# Homework 1: Classification Algorithms

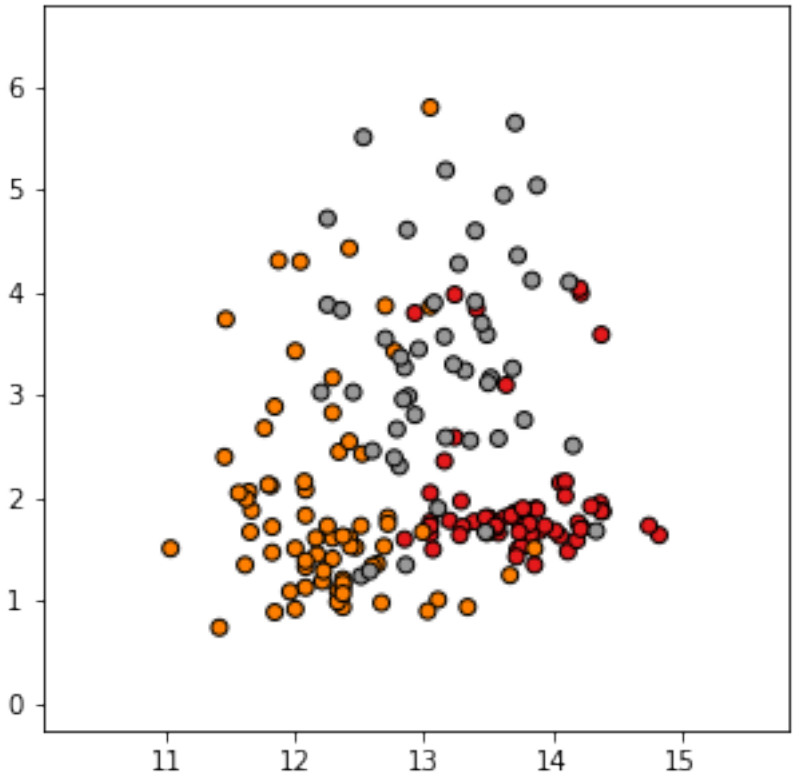
Lucia Innocenti

April 2020

## Contents

# 1 Introduction

The aim of this project is to apply different classification methods to the dataset "Wine" available in SK-Learn base datasets. As defined in the project-guide, for the analysis only the first and the second features are considered. After selecting these features, let's plot it in a bi-dimensional figure in which colors represents the different labels:




In order to better understand the dataset, we could print some statistics:

- nobs = 178
- minmax = [11.03, 0.74] , [14.83, 5.8 ]
- mean = [13.00061798, 2.33634831]
- variance = [0.65906233, 1.2480154 ]

Before starting the analysis, taking into account the statistics, it may be useful to apply some pre-processing methods to the data-set. Specifically, we can apply a standardization method:

$$X_{norm} = (X - X.mean())/X.std()$$

### 1.1 Performance Evaluation

After applying the different classification methods that we will see in this project, we need to understand which is the better one. So, I define that to evaluate the performances I will use these validations:

- Precision:
$$\frac{Total\_positive}{Total\_positive + False\_positive}$$
- Accuracy:
$$\frac{Total\_positive}{Total\_prediction}$$
- Sensitivity:
$$\frac{Total\_positive}{Total\_positive + False\_negative}$$
- F1_Score:
$$\frac{2 * Precision * Sensitivity}{Precision + Sensitivity}$$
- Confusion matrix: entry $i$, $j$ in a confusion matrix is the number of observations actually in group $i$ , but predicted to be in group $j$.

## 2 KNN Method

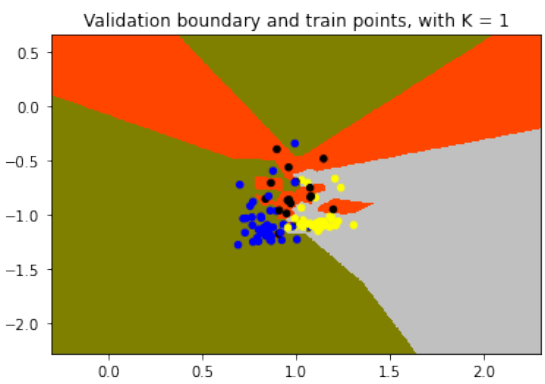
K-Nearest-Neighbors is a classification algorithm in supervised learning. It classifies the new points based on the class of the closest points to the considered one. This algorithm has one hyper-parameter: the value of K, that define the number of neighbors to consider in order to have the classification.

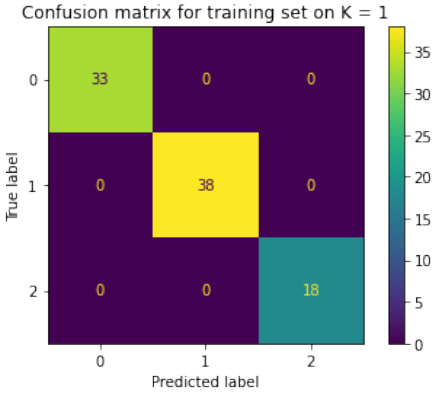
$$K \in \mathbb{N} \tag{1}$$

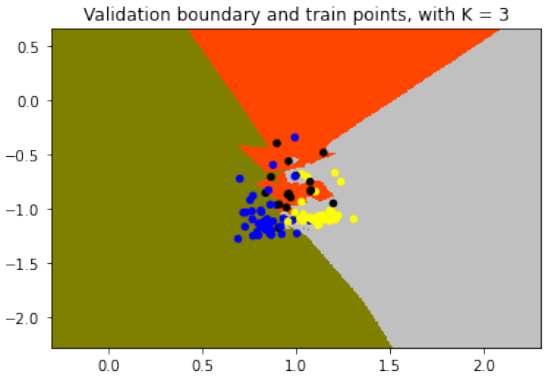
K must be an odd number in order to avoid uncertain classification.

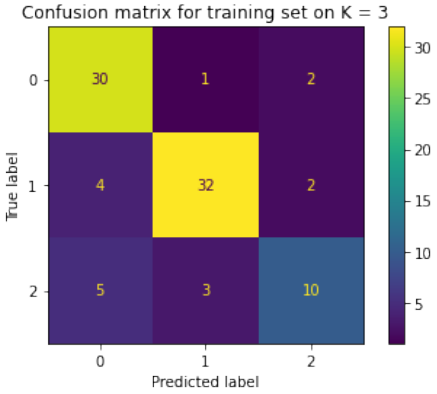
### 2.1 Training phase

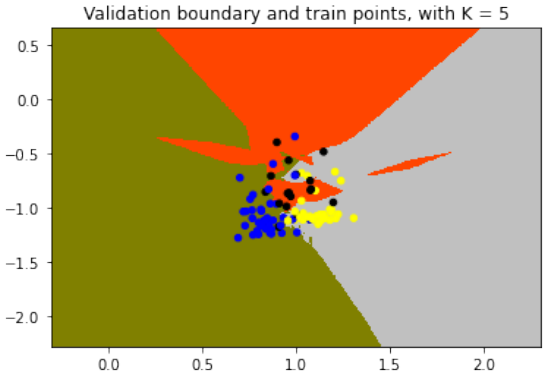
For K in $[1, 3, 5, 7]$, we will see how the test set and the evaluation set performs. After have trained the model, we will evaluate how it works on the validation set. In the next plots we can see that, changing the value of K, we obtain very different results both in train and validation set.

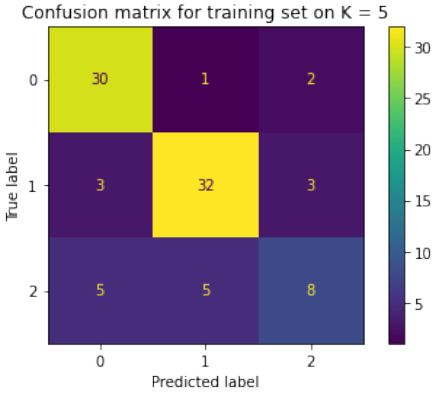

Validation boundary and train points, with K = 1
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.0
0.5
1.0
1.5
2.0

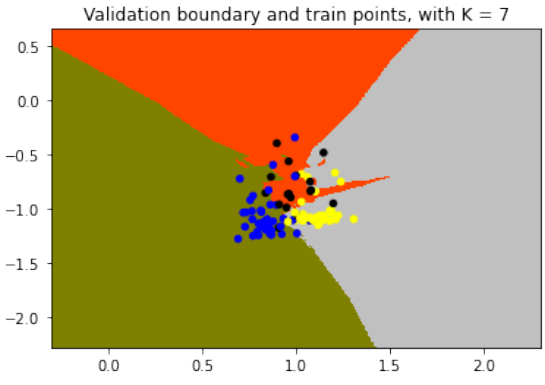

Confusion matrix for training set on K = 1
33
0
0
0
38
0
0
0
18
True label
Predicted label
0
1
2
35
30
25
20
15
10
5
0

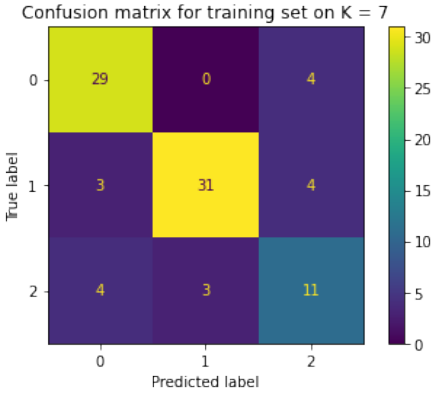

Validation boundary and train points, with K = 3
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.0
0.5
1.0
1.5
2.0


Confusion matrix for training set on K = 3
30
1
2
4
32
2
5
3
10
True label
Predicted label
0
1
2
30
25
20
15
10
5


Validation boundary and train points, with K = 5
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.0
0.5
1.0
1.5
2.0


Confusion matrix for training set on K = 5
30
1
2
3
32
3
5
5
8
True label
Predicted label
0
1
2
30
25
20
15
10
5


Validation boundary and train points, with K = 7
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.0
0.5
1.0
1.5
2.0


Confusion matrix for training set on K = 7
29
0
4
3
31
4
4
3
11
True label
Predicted label
0
1
2
30
25
20
15
10
5
0

In order to evaluate the performances on the validation set, at each loop I've saved the evaluation values and the confusion matrix. So, I can print both of them and see which is the best hyper-parameter for my set:

| | | |
|---|---|---|
| 12 | 1 | 0 |
| 2 | 14 | 1 |
| 2 | 1 | 2 |

Table 1: Confusion Matrix validation set: K = 1.
Accuracy = 0.80 F1_score = 0.79 Precision score = 0.798 Recall_score = 0.8

| | | |
|---|---|---|
| 13 | 0 | 0 |
| 1 | 16 | 0 |
| 0 | 1 | 4 |

Table 2: Confusion Matrix validation set: K = 3.
Accuracy = 0.94 F1_score = 0.94 Precision score = 0.94 Recall_score = 0.94

| | | |
|---|---|---|
| 13 | 0 | 0 |
| 1 | 16 | 0 |
| 2 | 1 | 2 |

Table 3: Confusion Matrix validation set: K = 5.
Accuracy = 0.88 F1_score = 0.87 Precision score = 0.90 Recall_score = 0.88

| | | |
|---|---|---|
| 13 | 0 | 0 |
| 1 | 16 | 0 |
| 1 | 1 | 3 |

Table 4: Confusion Matrix validation set: K = 7.
Accuracy = 0.91 F1_score = 0.90 Precision score = 0.92 Recall_score = 0.91

It's easy to see that the best model is the one having K = 3. So we will use it on the test set. An important thing to highlight is that the choice of K is very important both for accuracy and computation aspect: small value of k means that noise will have a higher influence on the result; large value make it computationally expensive and kinda defeats the basic philosophy behind KNN (that points that are near might have similar densities or classes)

## 2.2 Test phase

During the testing phase, I've decided to train the model with a set obtained by merging training and validation sets. Then I've trained the model with the hyper-parameters defined in the previous section

| | | |
|---|---|---|
| 11 | 0 | 2 |
| 1 | 12 | 3 |
| 6 | 3 | 16 |

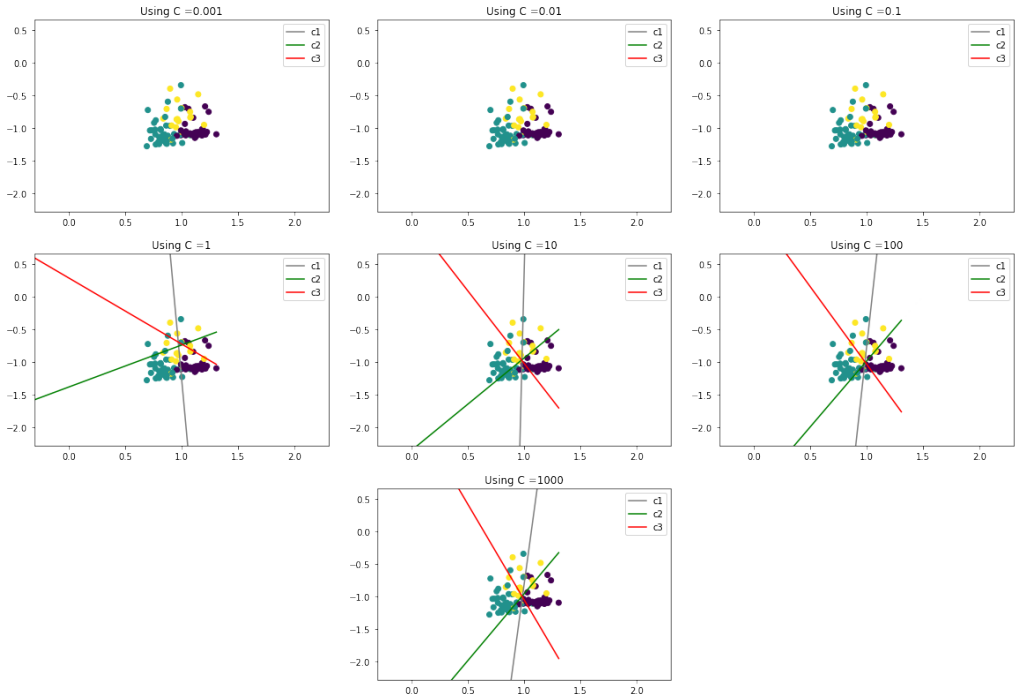
Table 5: Confusion Matrix test set
Accuracy = 0.72 Score = 0.72 ; precis = 0.74 ; recall = 0.72

# 3 SVM Linear Method

Linear Support Vector Machine (SVM) is a classification algorithm; it belongs to the supervised learning literature and it's goal is to find a hyper-plane in an N-dimensional space($N$ is the number of features) that distinctly classifies the data points.
Given training vectors $x_i \in \mathbb{R}^p i = 1, \ldots, n$, in two classes, and a vector $y \in \{1, -1\}^n$, our goal is to find $w \in \mathbb{R}^p$ and $b \in \mathbb{R}$ such that the prediction given by $\text{sign}(w^T \phi(x) + b)$ is correct for most samples. In the linear kernel version of SVM, the only hyper-parameter that need to be tuned is $C$, that is the Regularization parameter: it tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyper-plane if that hyper-plane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyper-plane, even if that hyper-plane misclassifies more points. For very tiny values of C, you should get misclassified examples, often even if your training data is linearly separable.

## 3.1 Training phase

For $C$ in $[0.001, 0.01, 0.1, 1, 10, 100, 1000]$, we will see how the test set and the evaluation set performs. After have trained the model, we will evaluate how it works on the validation set. In the next plots we can see that, changing the value of $C$, we obtain very different results both in train and validation set. The scatter points represent the train points, and the lines represent the hyper-planes found by the algorithm:




In order to evaluate the performances on the validation set, at each loop I've saved the accuracy and the confusion matrix on it. So, I can print both of them and see which is the best hyper-parameter for my set:

| | | |
|---|---|---|
| 0 | 13 | 0 |
| 0 | 17 | 0 |
| 0 | 5 | 0 |

Table 6: Confusion Matrix validation set: C = 0.001, C = 0.01, C = 0.1
Accuracy = 0.49 F1_score = 0.31 Precision score = 0.23 Recall_score = 0.48

| | | |
|---|---|---|
| 13 | 0 | 0 |
| 1 | 16 | 0 |
| 3 | 2 | 0 |

Table 7: Confusion Matrix validation set: C = 1.
Accuracy = 0.83 F1_score = 0.76 Precision score = 0.81 Recall_score = 0.83

| | | |
|---|---|---|
| 13 | 0 | 0 |
| 2 | 15 | 0 |
| 2 | 1 | 2 |

Table 8: Confusion Matrix validation set: C = 10.
Accuracy = 0.86 F1_score = 0.85 Precision score = 0.88 Recall_score = 0.86

| | | |
|---|---|---|
| 13 | 0 | 0 |
| 2 | 15 | 0 |
| 1 | 1 | 3 |

Table 9: Confusion Matrix validation set: C = 100.
Accuracy = 0.89 F1_score = 0.88 Precision score = 0.9 Recall_score = 0.88

| | | |
|---|---|---|
| 13 | 0 | 0 |
| 2 | 15 | 0 |
| 2 | 1 | 2 |

Table 10: Confusion Matrix validation set: C = 1000.
Accuracy = 0.86 F1_score = 0.84 Precision score = 0.88 Recall_score = 0.86

It's easy to see that the best model is the one having C = 100. So we will use it on the test set.

### 3.2 Test phase

During the testing phase, I've decided to train the model with a set obtained by merging training and validation sets. Then I've trained the model with the hyper-parameters defined in the previous section

| | | |
|---|---|---|
| 12 | 0 | 1 |
| 2 | 12 | 2 |
| 5 | 3 | 14 |

Table 11: Confusion Matrix test set
Accuracy = 0.70 F1_score = 0.70 Precision score = 0.73 Recall_score = 0.70

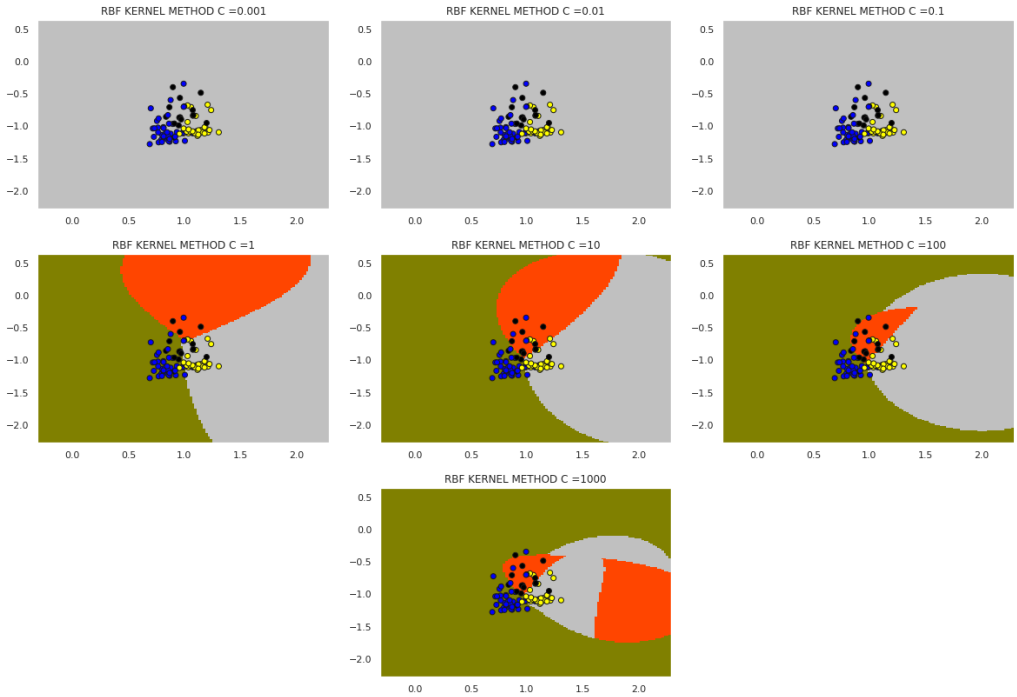
## 4 SVM RBF Kernel Method

The Radial basis function kernel, also called the RBF kernel, or Gaussian kernel, is a kernel that is in the form of a radial basis function (more specifically, a Gaussian function). The RBF kernel is defined as

$$K_{RBF}(x, x') = exp[-\gamma * \|x - x'\|^2] \tag{2}$$

Where $\gamma$ is an hyper-parameter that sets the "spread" of the kernel. Then, so long as $\gamma > 0$, it follows that $-\gamma * \|x - x'\|^2$ will be larger. Thus, closer vectors have a larger RBF kernel value than farther vectors. This function is of the form of a bell-shaped curve.
The $\gamma$ parameter sets the width of the bell-shaped curve. The larger the value of $\gamma$ the narrower will be the bell. Small values of $\gamma$ yield wide bells.
So, in our analysis, we have to set two hyper-parameters: $C$, as before, and $\gamma$; for them, values considered are:

$$C \in [0.001, 0.01, 0.1, 1, 10, 100, 1000]$$
$$\gamma \in [1, 10, 100, 1000]$$




## 4.1 Training Phase

First, we will look for the best value of C when $\gamma = \frac{1}{n * X_{var}}$ where $n = n\_features$
Then, we will apply a grid search both for $C$ and $\gamma$

Also in this case, let's take a look to evaluation methods:

| | | |
|---|---|---|
| 0 | 13 | 0 |
| 0 | 17 | 0 |
| 0 | 5 | 0 |

Table 12: Confusion Matrix validation set: C = 0.001, C = 0.01, C = 0.1
Accuracy = 0.49 F1_score = 0.31 Precision score = 0.23 Recall_score = 0.48

| | | |
|---|---|---|
| 13 | 0 | 0 |
| 1 | 16 | 0 |
| 3 | 2 | 0 |

Table 13: Confusion Matrix validation set: C = 1.
Accuracy = 0.83 F1_score = 0.76 Precision score = 0.71 Recall_score = 0.82

| | | |
|---|---|---|
| 12 | 0 | 1 |
| 1 | 15 | 1 |
| 2 | 1 | 2 |

Table 14: Confusion Matrix validation set: C = 10.
Accuracy = 0.83 F1_score = 0.82 Precision score = 0.82 Recall_score = 0.82

| | | |
|---|---|---|
| 13 | 0 | 0 |
| 1 | 16 | 0 |
| 1 | 1 | 3 |

Table 15: Confusion Matrix validation set: C = 100.
Accuracy = 0.91 F1_score = 0.84 Precision score = 0.85 Recall_score = 0.86

| | | |
|---|---|---|
| 13 | 0 | 0 |
| 1 | 15 | 1 |
| 2 | 1 | 2 |

Table 16: Confusion Matrix validation set: C = 1000.
Accuracy = 0.86 F1_score = 0.84 Precision score = 0.85 Recall_score = 0.86

Also in this case we can see that a small value for $C$ don't allow the model to recognize different classes: the margin is too high and includes all the points.The best performances are obtained with $C = 100$

## 4.2 Test Phase

During the testing phase, I've decided to train the model with a set obtained by merging training and validation sets. Then I've trained the model with the hyper-parameters defined in the previous section

| | | |
|---|---|---|
| 12 | 0 | 1 |
| 2 | 12 | 2 |
| 6 | 3 | 16 |

Table 17: Confusion Matrix test set: C = 100.
Accuracy = 0.74 F1_score = 0.74 Precision score = 0.77 Recall_score = 0.74

## 4.3 Hyper-parameter fitting

A heat map is a data visualization technique that shows magnitude of a phenomenon as color in two dimensions. In order to find the best pair $\gamma$ - $C$, I've trained the network on the (Train + Val) set and I have tested it on the test test. Then, by plotting the mean accuracy in an heatmap, I've selected the best parameters:
In this map, better values are represented by light colored squares. Let's see how the test set performs with $C = 10$ and $\gamma = 1$
The score of this combination is 0.76

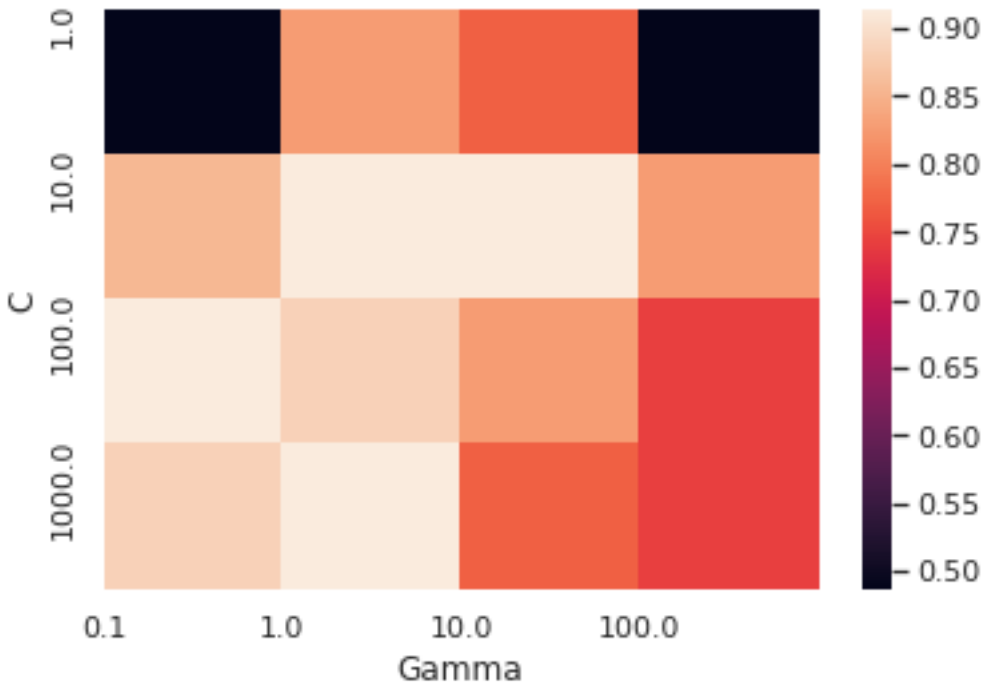



Figure 4: HeatMap for $C$ and $\gamma$

# 5 K-Fold Method

K-fold cross validation is one way to improve over the holdout method. The data set is divided into k subsets, and the holdout method is repeated k times. Each time, one of the k subsets is used as the test set and the other k-1 subsets are put together to form a training set. Then the average error across all k trials is computed. The best parameters are $C = 1$, $\gamma = 10$ with a score of 0.84

## 5.1 Test Phase

During the testing phase, I've decided to train the model with a set obtained by merging training and validation sets. Then I've trained the model with the hyper-parameters defined in the previous section

| | | |
|---|---|---|
| 12 | 0 | 1 |
| 2 | 12 | 2 |
| 4 | 5 | 16 |

Table 18: Confusion Matrix test set: $C = 1$, $\gamma = 10$
Accuracy = 0.74 F1_score = 0.73 Precision score = 0.75 Recall_score = 0.74

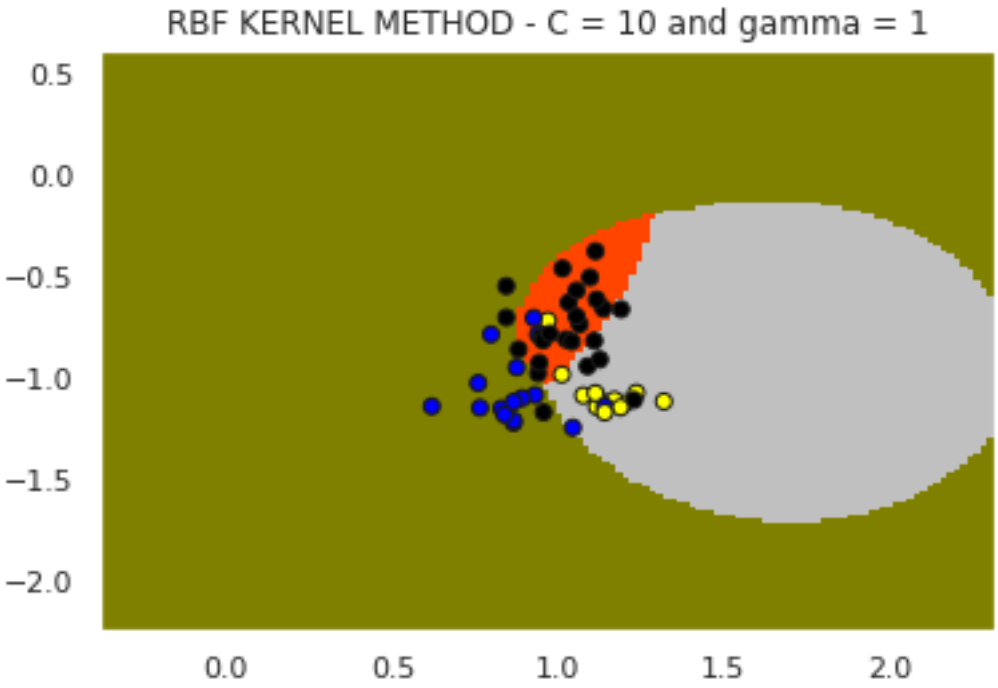



Figure 5: Decision Boundaries

## 6 Conclusions

The different models exposed provide comparable results. Usually, a SVM performs better than KNN when the dataset is separable by an hyperplane and a KNN performs better when the features considered are well identified by the features considered and that the classes are well separated from each other. In this case it's possible that the features selected (first and second from the original dataset) are not the best choice.

Because of the small dataset, and consequently the fact that classes don't have many elements, it might happen that in the split phase the train set will receive more elements from one class than one other and that the proportions among classes are misrepresented. In order to avoid it, I've modify my project by adding the attribute "stratify" to the train_test_split to be sure that classes proportionality was preserved on splits.

In order to verify if there are best pairs of attributes (i.e. that provides better results on the same tasks) I've applied Univariate feature selection from SK-Learn. This method selects ad the best features " Color intensity" [feature 10] and " Proline" [feature 13].

So I've repeated the analysis using them. Results are the following:

- KNN (with K = 1) performs worst than the previous 2 features
- Linear SVM (with C = 100) has an accuracy of 0.83, so it increase of 0.13 points.
- RBF Kernel SVM (with C = 100) has an accuracy of 0.87; with the older features the highest value reached was 0.74
- KFold has accuracy = 0.74, 0.4 points higher than the older configuration

The performances obtained by this model improvement are generally higher with respect to the other pairs, so the Feature selection has been a good choice.