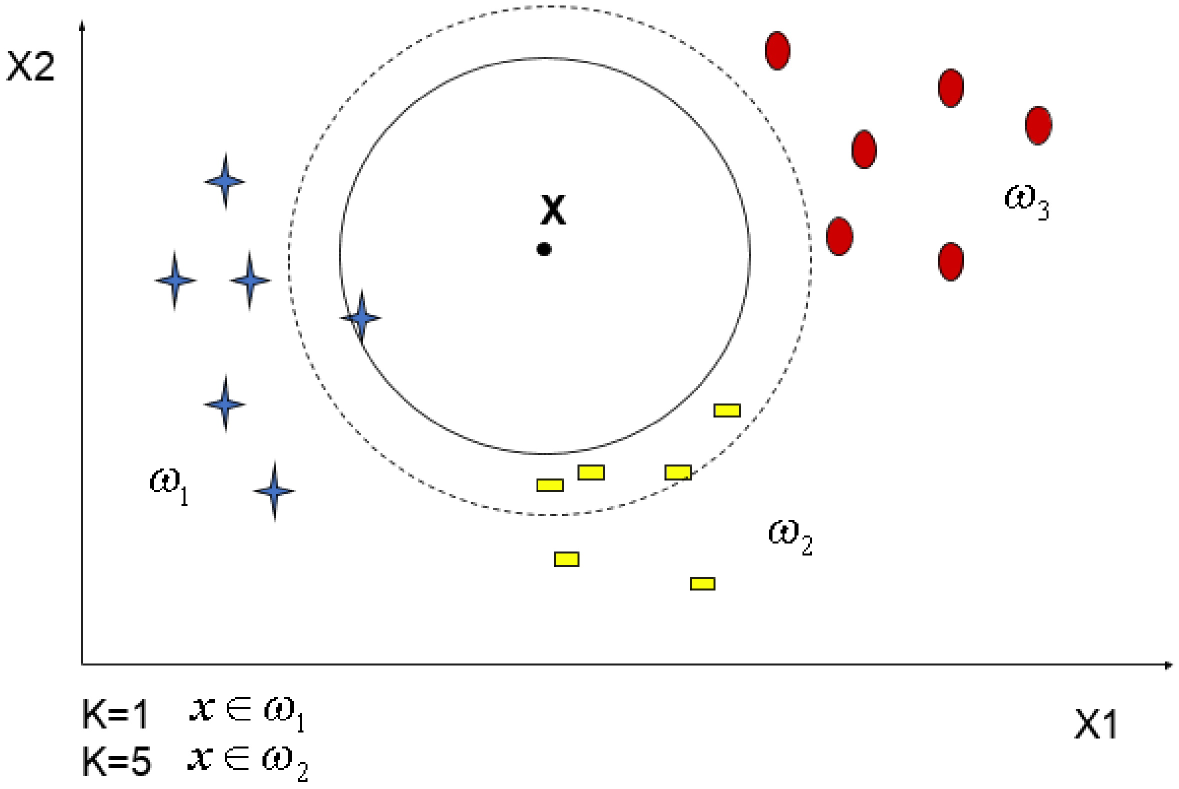
How it works

1. Given W classes $\omega_1,...,\omega_W$ and N supervised training samples x1,...,xN;
2. To classify a pattern x, the distance is computed (using a predefined metrics) between the pattern x and all the training samples;
3. Let Ki be the number of samples inside the sphere and belonging to ω_i (i = 1, ..., W), the classifier assigns the pattern x to the class ω_i such that Ki / K is maximum.

The Euclidean distance can also be used by changing step 2:

- To classify the x pattern, the x-centered sphere is expanded to include K samples (nearest samples).

The value of K affects the result of the classification.



Classification with different K values

The choice of K is crucial:

- If k is too small, there is a risk of overfitting and the class assignment is altered by the noise within the data;
- If k is too large, class assignment may be affected by samples that are quite different from the pattern to be classified.

Advantages and disadvantages

Advantages:

- Easy to design and understand;
- The classifier takes no time to train.

Disadvantages:

- We pay computational cost at test time, since classifying a test example requires a comparison to every single training example.

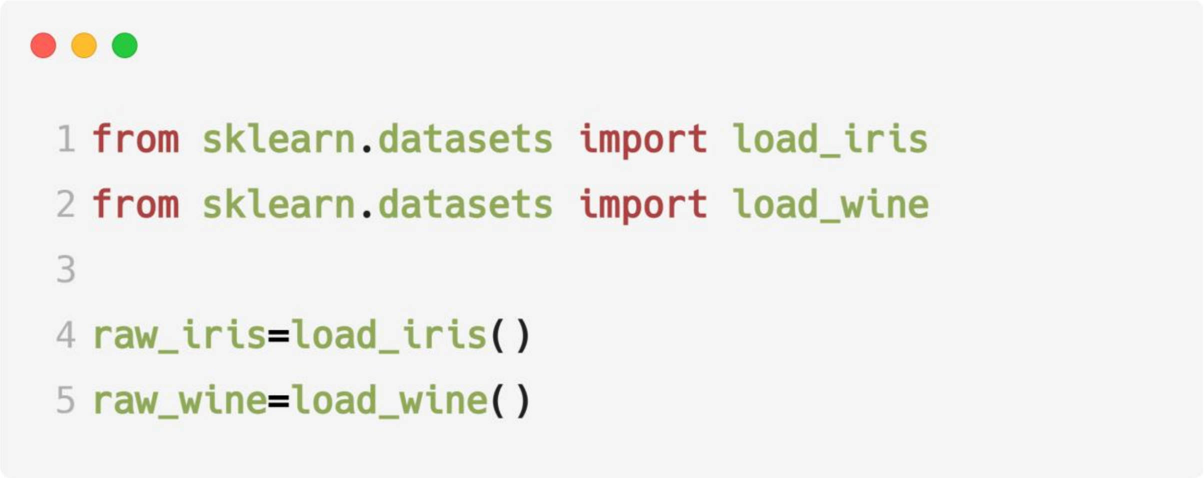
Part C

Used datasets

The datasets used for the experiment are:

- Iris;
- Wine.

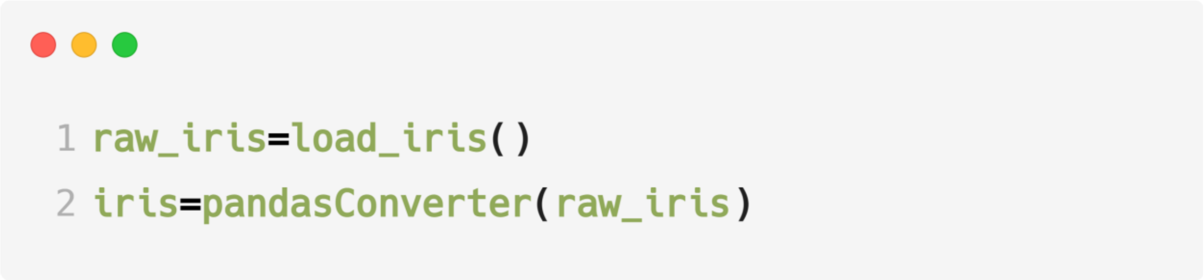
Both datasets come from the *sklearn.datasets* library



```python
from sklearn.datasets import load_iris
from sklearn.datasets import load_wine

raw_iris=load_iris()
raw_wine=load_wine()
```

Datasets import and loading

The newly imported datasets are in the form of an array and were subsequently converted into dataframes using the *pandas* library, in order to better manage the data.



```python
raw_iris=load_iris()
iris=pandasConverter(raw_iris)
```

Operation of dataset loading and conversion into pandas dataframe

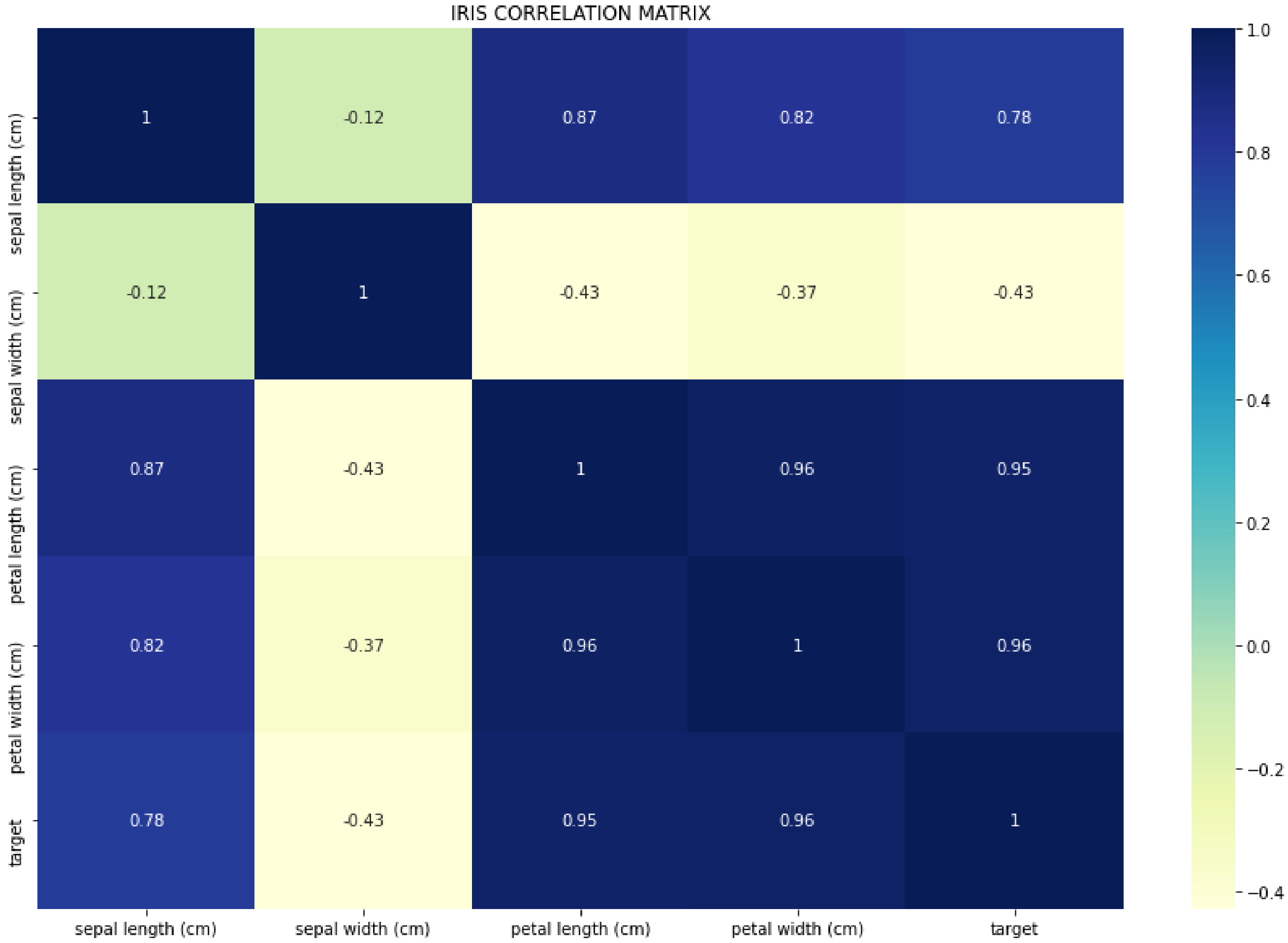
Iris

The dataset presents 150 instances divided into 3 classes Iris-Setosa, Iris-Versicolor and Iris-Virginica. Each class comprises 50 instances. The attributes that characterize the dataset are 4:

- sepal length in cm;
- sepal width in cm;
- petal length in cm;
- petal width in cm.

The data presents the following characteristics:

Feature	**Min**	**Max**	**Mean**	**SD**
sepal length	4.3	7.9	5.84	0.83
sepal width	2.0	4.4	3.05	0.43
petal length	1.0	6.9	3.76	1.76
petal width	0.1	2.5	1.20	0.76



Iris correlation matrix

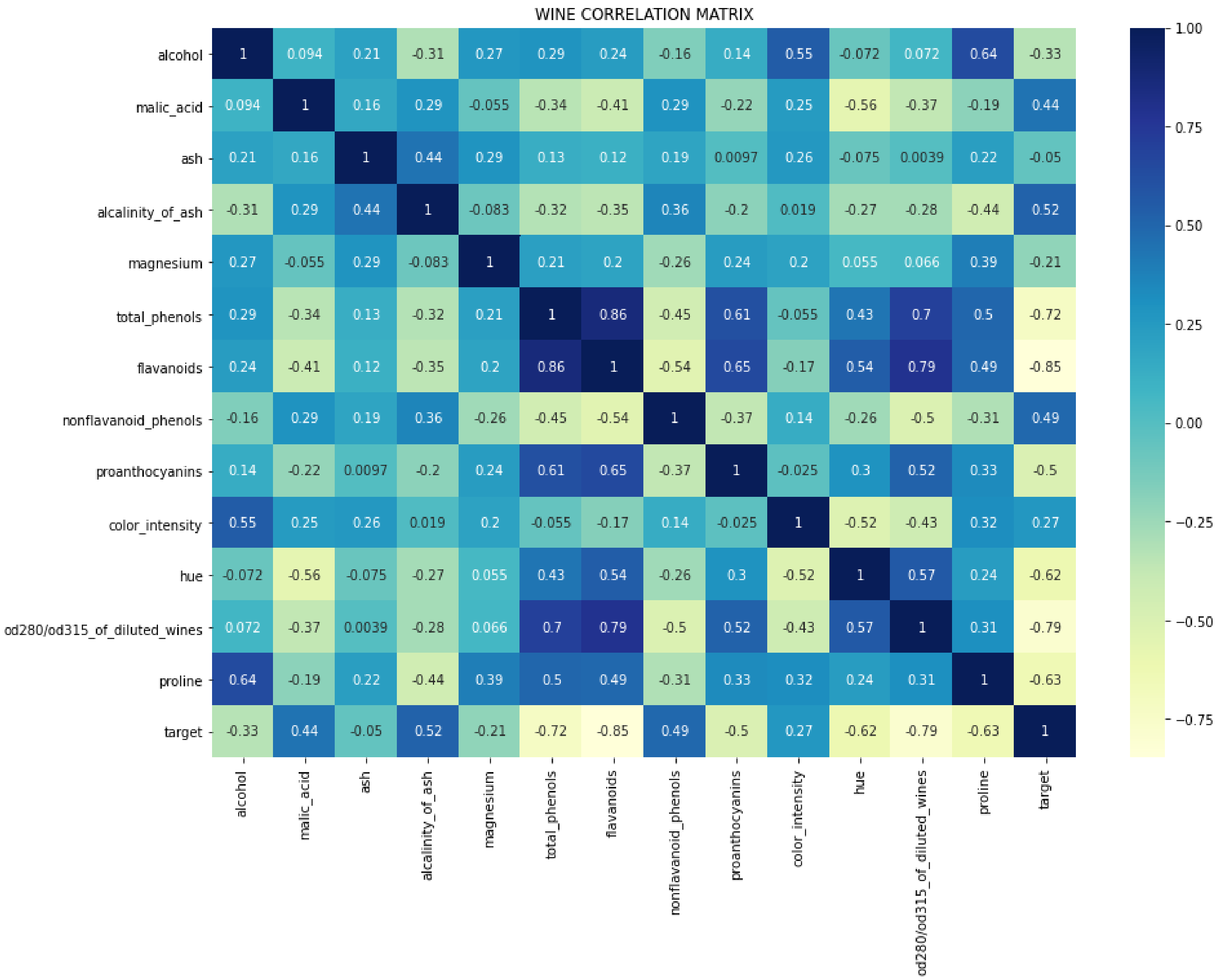
Wine

The dataset presents 178 instances divided into 3 classes class-0, class-1 and class-2. The classes contain 59, 71 and 48 instances respectively. The attributes that characterize the dataset are 13:

- alcohol;
- malic_acid;
- ash;
- alcalinity_of_ash;
- magnesium;
- total_phenols;
- flavonoids;
- nonflavonoid_phenols;
- proanthocyanis;
- color_intensity;
- hue;
- od280/od315_of_diluited_wines;
- proline

The data presents the following characteristics:

Feature	**Min**	**Max**	**Mean**	**SD**
alcohol	11.0	14.8	13.0	0.8
malic_acid	0.74	5.80	2.34	1.12
ash	1.36	3.23	2.36	0.27
alcalinity_of_ash	1.06	30.0	19.5	3.3
magnesium	70.0	162.0	99.7	14.3
total_phenols	0.98	3.88	2.29	0.63
flavonoids	0.34	5.08	2.03	1.00
nonflavonoid_phenols	0.13	0.66	0.36	0.12
proanthocyanis	0.41	3.58	1.59	0.57
color_intensity	1.3	13.0	5.1	2.3
hue	0.48	1.71	0.96	0.23
od280/od315_of_diluited_wines	1.27	4.00	2.61	0.71
proline	278	1680	746	315



Wine correlation matrix

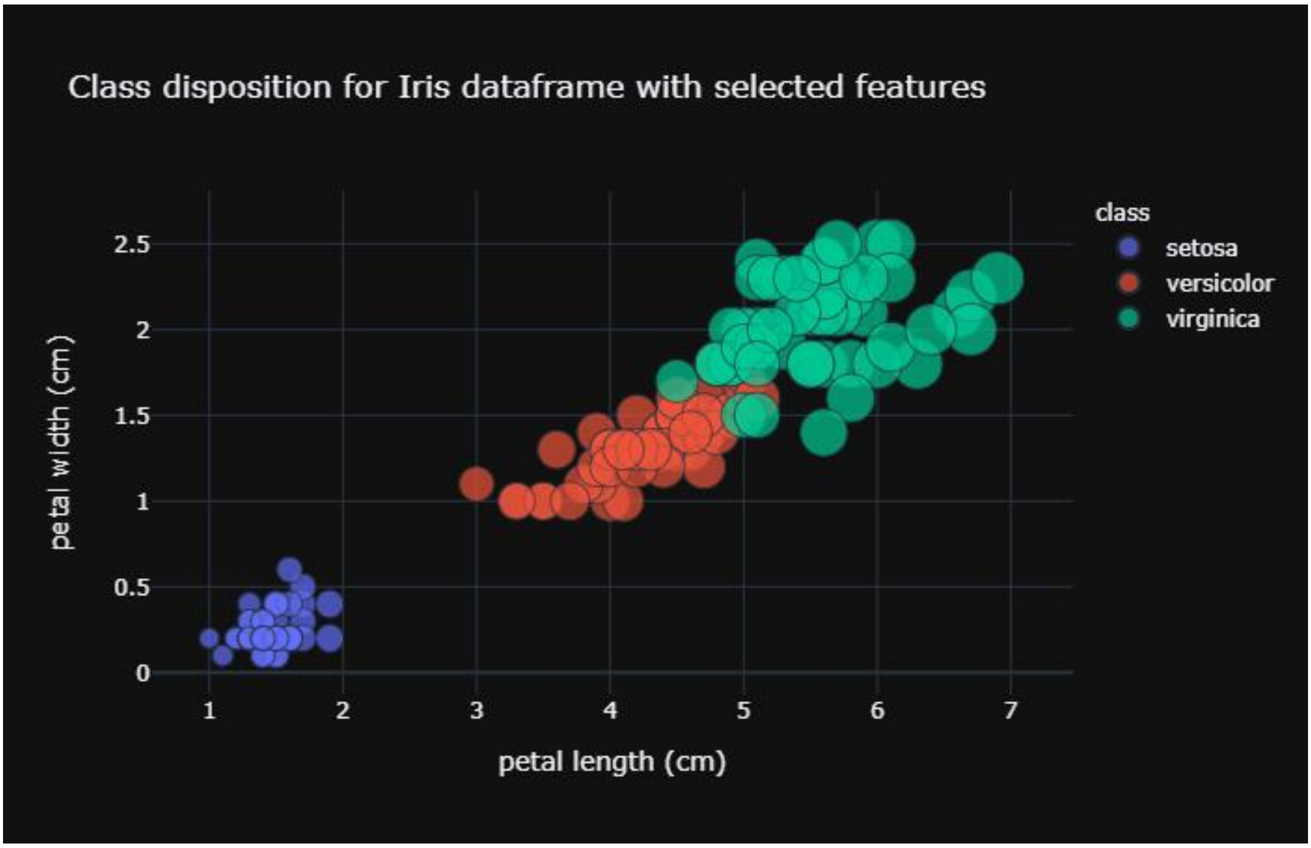
Feature selection

For the feature selection operation we relied on the SelectKBest feature selector, coming from the *sklearn.feature_selection.SelectKBest* library.The selector uses chi2, so it computes chi-squared stats between each non-negative feature and class and it is used to select the n_features (n_features=2) features with the highest values for the test chi-squared statistic.

KBest Iris

Feature	**Score**	**Pval**
sepal length	10.817821	4.476515e-03
sepal width	3.710728	1.563960e-01
petal length	116.312613	5.533972e-26
petal width	67.048360	2.758250e-15

KBest features for Iris based on score: **petal length, petal width**.

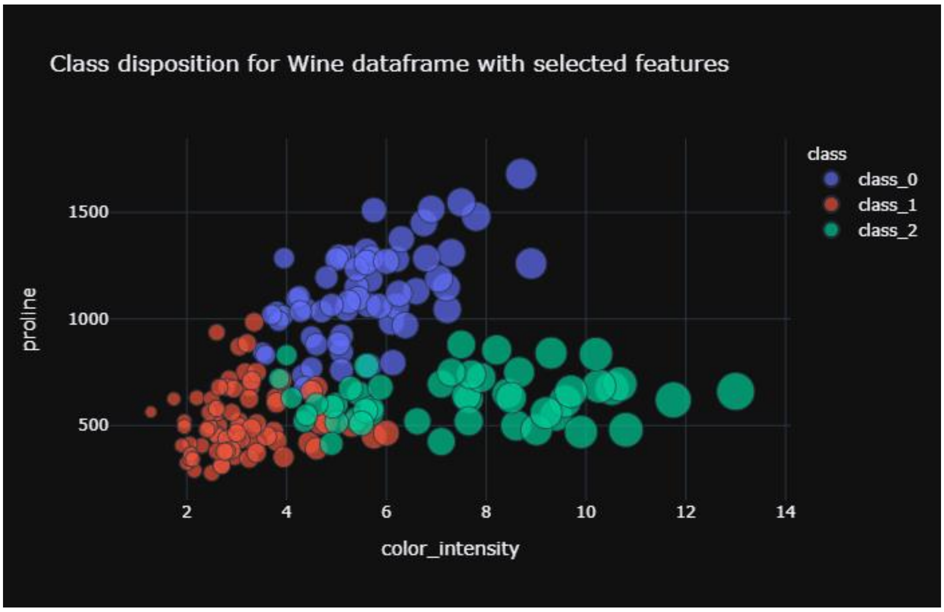


Plot of the class disposition for Iris dataframe with selected features

Kbest Wine

Feature	**Score**	**Pval**
alcohol	5.445499	6.569389e-02
malic_acid	28.068605	8.034890e-07
ash	0.743381	6.895678e-01
alcalinity_of_ash	29.383695	4.163050e-07
magnesium	45.026381	1.669728e-10
total_phenols	15.623076	4.050346e-04
flavonoids	63.334308	1.766565e-14
nonflavonoid_phenols	1.815485	4.034340e-01
proanthocyanis	9.368283	9.240664e-03
color_intensity	109.016647	2.124887e-24
hue	5.182540	7.492483e-02
od280/od315_of_diluited_wines	23.389883	8.335878e-06
proline	16540.067145	0.000000e+00

KBest features for Wine based on score: **proline, color_intensity**.



Plot of the class disposition for Wine dataframe with selected features

Pval varies from 0 to 1 and the lower it is, the more important the feature is.

Models used

The models chosen for the experiments on the two mentioned datasets are the supervised classifiers:

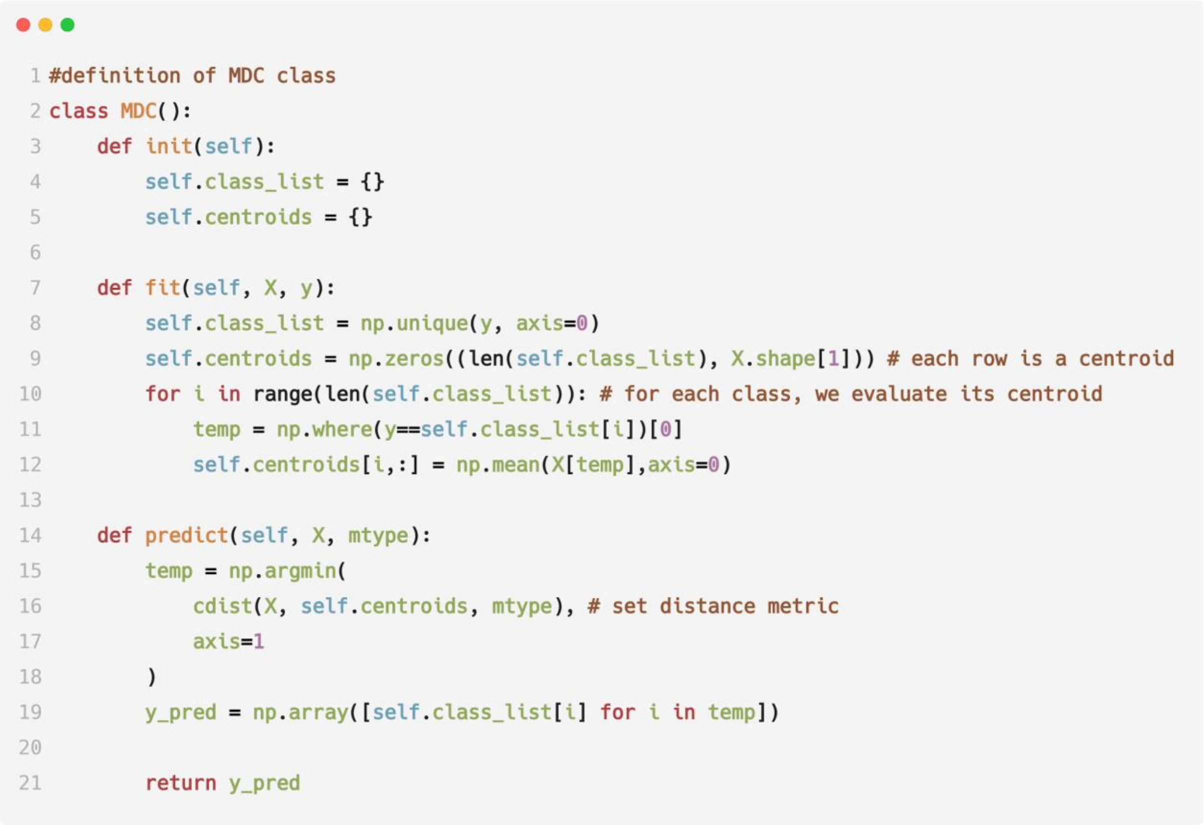
- Minimum Distance Classifier;
- K-Nearest Neighbor.

MDC

For the use of this classifier, we have set a division between training set and test set with a proportion of 70% and 30%. The number of resampling that is performed is also set to 40.

Metrics

Within the classifier class, you can set the metric for the distance calculation. The default metric is the Euclidean distance. Besides this metric we also used Manhattan distance and Chessboard (Chebyshev) distance.

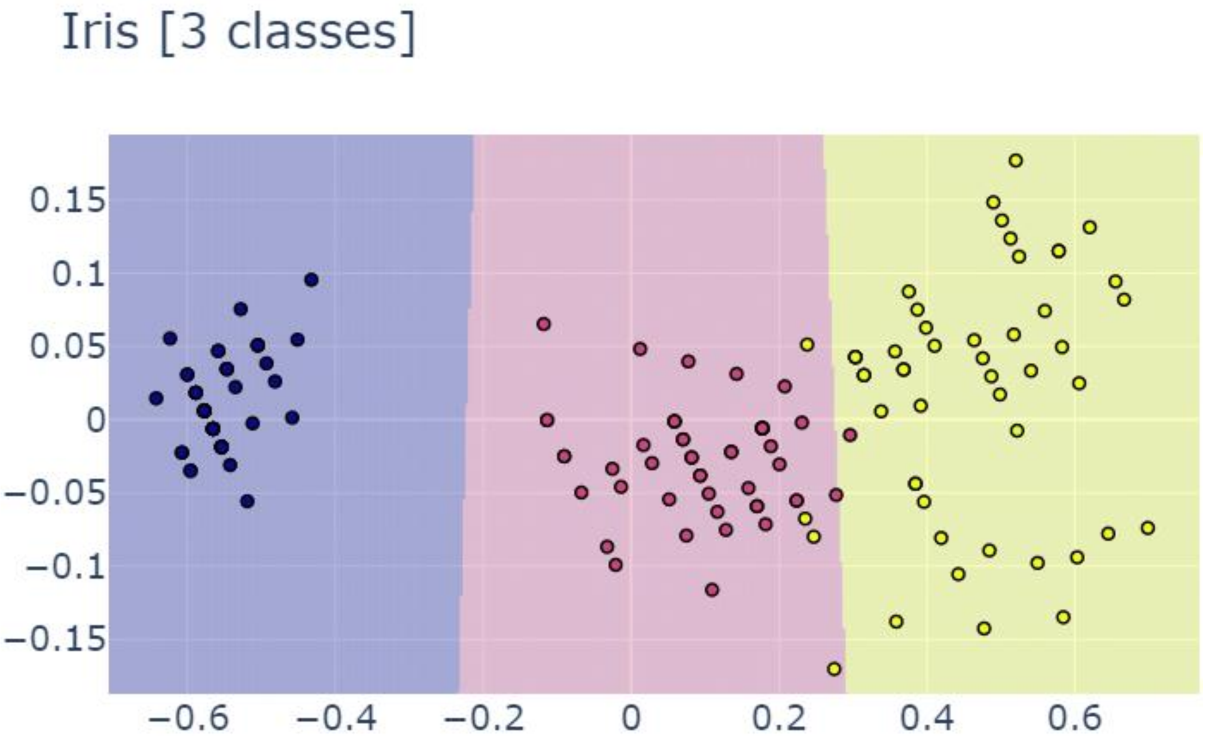


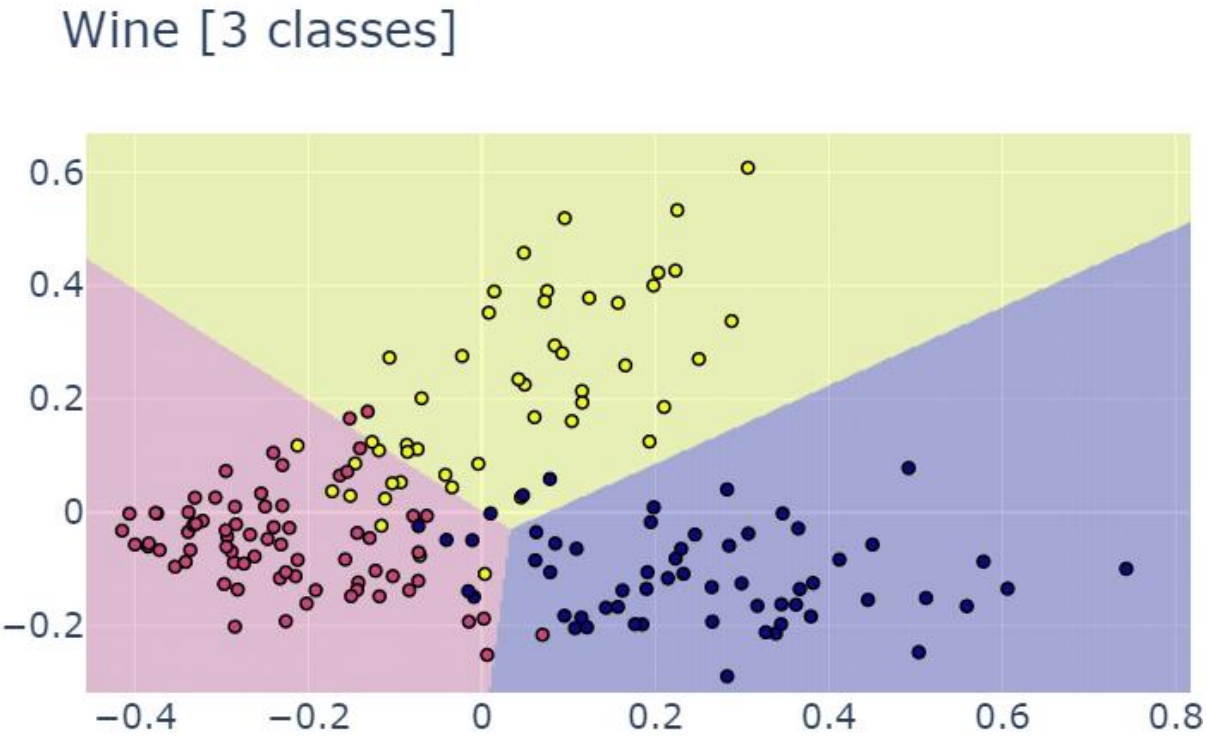
```python
#definition of MDC class
class MDC():
    def init(self):
        self.class_list = {}
        self.centroids = {}

    def fit(self, X, y):
        self.class_list = np.unique(y, axis=0)
        self.centroids = np.zeros((len(self.class_list), X.shape[1])) # each row is a centroid
        for i in range(len(self.class_list)): # for each class, we evaluate its centroid
            temp = np.where(y==self.class_list[i])[0]
            self.centroids[i,:] = np.mean(X[temp],axis=0)

    def predict(self, X, mtype):
        temp = np.argmin(
            cdist(X, self.centroids, mtype), # set distance metric
            axis=1
        )
        y_pred = np.array([self.class_list[i] for i in temp])

        return y_pred
```

Definition of MDC class

Euclidean distance

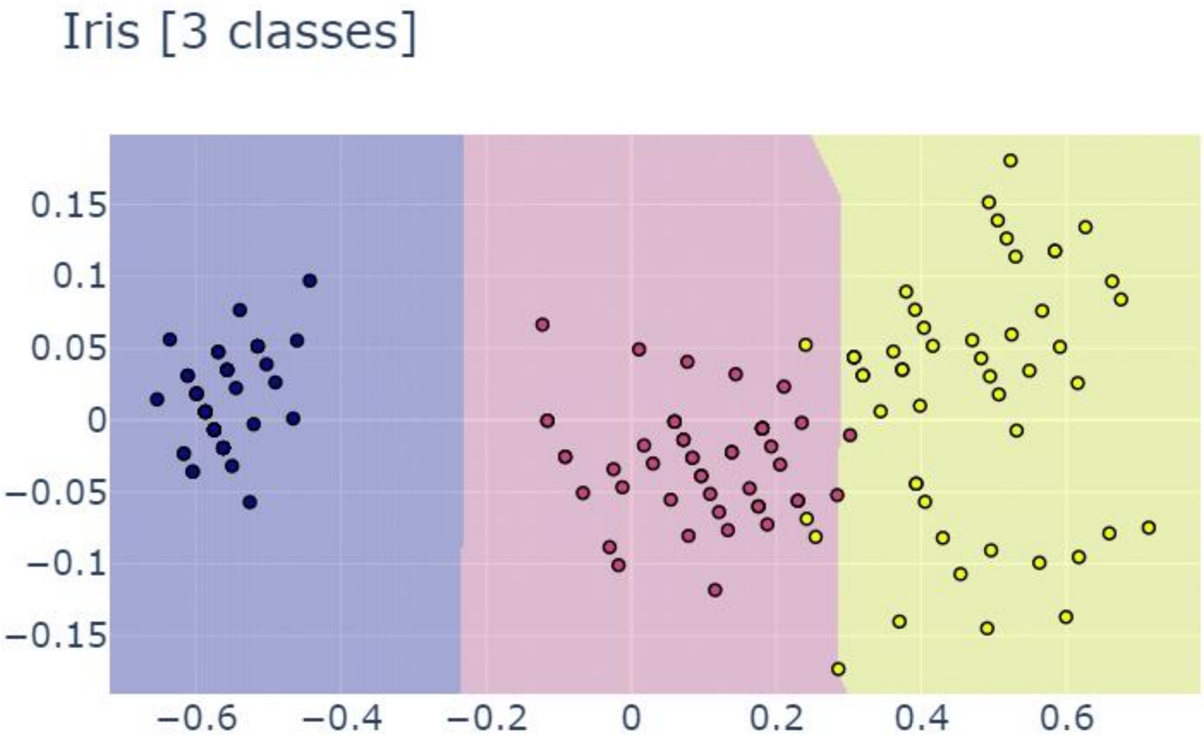
Division of planes using Euclidean distance:

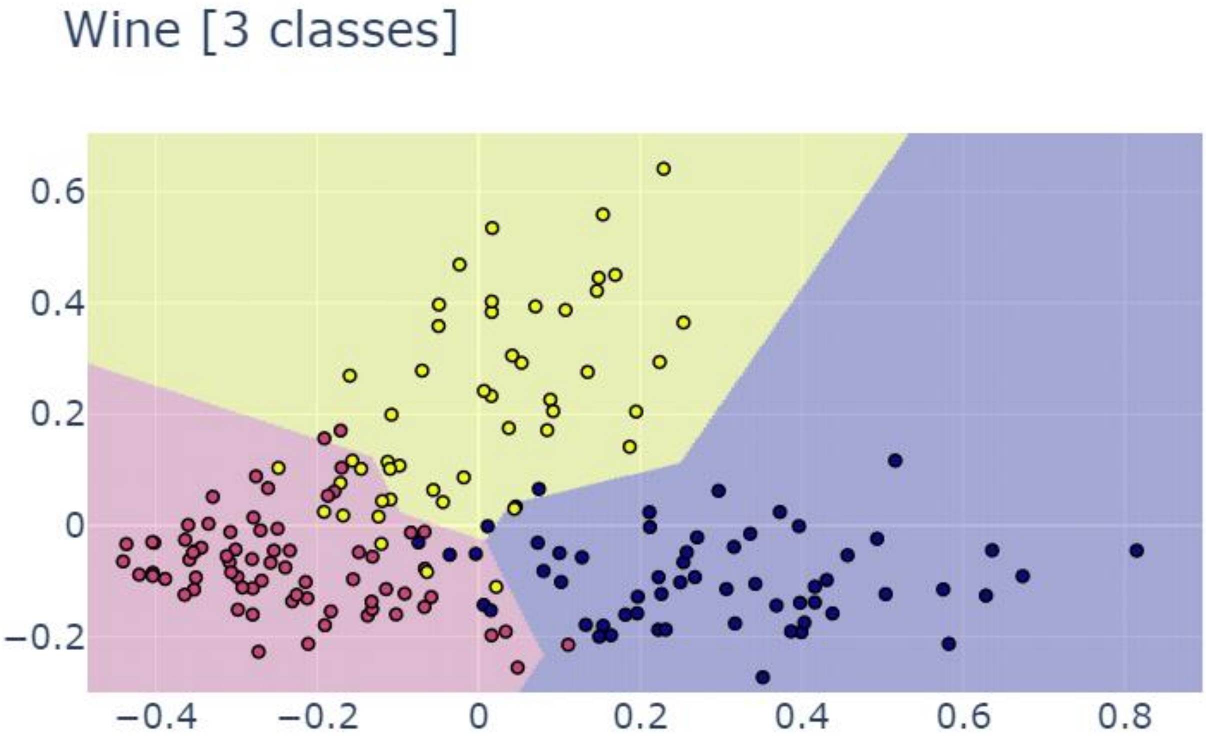




Manhattan distance

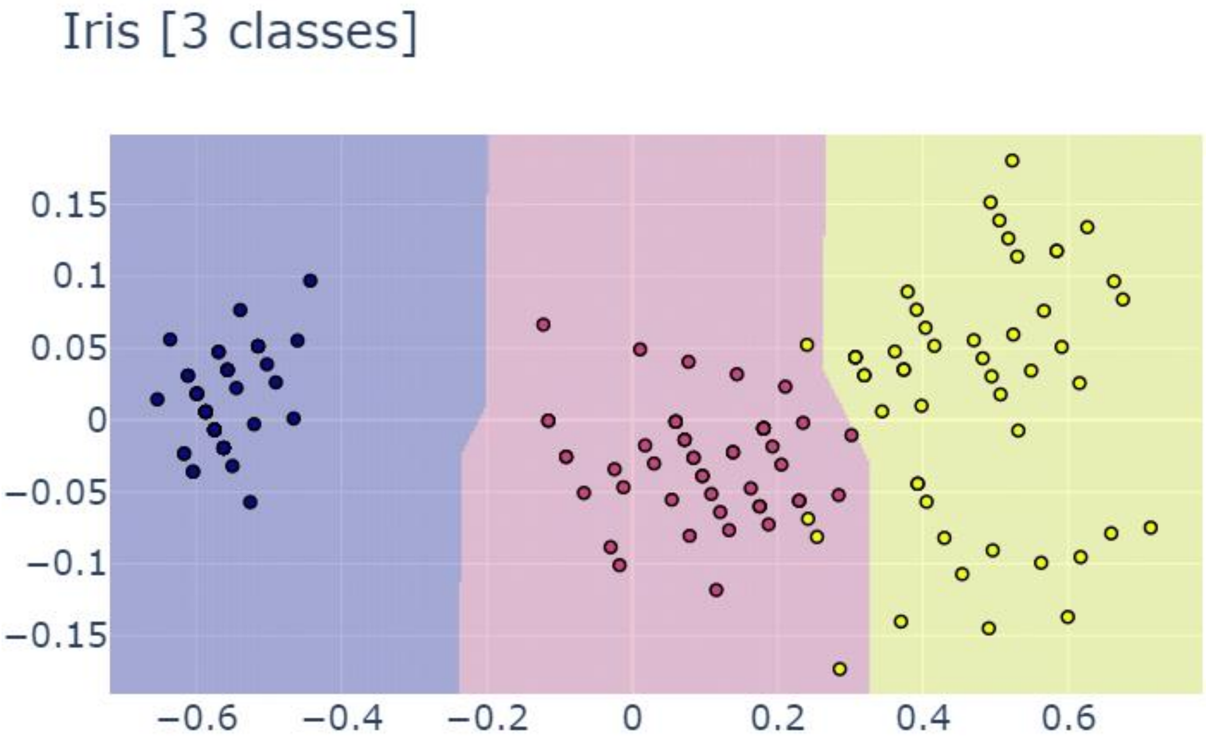
Division of planes using Manhattan distance:

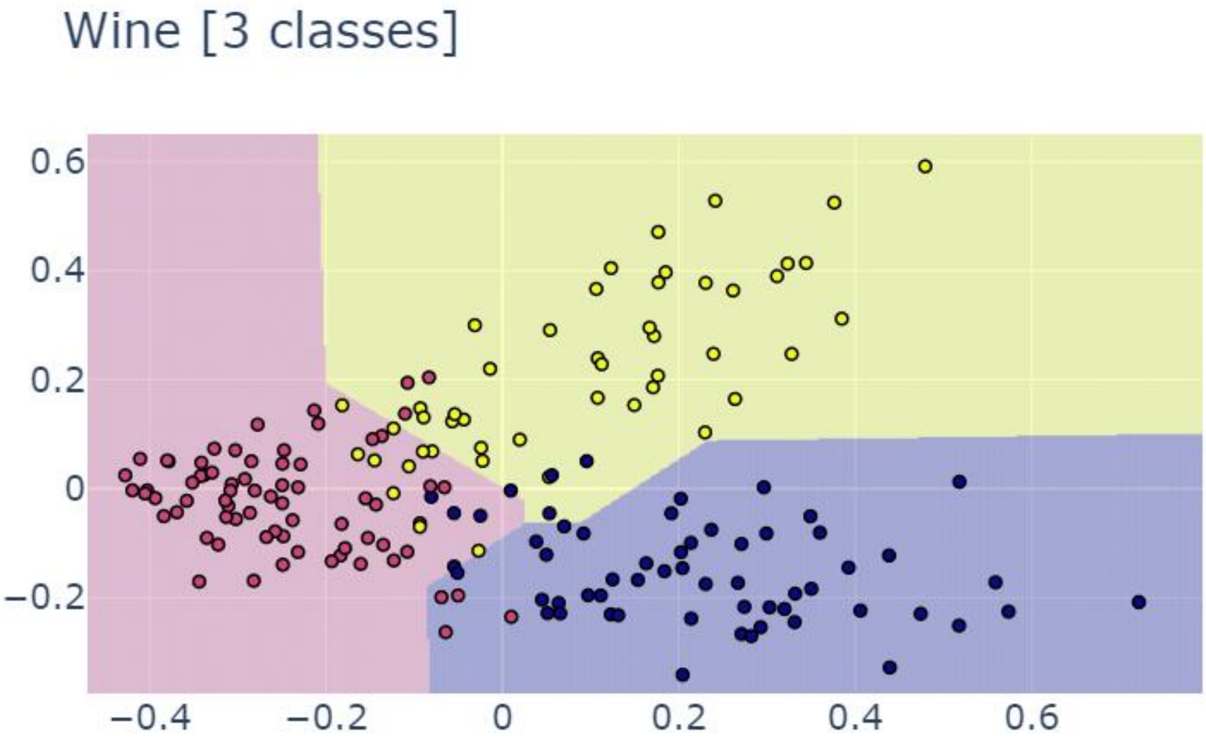




Chessboard (Chebyshev) distance

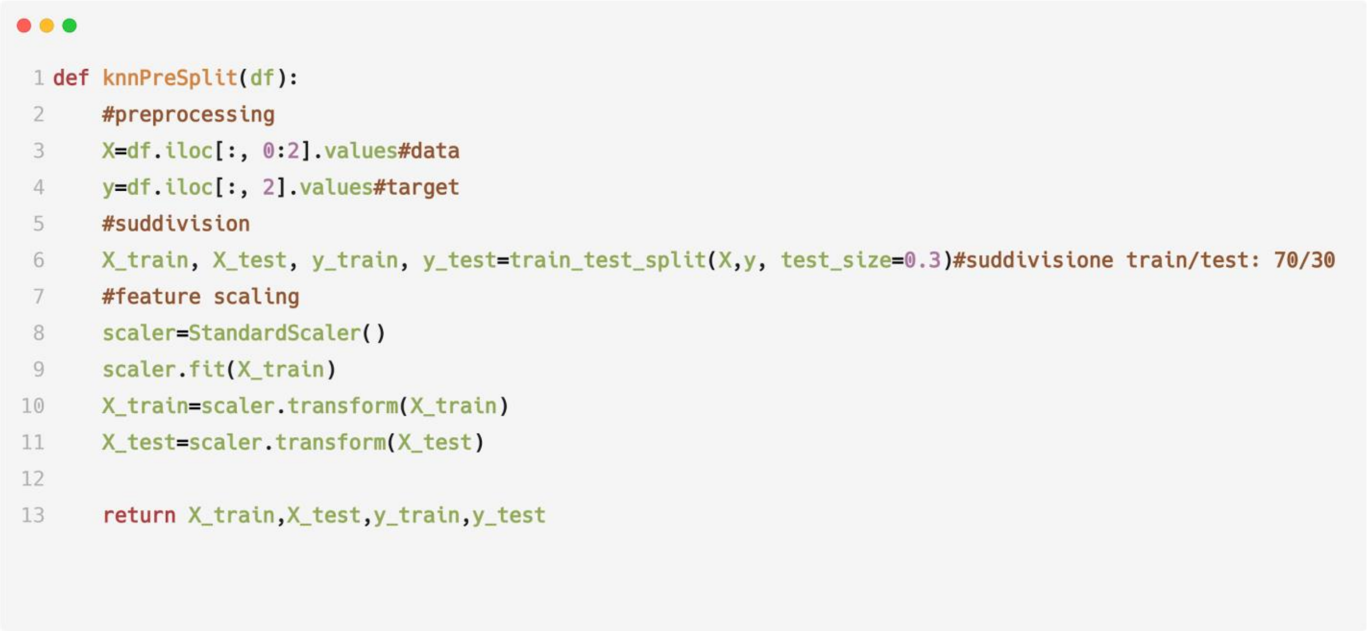
Division of planes using Chessboard distance:





KNN

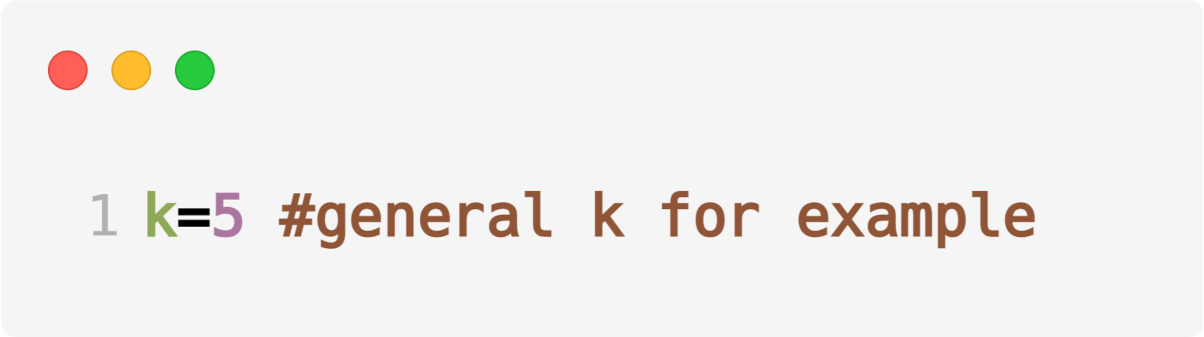
Within the project we have implemented a second algorithm for the classification of patterns: the K-Nearest Neighbor.
K-Nearest Neighbor makes use of Minkowski's metrics, but in this case, we only used the default metric, which is the Euclidean metric.
At the beginning we divided the dataset into training set and test set, with a proportion of 70% - 30%.



```python
def knnPreSplit(df):
    #preprocessing
    X=df.iloc[:, 0:2].values#data
    y=df.iloc[:, 2].values#target
    #suddivision
    X_train, X_test, y_train, y_test=train_test_split(X,y, test_size=0.3)#suddivisione train/test: 70/30
    #feature scaling
    scaler=StandardScaler()
    scaler.fit(X_train)
    X_train=scaler.transform(X_train)
    X_test=scaler.transform(X_test)

    return X_train,X_test,y_train,y_test
```

Definition of knnPreSplit

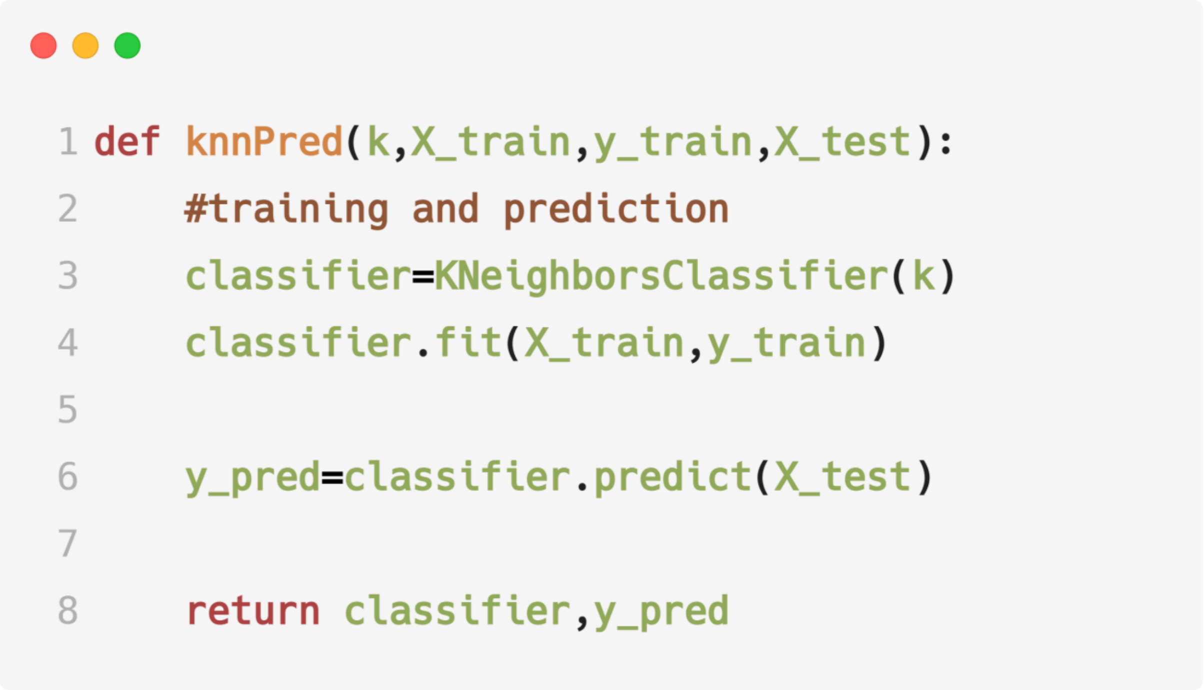
We then set a generic k value for an initial classification.



```python
k=5 #general k for example
```

General case k definition and allocation

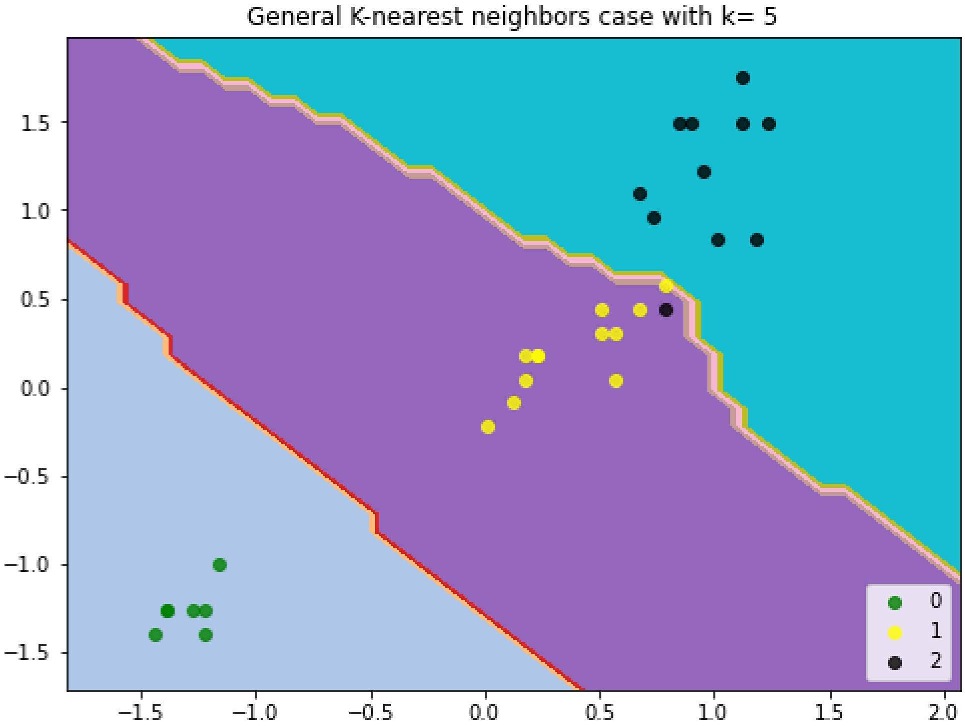
We then ran the classification.



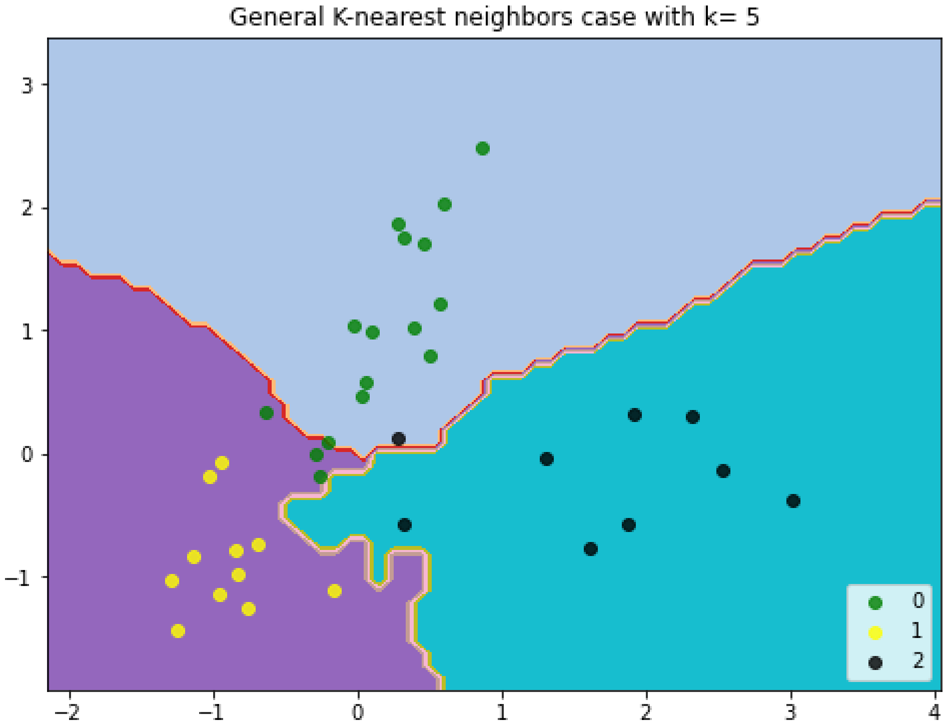
```python
def knnPred(k,X_train,y_train,X_test):
    #training and prediction
    classifier=KNeighborsClassifier(k)
    classifier.fit(X_train,y_train)

    y_pred=classifier.predict(X_test)

    return classifier,y_pred
```

Definition of knnPred

Finally, with the results obtained it was possible to generate the classification plot.



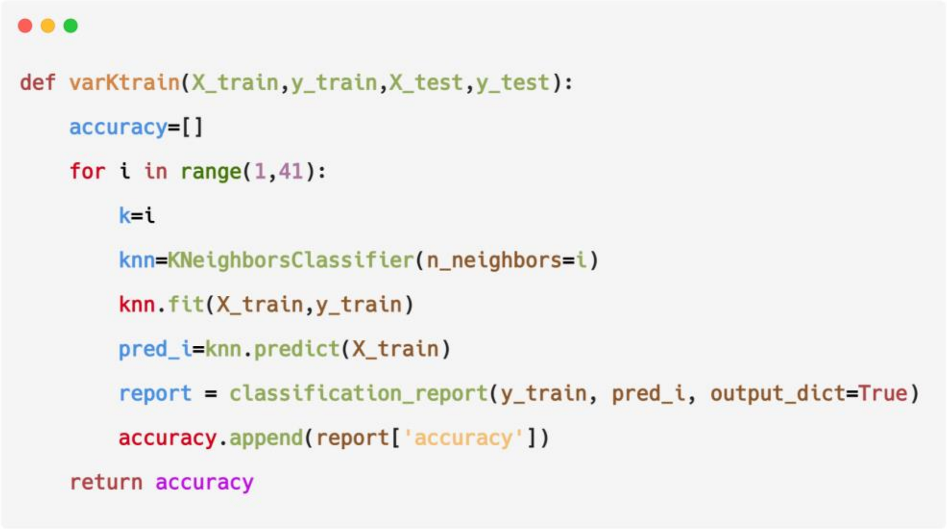
Plot KNN Iris



Plot KNN Wine

Best K case & Worst K case

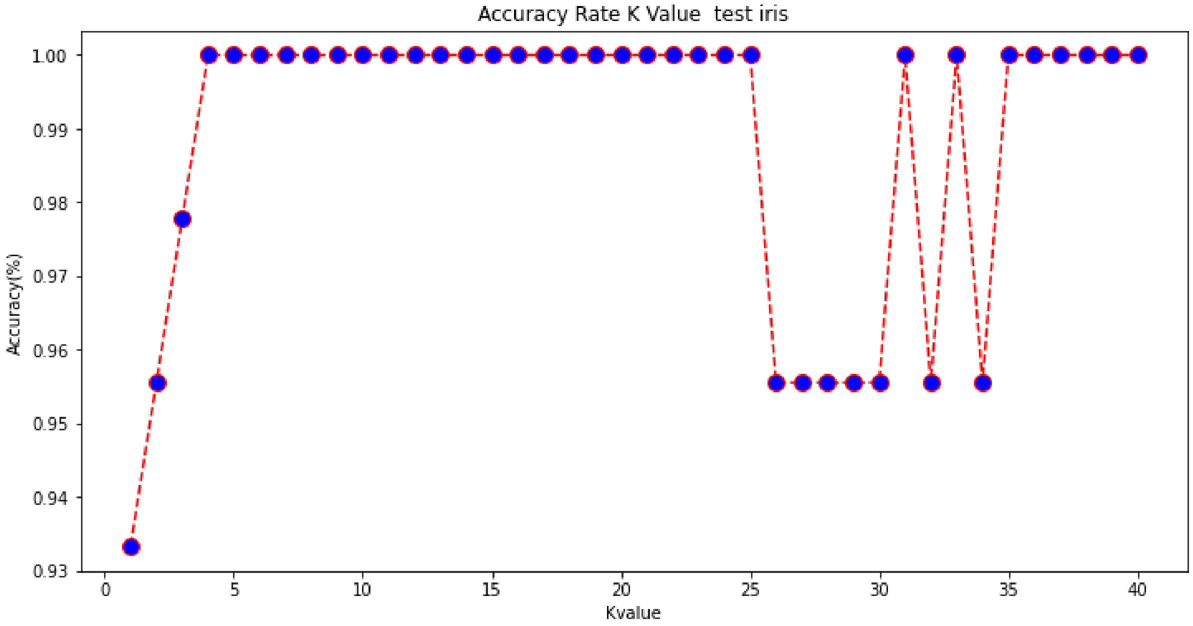
Next, we wanted to see how the classification varies, running the KNN with k increasing from 1 to 40 inclusive. We thus obtain <k - accuracy> pairs.

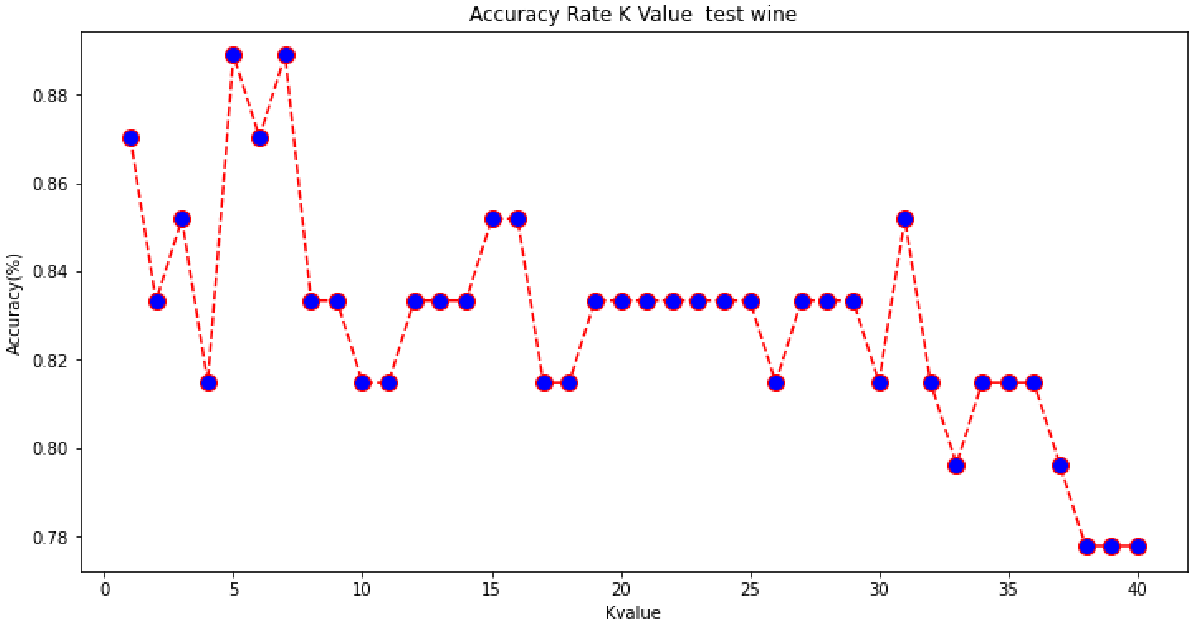


```
def varKtrain(X_train,y_train,X_test,y_test):
    accuracy=[]
    for i in range(1,41):
        k=i
        knn=KNeighborsClassifier(n_neighbors=i)
        knn.fit(X_train,y_train)
        pred_i=knn.predict(X_train)
        report = classification_report(y_train, pred_i, output_dict=True)
        accuracy.append(report['accuracy'])
    return accuracy
```

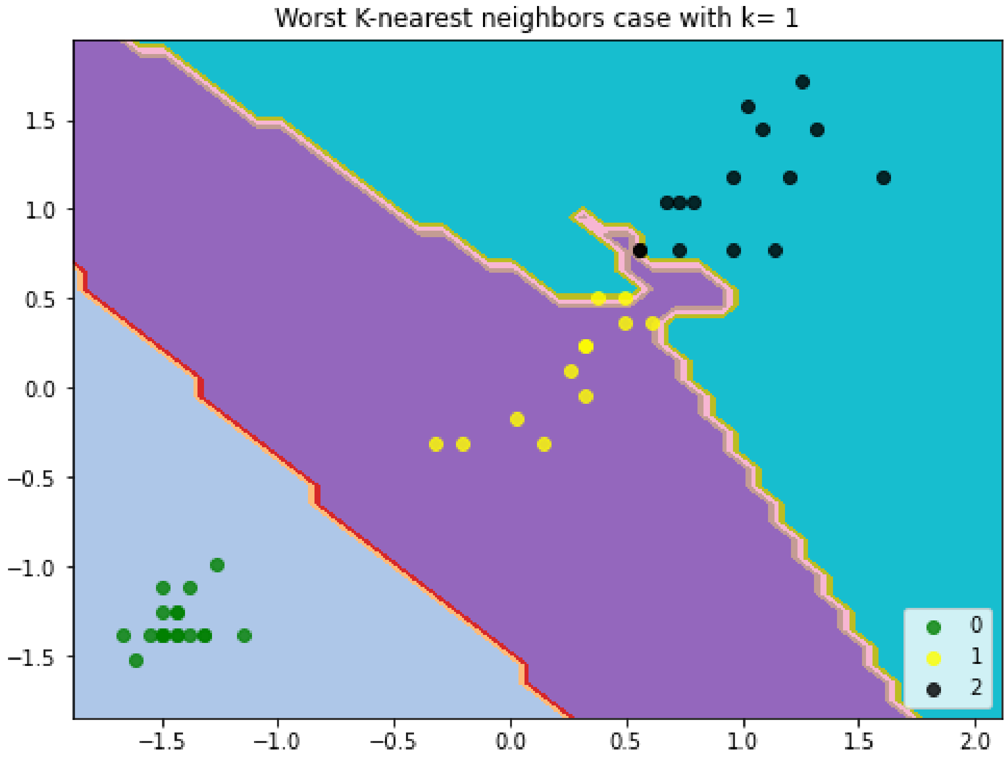
Definition of varKtrain

This will serve us to evaluate a best case (with k having a high level of accuracy) and a worst case (with k having a low level of accuracy).

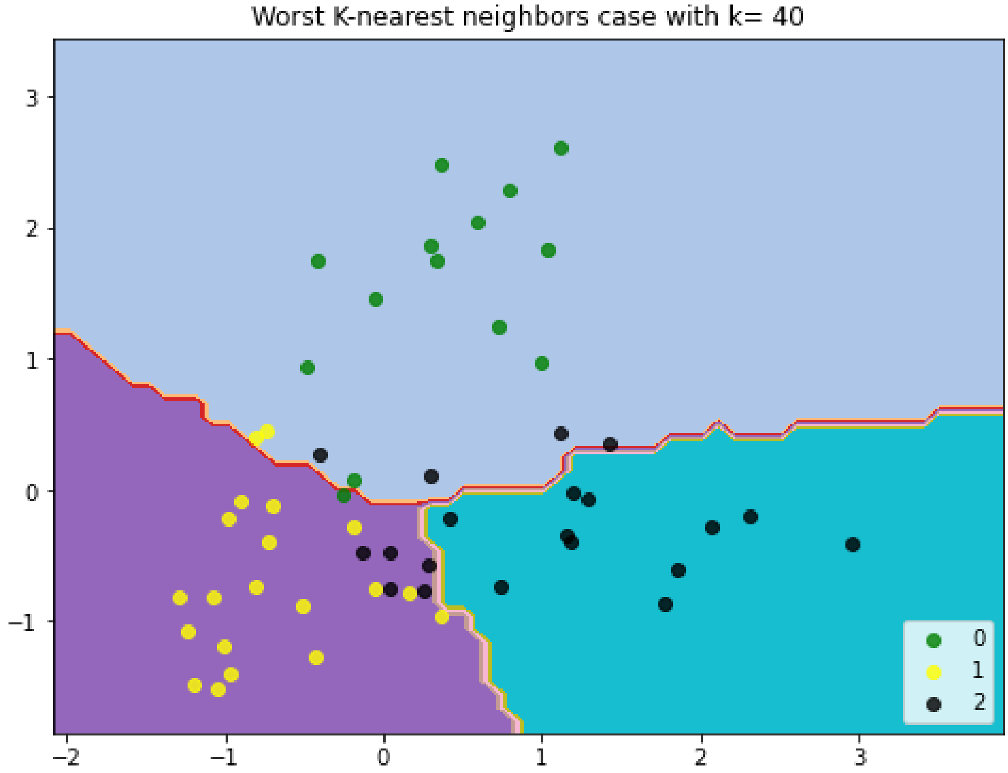




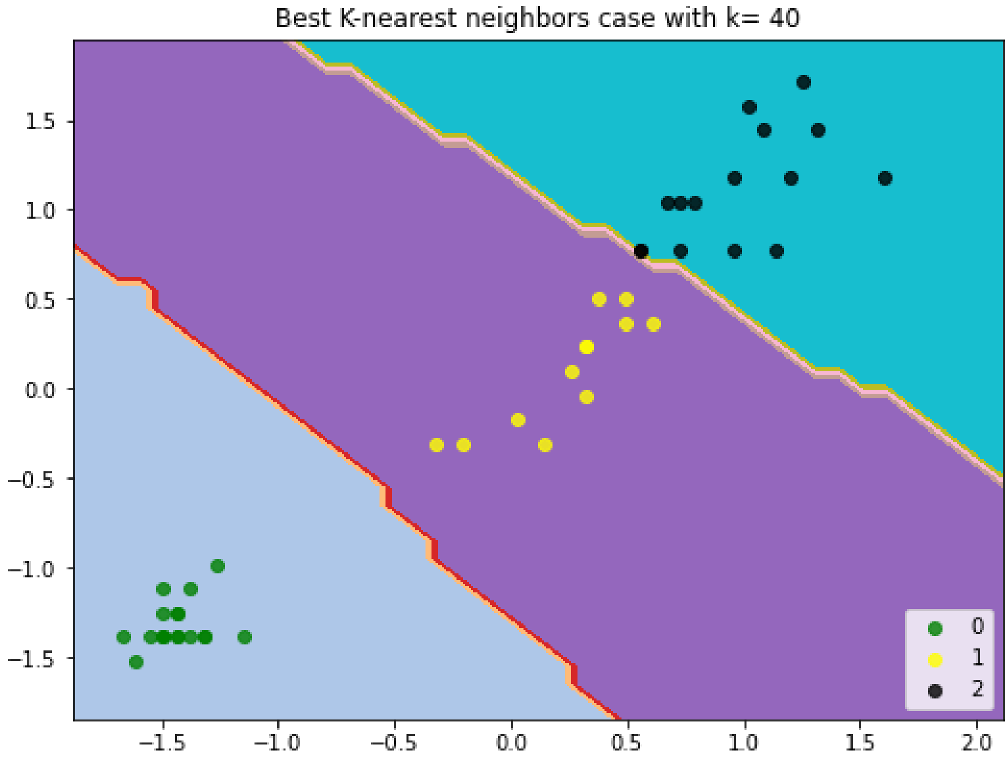
On the basis of the extracted k value we have generated the relative graphs.



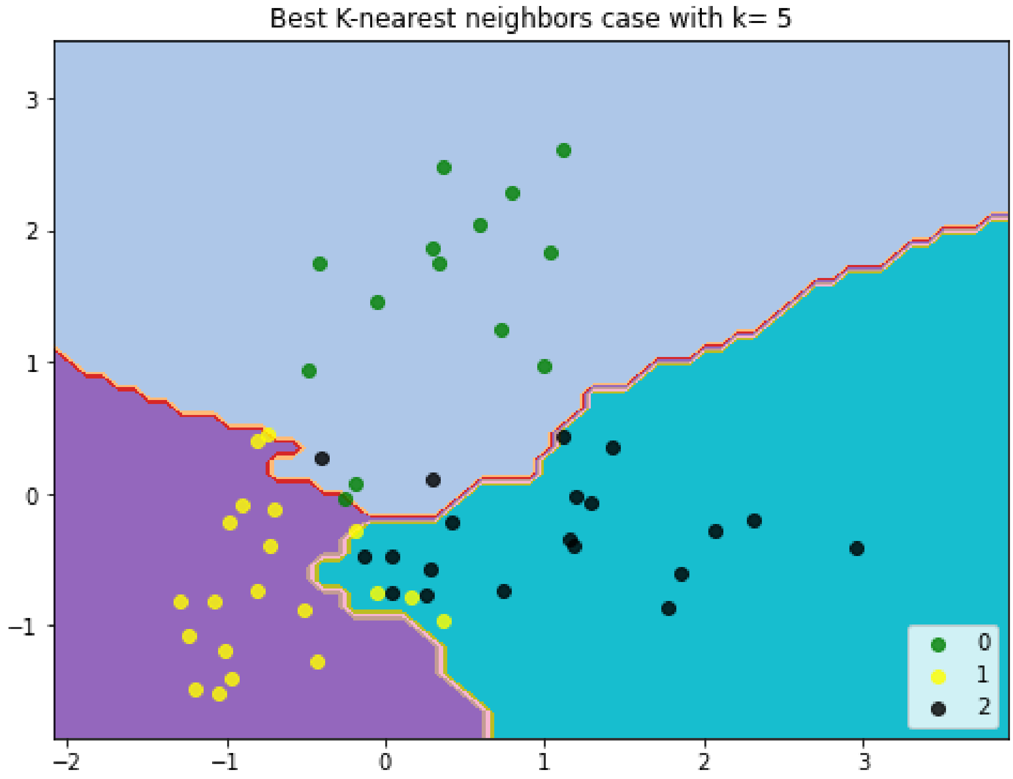
Worst k for Iris



One of the worst k for Wine



One of the best k for iris



One of the best k for wine

Classifiers Evaluation

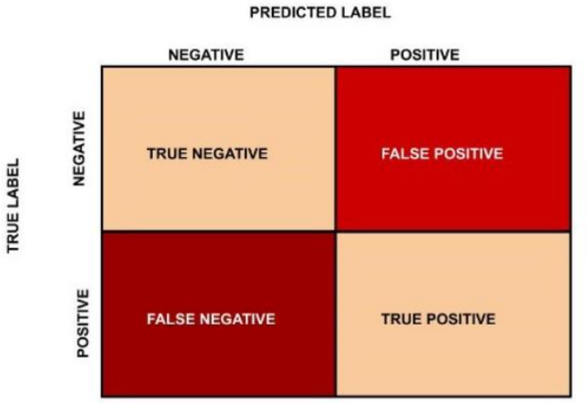
For the evaluation of the performance of the MDC with the three metrics used, we relied on accuracy. Accuracy is the number of correctly classified data instances out of the total number of data instances.

$$accuracy = \frac{true\ negative\ +\ true\ positive}{true\ negative\ +\ true\ positive\ +\ false\ positive\ +\ false\ negative}$$

Regarding the KNN we made use of the *confusion_matrix* and *classification_report* functions. Classification report provided us with the following parameters:

- Precision, is the ability of the classifier not to label as positive a sample that is negative.

$$precision = \frac{true\ positive}{true\ positive + false\ positive}$$



Example of true/false positive/negative

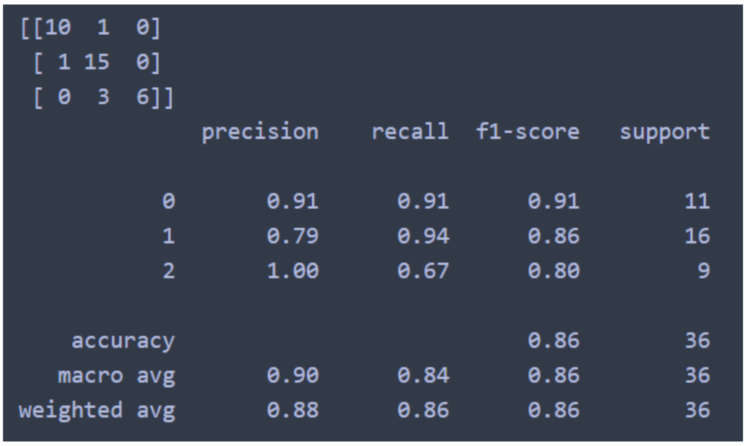
- Recall, indicates the percentage of true positives and the value;

$$recall\ = \frac{true\ positive}{(true\ positive\ +\ false\ negative)}$$

- F1-Score, is the harmonic mean between precision and recall and is a better measure of precision. The value fluctuates between 0 and 1;

$$f1 - score\ = 2\ *\ \frac{Precision\ *\ Recall}{Precision\ +\ Recall}$$

- Support, is the number of occurrences of each class in y_test;
- Accuracy;
- Macro-AVG, calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.
- Weighted-AVG, calculate metrics for each label, and find their average weighted by support (the number of true instances for each label).



```
[[10  1  0]
 [ 1 15  0]
 [ 0  3  6]]
              precision    recall  f1-score   support

           0       0.91      0.91      0.91        11
           1       0.79      0.94      0.86        16
           2       1.00      0.67      0.80         9

    accuracy                           0.86        36
   macro avg       0.90      0.84      0.86        36
weighted avg       0.88      0.86      0.86        36
```

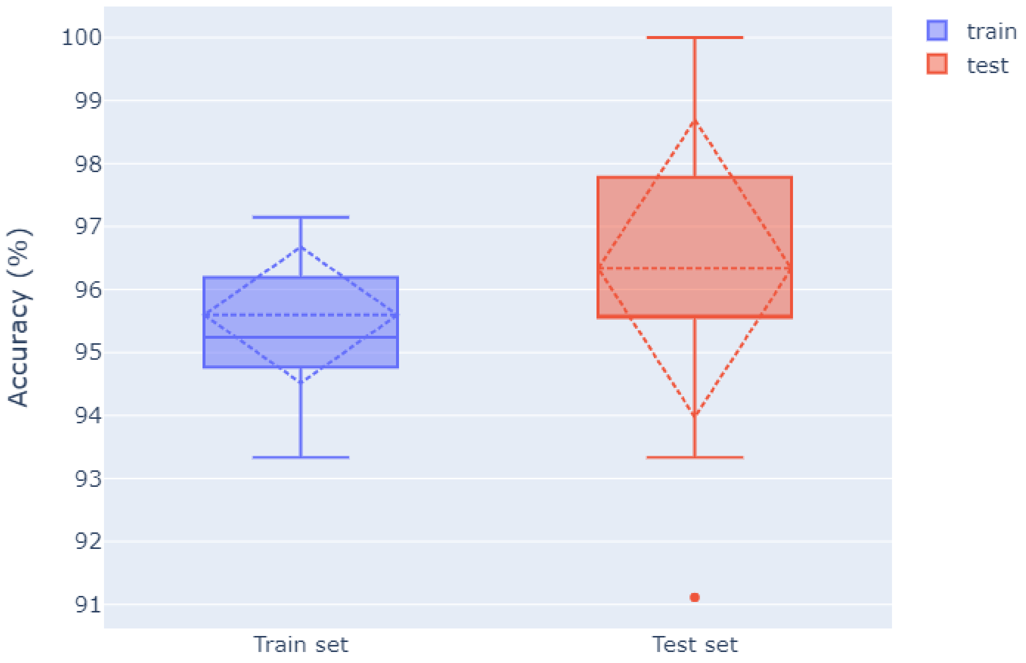
Example of output from the two functions

In order to compare the performance of MDC and KNN with the same accuracy metric, for the KNN we have chosen to rely exclusively on the accuracy metric.

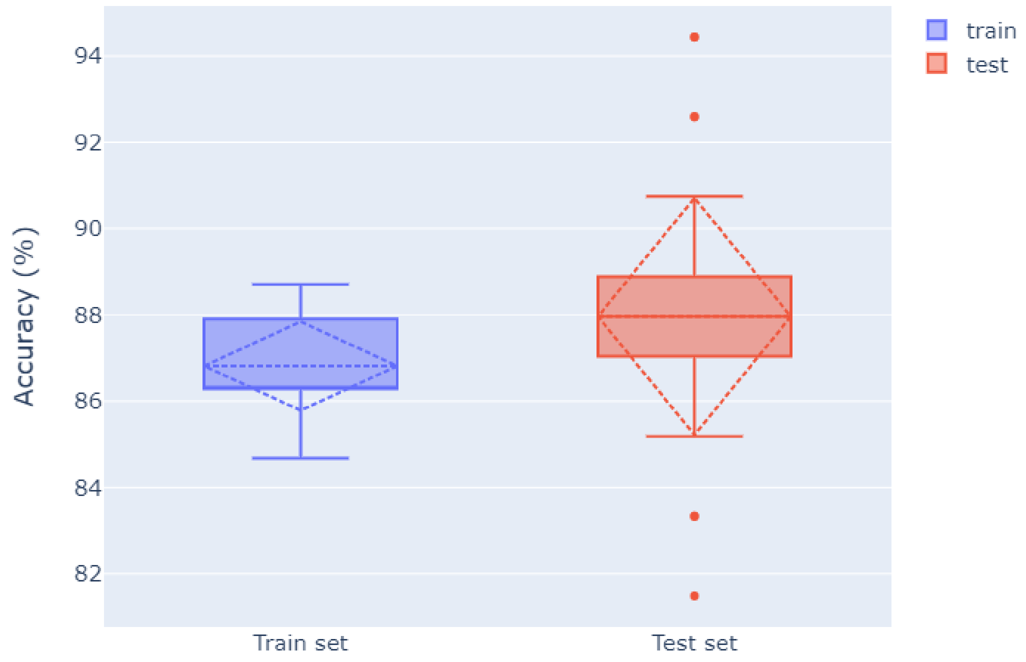
The data that will be commented in the next sections are the results obtained from our experiment and may vary with new executions of the code. The performance evaluation of the two classifiers is based on these data.

MDC Performance Evaluation

MDC Euclidean



Iris dataframe

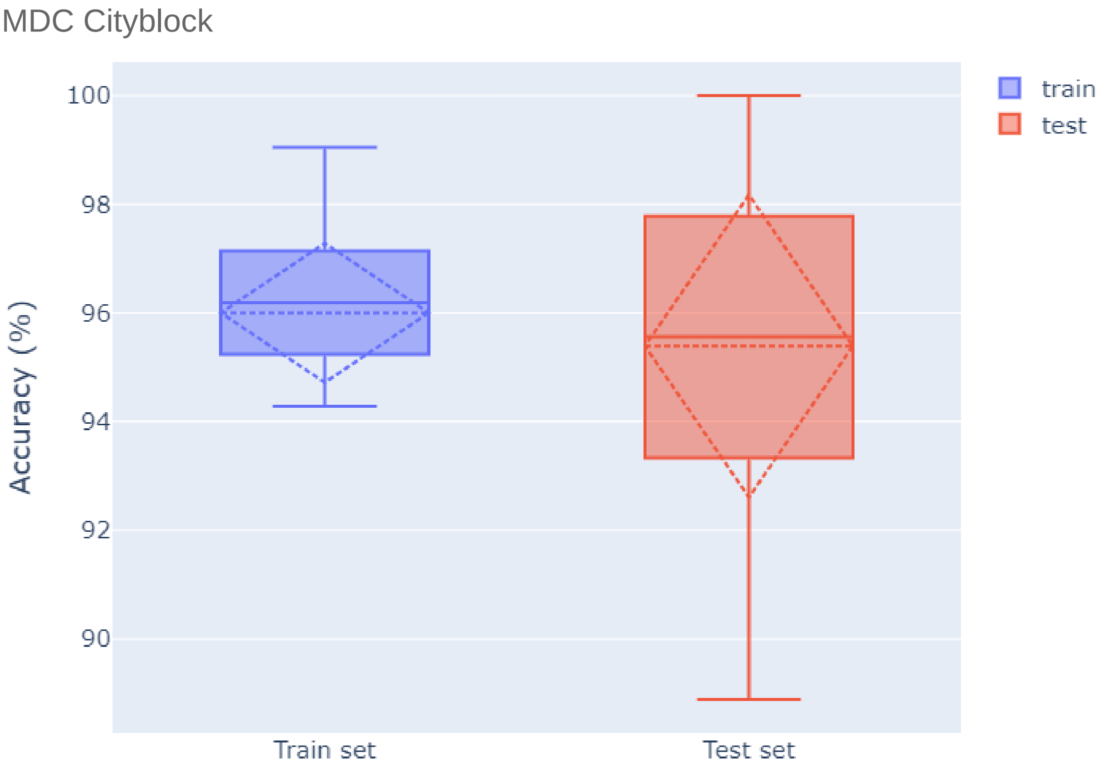


Wine dataframe

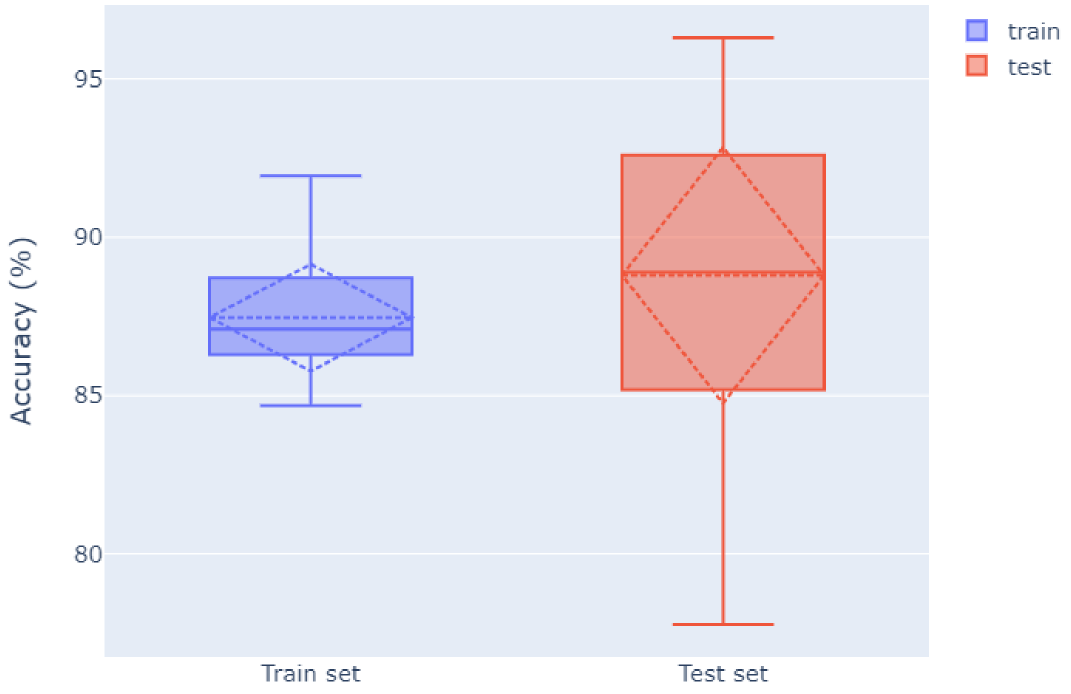
As can be seen from the boxplots, the MDC with Euclidean metric, performs better in terms of accuracy, in the train phase, on the Iris dataframe, with an average accuracy of 95.59%, minimum value of 93.33% and maximum of 97,14% while for the Wine dataframe an average of 86.81% is obtained, a minimum value of 84.67% and a maximum of 88.70%. The best distribution in the train phase is obtained for the Wine dataframe although its values are significantly lower than those obtained by the Iris dataframe.

As can be seen from the boxplot, the MDC with Euclidean metric, performs better in terms of accuracy, in the test phase, on the Iris dataframe, with an average accuracy of 96.33%, minimum outlier value of 91.11% and maximum of 100% while for the Wine dataframe an average of 87.96% is obtained, a minimum outlier value of 81.48% and a maximum outlier of 94.44%. The best distribution in the test phase is obtained for the Wine dataframe despite its values are significantly lower than those obtained by the Iris dataframe.

Using the Euclidean metric, the MDC performs better both in the train phase and in the test phase for the Iris dataframe.



Iris dataframe

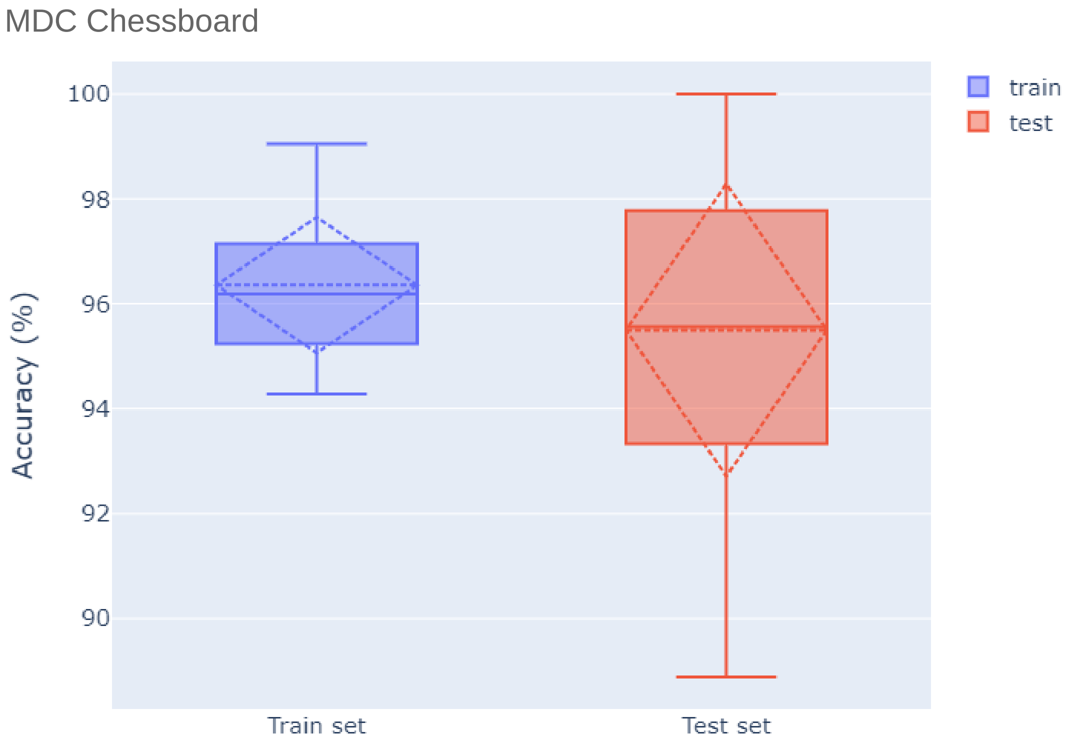


Wine dataframe

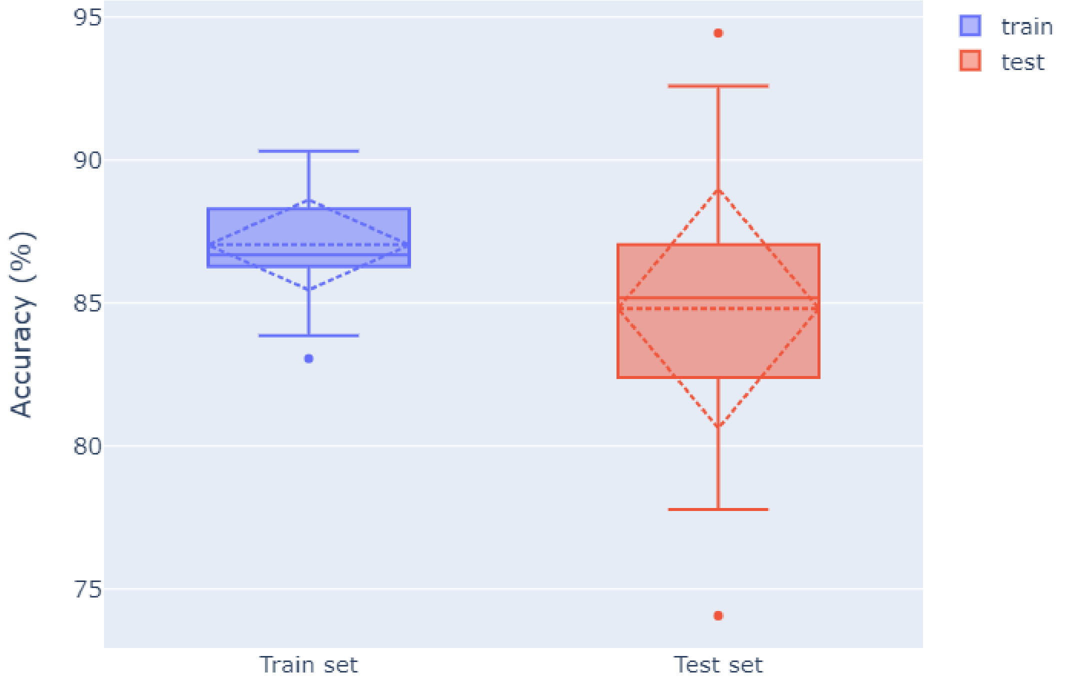
As you can see from the boxplots, the MDC with Cityblock metric performs better in terms of accuracy, in the train phase, on the Iris dataframe, with an average accuracy of 96%, a minimum value of 94.28% and a maximum of 99. 04% while for the Wine dataframe an average of 87.46% is obtained, a minimum value of 84.67% and a maximum of 91.93%. In this case there is a better distribution in the train phase for the Iris dataframe.

As you can see from the boxplots, the MDC with Cityblock metric, performs better in terms of accuracy, in the test phase, on the Iris dataframe, with an average accuracy of 95.39%, minimum value of 88.89% and maximum of 100% while for the Wine dataframe an average of 88.80% is obtained, a minimum value of 77.78% and a maximum of 96.29%. Also in the test phase there is a better distribution of the data for the Iris dataframe.

With the Cityblock metric, the MDC performs better both in the train phase and in the test phase for the Iris dataframe.



Iris dataframe



Wine dataframe

According to the boxplots, the MDC with Chessboard metric performs better in terms of accuracy, in the train phase, on the Iris dataframe, with an average accuracy of 96.36%, minimum value of 94.29% and maximum of 99.05 % while for the Wine dataframe an average of 87.03% is obtained, a minimum outlier value of 83.06% and a maximum of 90.32%. With this metric there is a better distribution in the train phase for the Iris dataframe.

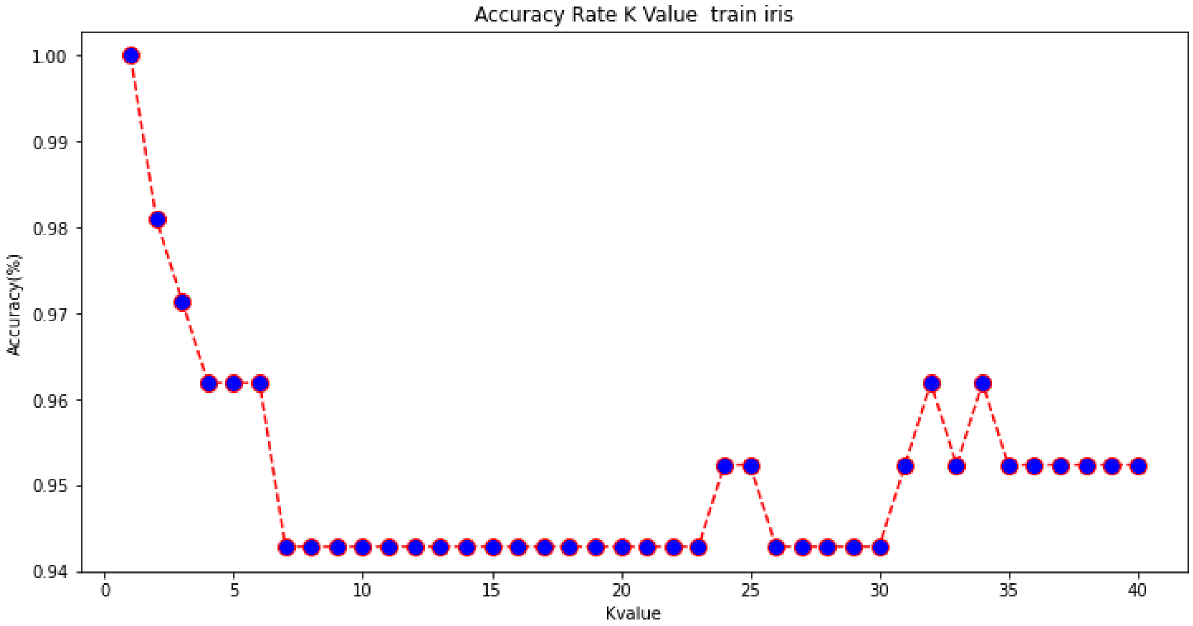
According to the boxplots, the MDC with Chessboard metrics performs better in terms of accuracy, in the test phase, on the Iris dataframe, with an average accuracy of 95.50%, a minimum value of 88.89% and a maximum of 100% while for the Wine dataframe an average of 84.81% is obtained, a minimum outlier value of 74.07% and a maximum outlier of 94.44%. With this metric you have a better distribution in the test phase for the Iris dataframe.

Using the Chessboard metric, the MDC performs better in both the train and test phase for the Iris dataframe.

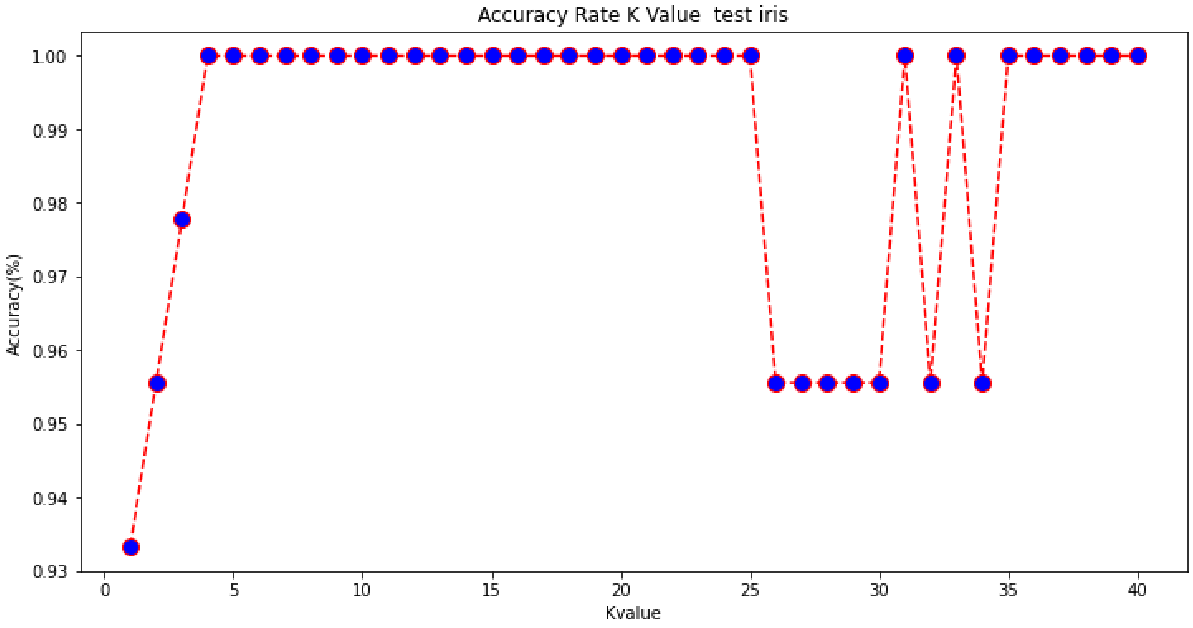
Considering the results seen so far we can say that the MDC achieves better accuracy values with all three metrics for the Iris dataframe. The distribution of these values, most of the time, is better for the Iris dataframe while in the case with Euclidean metrics the distribution of the data is in favor of the Wine dataframe, despite having obtained worse accuracy values.

KNN Performance Evaluation

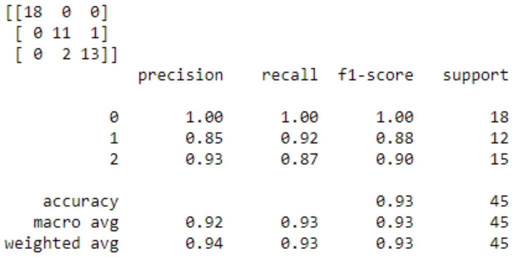
Iris dataframe



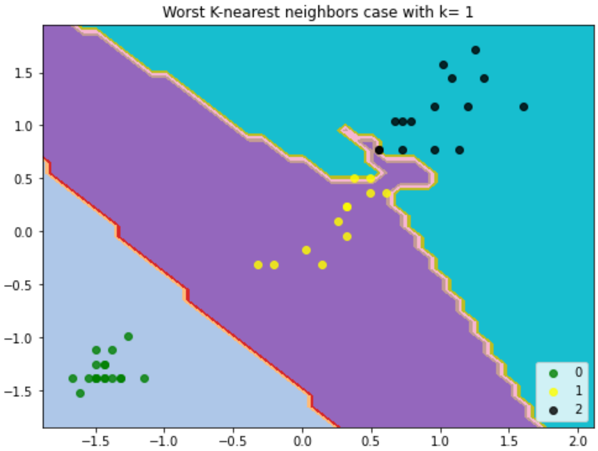
By varying the value of k in the train phase, we have noticed that there is a loss of accuracy as the value of k rises (k = 1 100%, k = 7 94% ~) reaching the minimum value. By further increasing k there is a slight rise in the accuracy value (k = 31 95.5%, k = 32 96%) but without the results obtained from a low k value.



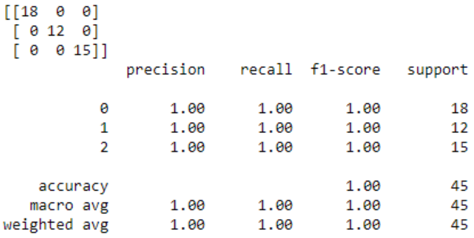
The analysis continues with the variation of the k value in the test phase, where we noticed a rapid increase in accuracy right from the start (k = 1 93%, k = 4 100%) with the increase of the k value by a few units. The accuracy is kept stable up to a high number of k (k = 25) and then decreases in value (k = 26-30 95.5%) and then rises swinging up to stabilize with very high k (k = 35- 40 100%). The accuracy in these 40 k remains high in most cases.



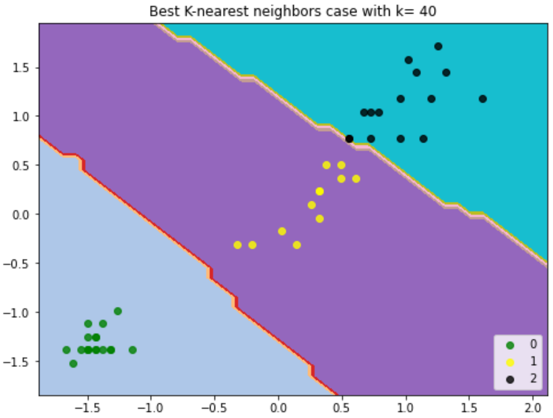
```
[[18  0  0]
 [ 0 11  1]
 [ 0  2 13]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        18
           1       0.85      0.92      0.88        12
           2       0.93      0.87      0.90        15

    accuracy                           0.93        45
   macro avg       0.92      0.93      0.93        45
weighted avg       0.94      0.93      0.93        45
```



Worst k for Iris dataframe (k=1)

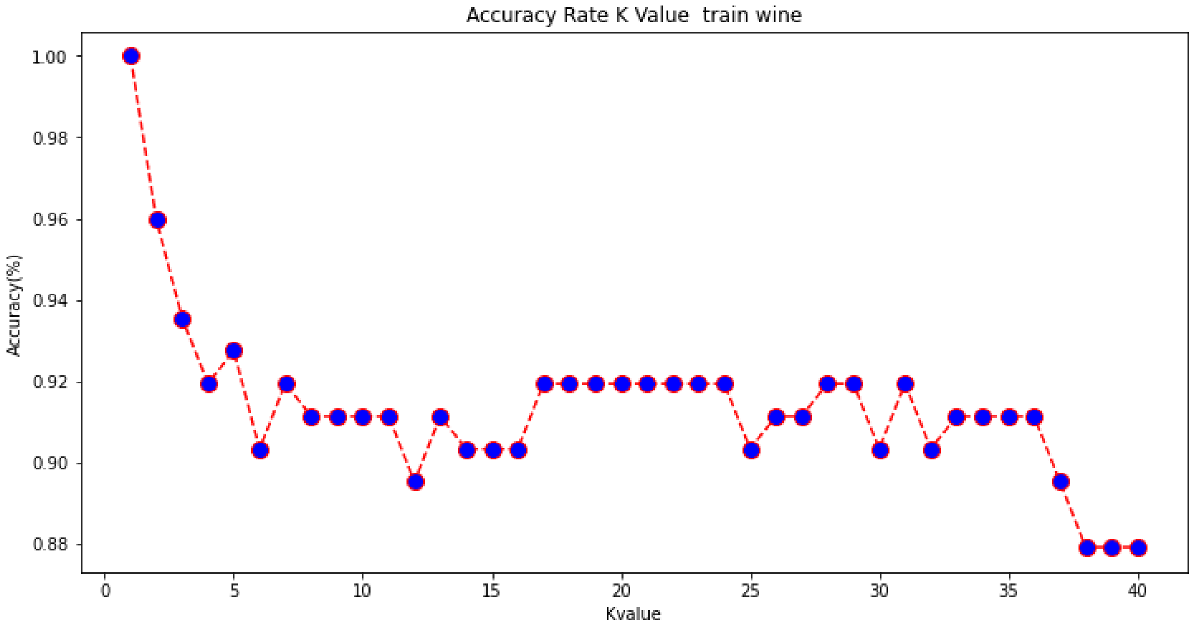


```
[[18  0  0]
 [ 0 12  0]
 [ 0  0 15]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        18
           1       1.00      1.00      1.00        12
           2       1.00      1.00      1.00        15

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

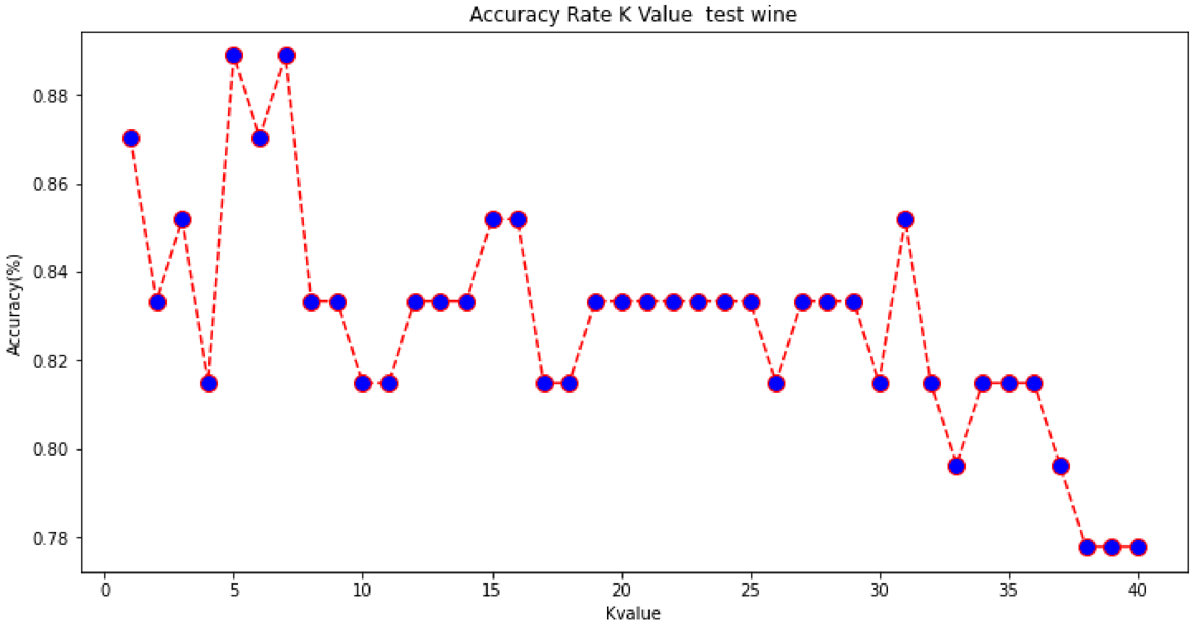


One of the best k for Iris dataframe (k=40)

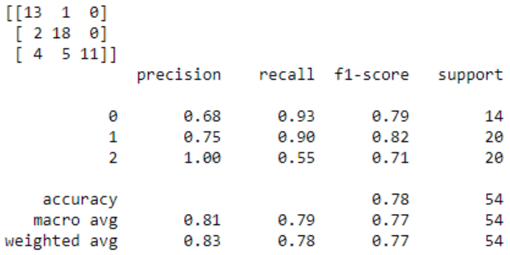
Wine dataframe



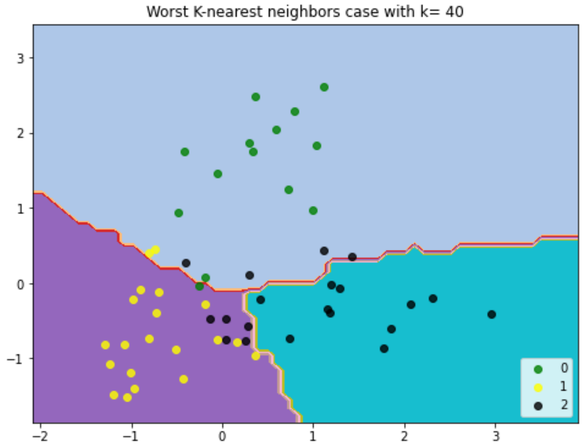
By varying the value of k during the train phase for the Wine dataframe, we immediately noticed a loss of accuracy (k = 1 100%, k = 6 90%). By increasing the k value (from 7 to 35) the accuracy remains in a range that varies from 80% to 92.5%. Arriving at very high k values (k = 36), a further decrease in accuracy begins up to an absolute minimum (k = 38 88%).



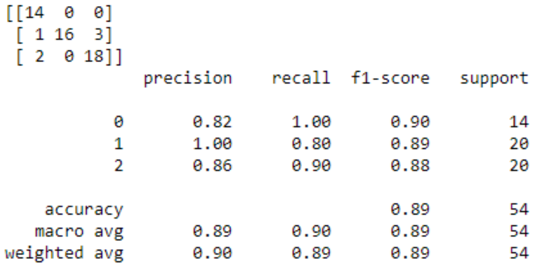
By varying the value of k in the test phase for the Wine dataframe, we notice a fluctuating trend, with 2 points of maximum accuracy (k = 5 and k = 7, 89%). The accuracy level in the general tendence is decreasing until reaching very high k values (k = 38,39,40) where the lowest accuracy level is obtained (78%).



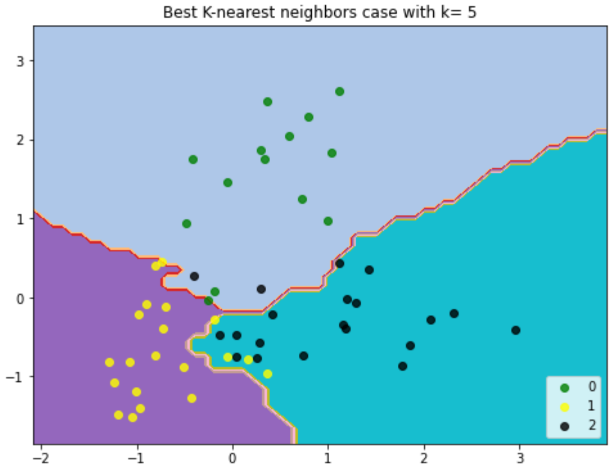
```
[[13  1  0]
 [ 2 18  0]
 [ 4  5 11]]
              precision    recall  f1-score   support

           0       0.68      0.93      0.79        14
           1       0.75      0.90      0.82        20
           2       1.00      0.55      0.71        20

    accuracy                           0.78        54
   macro avg       0.81      0.79      0.77        54
weighted avg       0.83      0.78      0.77        54
```



One of the worst k for Wine dataframe (k=40)

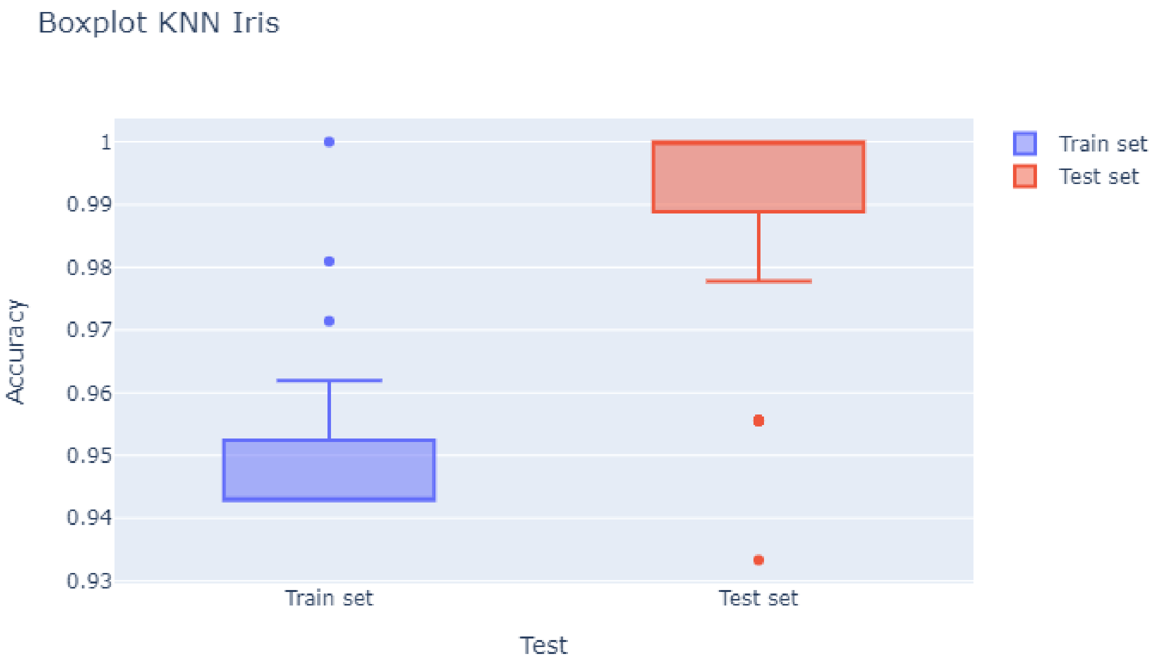


```
[[14  0  0]
 [ 1 16  3]
 [ 2  0 18]]
              precision    recall  f1-score   support

           0       0.82      1.00      0.90        14
           1       1.00      0.80      0.89        20
           2       0.86      0.90      0.88        20

    accuracy                           0.89        54
   macro avg       0.89      0.90      0.89        54
weighted avg       0.90      0.89      0.89        54
```

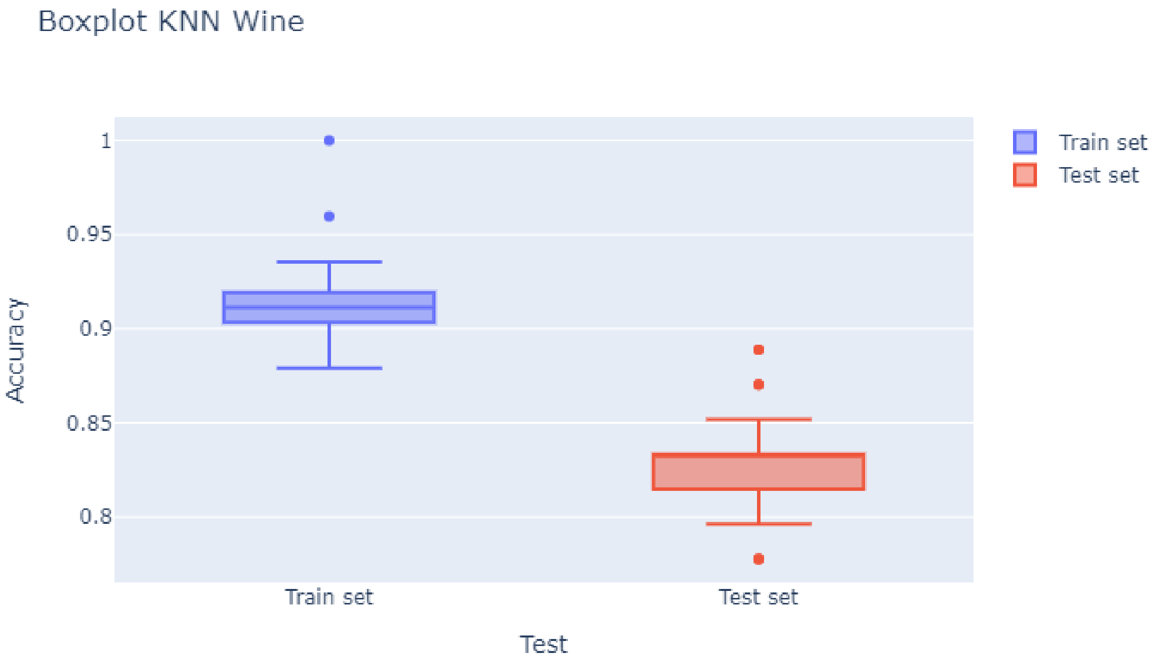


One of the best k for Wine dataframe (k=5)

Overall accuracy evaluation for KNN



Iris dataframe KNN boxplots



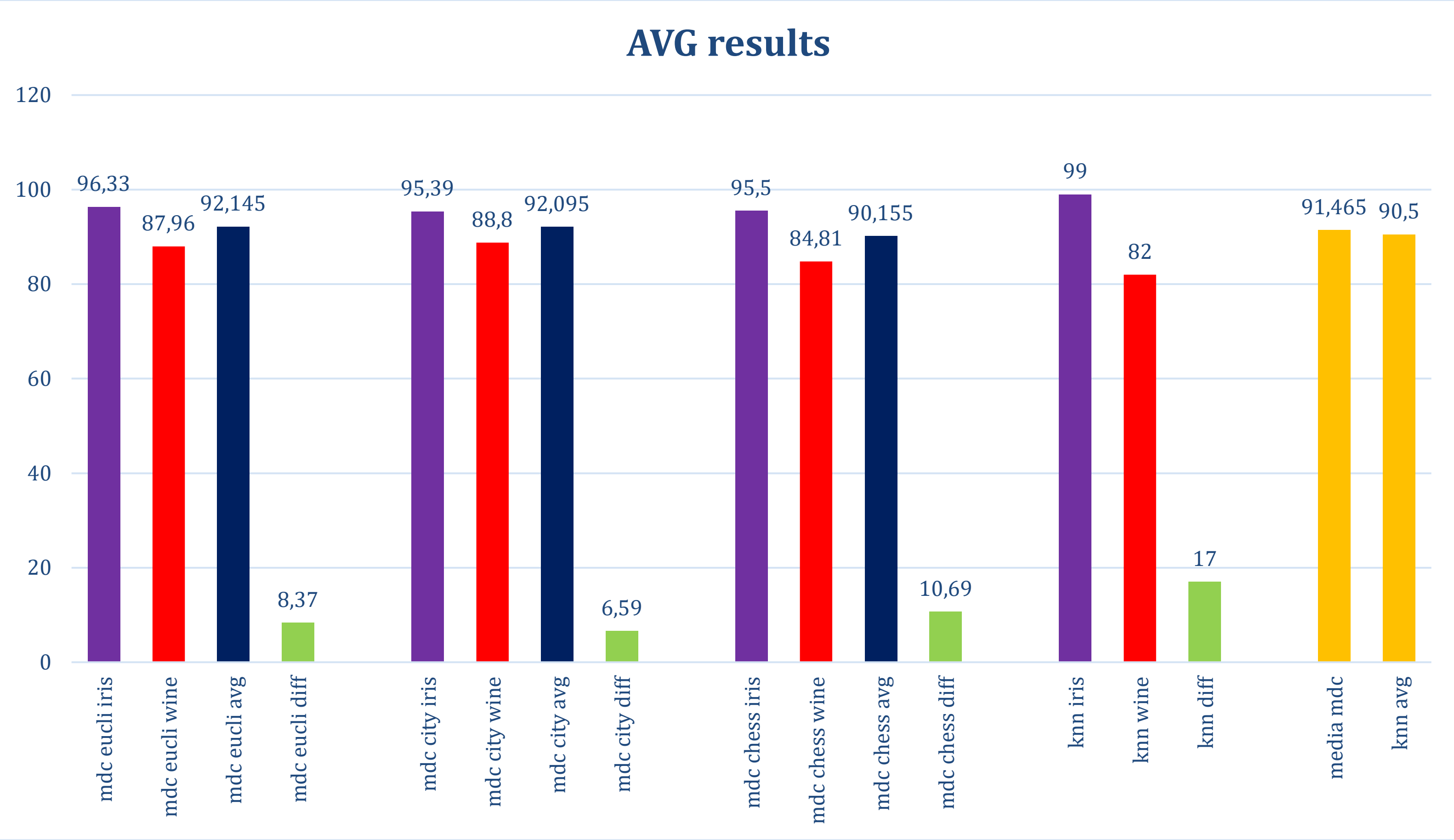
Wine dataframe KNN boxplots

The distribution of the accuracy values in the train phase is better for the Iris dataframe, with a range that varies from 0.94 to 0.96 and with the presence of 3 outliers, one of which with a score of 1. Regarding the Wine dataframe, its values are in a range that varies between 0.87 and 0.94 with the presence of 2 outliers, one of which with a score of 1. The KNN in the training phase has brought better results for the Iris dataframe.

The distribution of the accuracy values in the test phase is better for the Iris dataframe, with a range that varies from 0.98 to 1. There are also 2 outliers of which the minimum is 0.93. Regarding the Wine dataframe, its values are in a range that varies between 0.79 and 0.85. There are 3 outliers of which a minimum with a value of 0.78 and a maximum of 0.89. The KNN in the test phase performs better on the Iris dataframe than the Wine dataframe, which performed worse than the training phase.

Considering the results obtained, we can affirm that the KNN performs better both in the training phase and in the test phase with the Iris dataframe.

Overall performance evaluation



AVG results plot

According to the results obtained, it is clear that the MDC performs better than the KNN.
In terms of accuracy, the best result was obtained with the MDC with Euclidean metric on the Iris dataframe, while the worst was obtained with the MDC with Chessboard metric on the Wine dataframe.
By averaging the results obtained by varying the distance metric on the MDC, the best result is obtained with the MDC with Euclidean metric and the worst is obtained with the MDC with Chessboard metric.
Based on the difference in the average accuracy value on the two data frames, the classifier that gave us the closest accuracy values for both data frames is the MDC with Cityblock metric and the one with the most different values is the KNN.

In our opinion, the MDC with Cityblock metric performs better as it has managed to provide more similar accuracy values than all the other metrics used for the MDC and KNN.

Bibliography

- Slides from Intelligent System course
- Slides from Data Science for Business course
- scikit-learn.org
- pandas.pydata.org
- plotly.com
- medium.com