Homework 1

By Walter Forcignanò
Candidate s262829
November 17, 2019

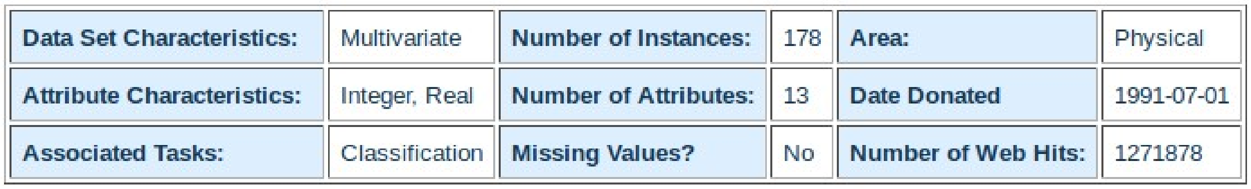
Contents

1 Introduction

The purpose of this homework is to train different kind of classificators based on the K-Nearest Neighbours and the Support Vector Machines, both with a linear and a radial basis function kernel, with different parameters, tuning them to produce representative models useful to classify future data.

1.1 Dataset

The dataset on which the models were trained is called wine. [1]
These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars and are characterized by 13 dimensions that represent quantities of 13 different constituents found in each of the three types of wines.



Data Set Characteristics:	Multivariate	**Number of Instances:**	178	**Area:**	Physical
Attribute Characteristics:	Integer, Real	**Number of Attributes:**	13	**Date Donated**	1991-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	1271878

Figure 1: Info about the dataset.

The instances belong to the three classes of wine with 59, 71 and 48 samples respectively.

The attributes are:

1. Alcohol
2. Malic acid
3. Ash
4. Alcalinity of ash
5. Magnesium
6. Total phenols
7. Flavanoids
8. Nonflavanoid phenols
9. Proanthocyanins
10. Color intensity
11. Hue
12. OD280/OD315 of diluted wines
13. Proline

1.2 Analysis

Among all the dimensions only the first two are considered in this analysis, in particular the quantity of Malic Acid and of Alcohol in the several wines, and for this reason the models that will be built, will infer the type of wine only according to these two features.

A 2D representation of the data with colors according to the classes is reported:

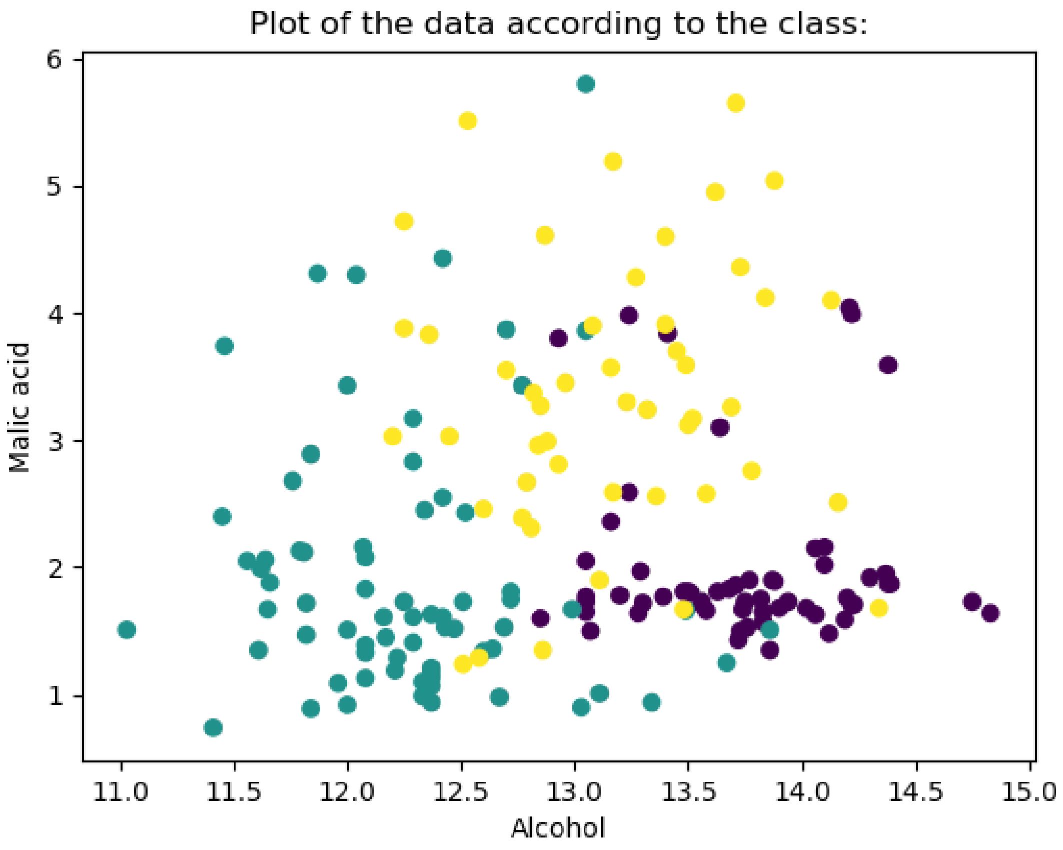


Figure 2: 2D representation of the data.

To do so, the first operation done on the dataset was to slice the data and then to plot them according to the following code:

```
import matplotlib.pyplot as plt
import numpy as np

X_slice = X[:,:2] #select the first two cloumns
X_plot = np.transpose(X_slice) #produce the transposed matrix

# Select the first two attributes for a 2D representation of the image.
plt.scatter(X_plot[0],X_plot[1],c=y) #scatters the points on the plot
plt.title("Plot of the data according to the class:")
plt.xlabel("Alcohol")
plt.ylabel("Malic acid")
plt.show()
```

1.3 Preprocessing

The data were splitted into train, validation and test sets in proportion 5 : 2 : 3 , and they were created with the option stratify in order to have a split that is more representative of the initial collection since in each set the proportion among the labels was saved.
The following piece of code shows how it was done:

```
from sklearn.model_selection import train_test_split

#split in train+validation and test sets -> 70:30
X_trainValidation,X_test, y_trainValidation, y_test =
train_test_split(X_slice, y, test_size=0.30, random_state=3, stratify=y)

#split in train and validation sets in order to have eventually a percentage of
# 50:20:30 respectively for train,validation and test sets.
X_train,X_Validation,y_train,y_Validation =
train_test_split(X_trainValidation,y_trainValidation, test_size=0.2857,
random_state=3,stratify=y_trainValidation)
```

Furthermore before the training phase, data are preprocessed by standardizing them with the purpose of putting different features on the same scale in order to avoid to be misled when computing the distance measure by the features with a bigger magnitude.
The distance is an important index, since it is useful for istance in the KNN model, to define the dissimilarity among the data and, in a way, in the definition of the label that needs to be assigned to the new incoming samples.

The following piece of code shows how the standardization was done:

```
#Standardization of data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
#Compute the mean and std to be used for later scaling.
scaler.fit(X_train)

#Perform standardization by centering and scaling
X_train = scaler.transform(X_train)
X_Validation = scaler.transform(X_Validation)

#Compute the mean and std to be used for later scaling.
scaler.fit(X_trainValidation)

#Perform standardization by centering and scaling
X_test = scaler.transform(X_test)
X_trainValidation=scaler.transform(X_trainValidation)
```

The relevant thing in this piece of code is that since the test set must be treated as unknow, it must be standardized according to the union of the train and the validation set (since the models at the end will be trained on both). According to the same principle the validation set is treated as unknown and for this reason scaled according to the train set.

2 K-Nearest Neighbour

2.1 Introduction to KNN

The K-Nearest Neighbour classifier is a model based on the simple idea that similar points will belong to the same class, or in other words nearby regions of space will be classified with the same label.
The label is assigned according to the majority among the ”votes” coming from the K Nearest-Neighbours.
For this classifier only two main parameters must be set and considered:

- K
- Distance metric

For the parameter K, most of the times, an odd number is used in order to avoid a draw among the votes when the problem is regarding a binary classification, but apart from this there is not any limitation on its number.
Anyway a too high value will produce a misleading classification since the points will be classified according to the most common class in your collection, on the other hand a too low value will be measleading as well, since the classification can be affected by noise and outliers.
For what concern the distance function to be used, since it's a measure of dissimilarity, it will define how much two samples can be considered similar and obviously this will impact in the definition of the ”nearest” neighbours to be considered in the vote.

2.2 Model Generation and Validation

In this step, the purpose is to generate different KNN models by tuning their parameters in order to select the best one.
As said for the KNN classifier, the only parameters to be set are K and the distance function. The proposed paramaters were $K = [1, 3, 5, 7]$ and among all the distance measures it was used the most common one, the euclidean metric.
The selection among the classifiers is based on their accuracy, in other words their ability to correctly classify the data coming from the validation set.
For each model the decisions boundaries were plotted showing also how they were built by scattering the training points on them as reported on the image:

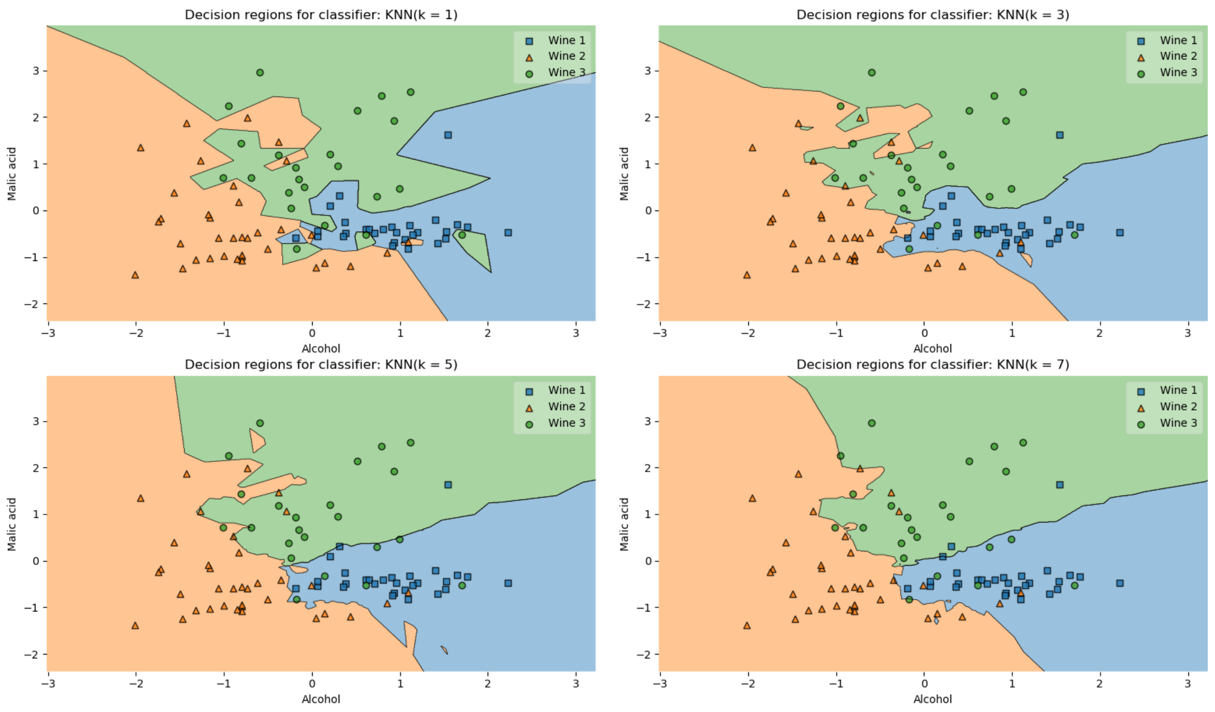


Figure 3: Decision Regions of KNN models.

As it can be seen in the image 3, the decision boundaries change according to the parameter K. A more regular shape is obtained for a number of $K >= 3$, and this is due to the fact that with an higher number K, on the boundaries, portion where you can have some noise, the most common class is usually selected as label and a smoother effect on the representation comes as result. On the other hand with a number $K = 1$, the boundaries are more affected by the noise and outliers, this can be notice by comparing the different shapes of the boundaries obtained by this classifier with respect to the others, and an example can be seen in some points like the ones in the blue region, below on the right in the plot related to the model with K=1 in the figure 3, where some green points completly surronded by the blue area can be spotted out.

In the succeeding step, the evaluation one, the model fitted on the train set is evaluated on the validation one, taking into consideration its accuracy in correctly classify the incoming points.
An output of the program highligthing the different accuracies obtained and the corresponding graphs of the accuracy and the scattered points are reported here:

```
The accuracy of the model with k equal to 1 is: 0.778
The accuracy of the model with k equal to 3 is: 0.833
The accuracy of the model with k equal to 5 is: 0.806
The accuracy of the model with k equal to 7 is: 0.833
```

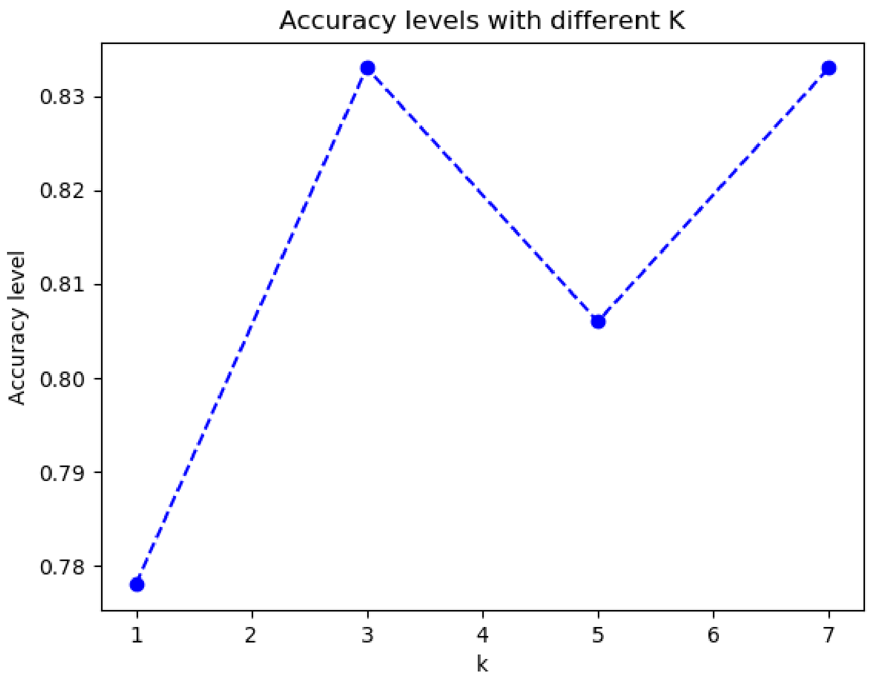


Figure 4: Accuracy KNN models.

By looking to the accuracy plot, it can be spotted out that for $K >= 3$ the accuracies are in the range 80%-83% with a minimum in $K = 5$.
It cannot be excluded that this little margin comes from the casuality of how the split were generated.

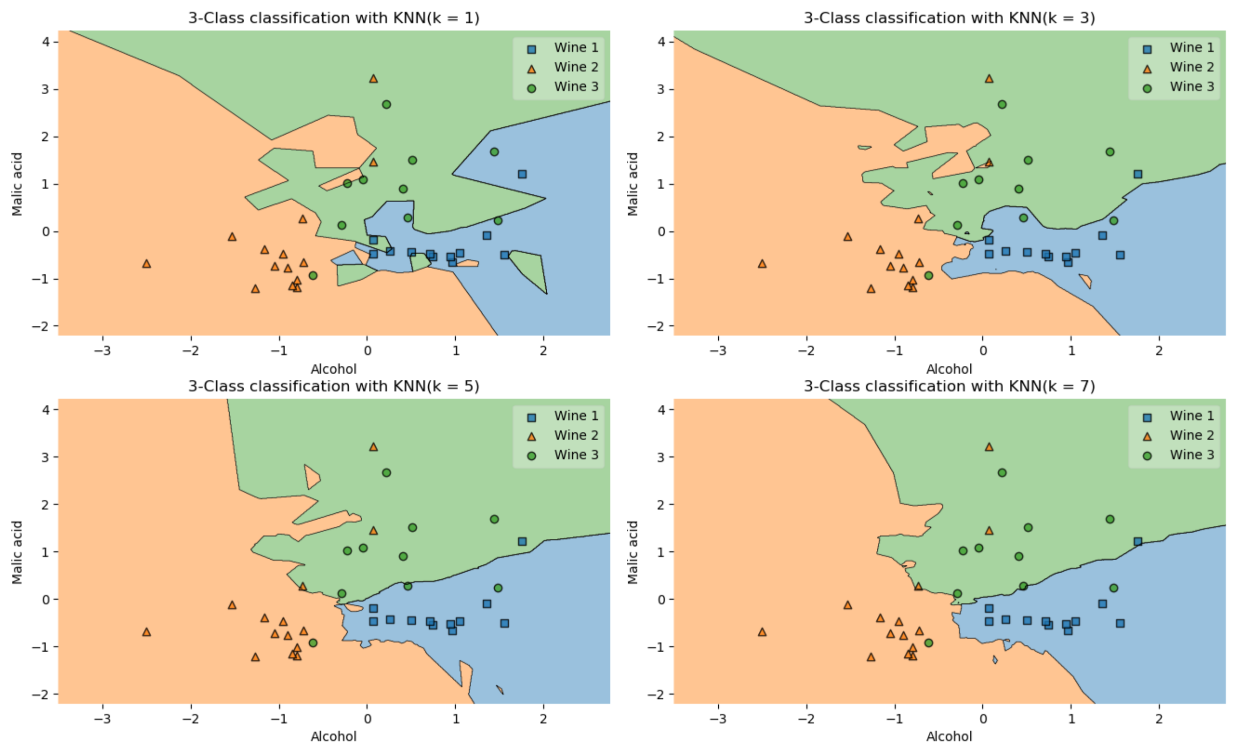


Figure 5: KNN models evaluation.

On the last image the validation set was scattered in order to have an idea of how well the different models classified correctly the points.

The classifiers with $K = 7$ and $K = 3$ performed better in terms of accuracy on the validation set, but no cross validation procedure was done so the estimation of the skill of the models really depends on how the splits were generated, so it can be overestimated.
According to the algorithm reported in the following section, the last optimum solution is selected and the classifier with K equal to 7 is in this situation selected.

```
#search of optimum value for K
#key => K
#value => accuracy
for key,value in accuracylist:
    if(value>=bestV):
        bestV = value
        bestK = key
```

By running the program with different splits, obviously the accuracies will change, so to have a less biased estimation of the model accuracy, the selection of the best parameter K should be done following a cross validation approach.
Since this procedure was not required in this step, a local optimal solution was selected, but even so, this doesn't guarantee that the best parameter K will always be the one selected in this situation.

Once the best parameter is found and selected according to the score obtained by the model on the validation set, a new model is built, fitted on both the training and the validation sets and then eventually evaluated on the test set.
Here is reported an image of the final decisions boundaries and an output highligthing its accuracy:

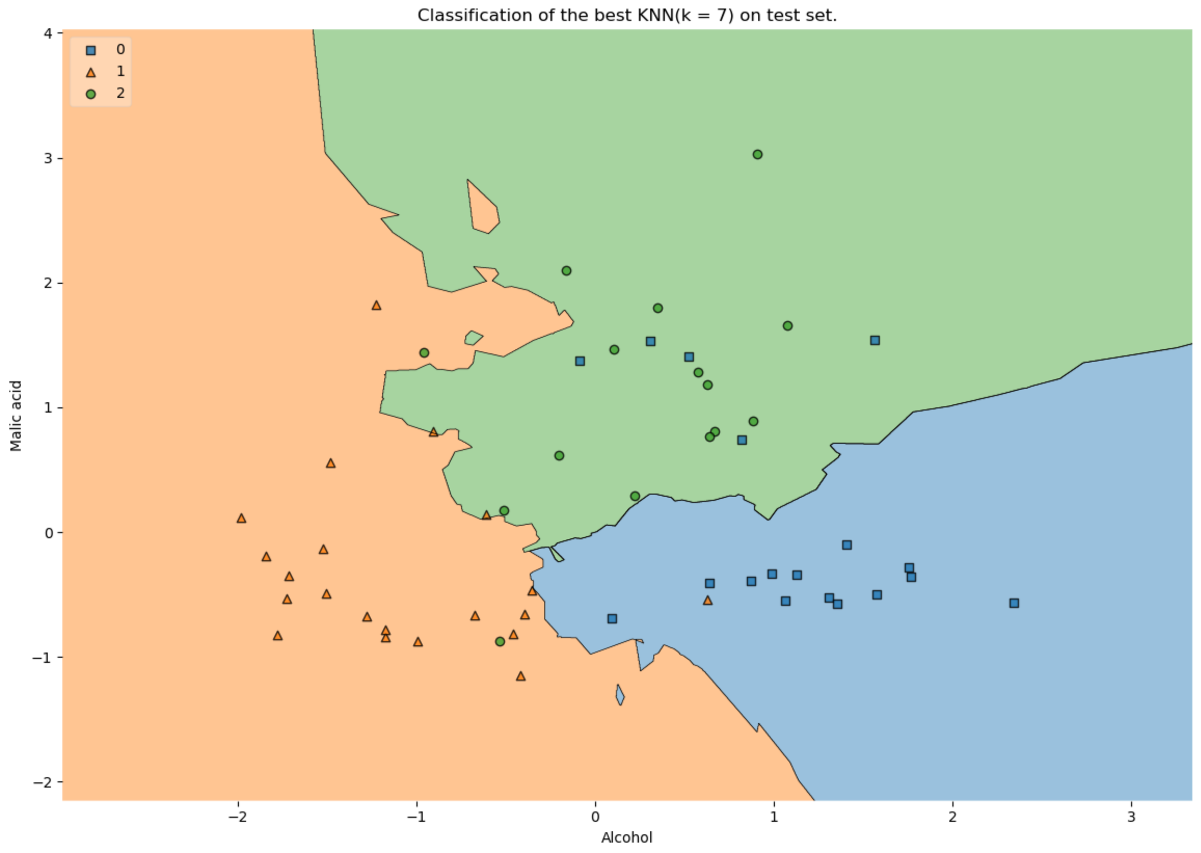


Figure 6: Evaluation of the KNN with the best parameter on test set.

The KNN model with k = 7 has an accuracy on the test set up to: 0.815

In the figure 6, the test set was scattered in order to have an idea of how well the model correctly classified the points.
The accuracy obtained is aligned with the ones obtained in the validation phase, but this is in a way biased from the fact that the test comes exactly from the same collection of data.
It can be notice that the decision regions are different with respect to the ones generated in figure 3, because the model was fitted on the entire train set.

3 Support Vector Machine

3.1 Introduction to SVM

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes the samples on which it was trained and can be used also to classify new data.
In a two dimentional space this hyperplane is a line dividing a plane in two parts where the points are separated according to the class label.
Intuitively, a good separation is achieved by the hyperplane that has the largest distance from the nearest training data point of any class, since in general the larger the margin, the lower the generalization error of the classifier.
Whereas the original problem may lay in a finite-dimensional space, it often happens that the data to discriminate are not linearly separable in that space.
For this reason, the original finite-dimensional space can be mapped into a much higher-dimensional one, presumably making the task of discerning points according to the classes, a linearly separable problem.
These new high dimensional spaces are designed in such a way to ensure that dot products of pairs of input data vectors, can be easily expressed in terms of the kernel function $k(x, x')$ selected to suit the problem.
Furthermore since the samples in our case are characterized by 3 different classes the SVMs generated were Multiclass ones and this means that for each possible split in two groups of the original classes the Multiclass SVM generates a line that separates the first group characterized by a particular label from the rest of the collection with different labels.

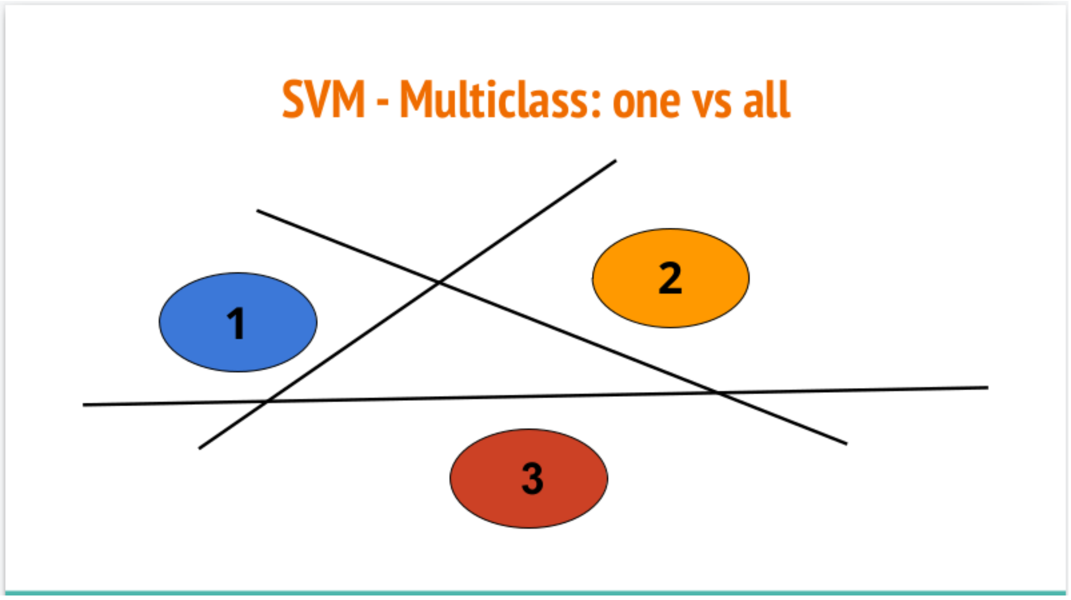


Figure 7: SVM-multiclass

3.2 Model Generation and Validation

The purpose here was to build different SVMs with different parameters and to evaluate the best ones on the validation set.

3.2.1 Linear Kernel:

Since the samples are not linearly separable (as can be seen from the image 2), the model was built using **soft** margins.
This idea is based on a simple premise: allow SVM to make a certain number of mistakes and keep margin as wide as possible so that other points can still be classified correctly. [2]
This can be done by modifying the objective of SVM with the same aim of minimizing it:

$$L = \frac{1}{2}\|w\|^2 + C(\#ofmistakes)$$

This differs from the original objective in the second term.
The hyperparameter C is introduced in the linear SVC and can be seen as a cost of misclassification.
The values on which the classifiers were trained are:

$$C = [0.001, 0.01, 0.1, 1, 10, 100, 1000]$$

This hyperparameter decides the trade-off between maximizing the margin and minimizing the mistakes. When C is small, classification mistakes are given less importance and focus is more on maximizing the margin, whereas when C is large, the focus is more on avoiding misclassification at the expense of keeping the margin small.
The models were tuned according to the different values of the parameter C and at each step a visualization of the accuracies and of the decision boundaries were reported in order to evaluate the model and to see how did they change.
At the end the best model fitted on the train and validation sets built with the best parameter C, was evaluated on the test set.

The models were built with the following piece of code:

```
from sklearn.svm import SVC

C = [0.001, 0.01, 0.1, 1, 10, 100,1000]

model=[] # vector of the clfs (classifiers)

for i in C :
    model.append(SVC(C=i,kernel='linear',max_iter=10**9)) #definition of the clfs
```

After the creation the models were fitted on the train set and the obtained decision regions are reported in the following image:

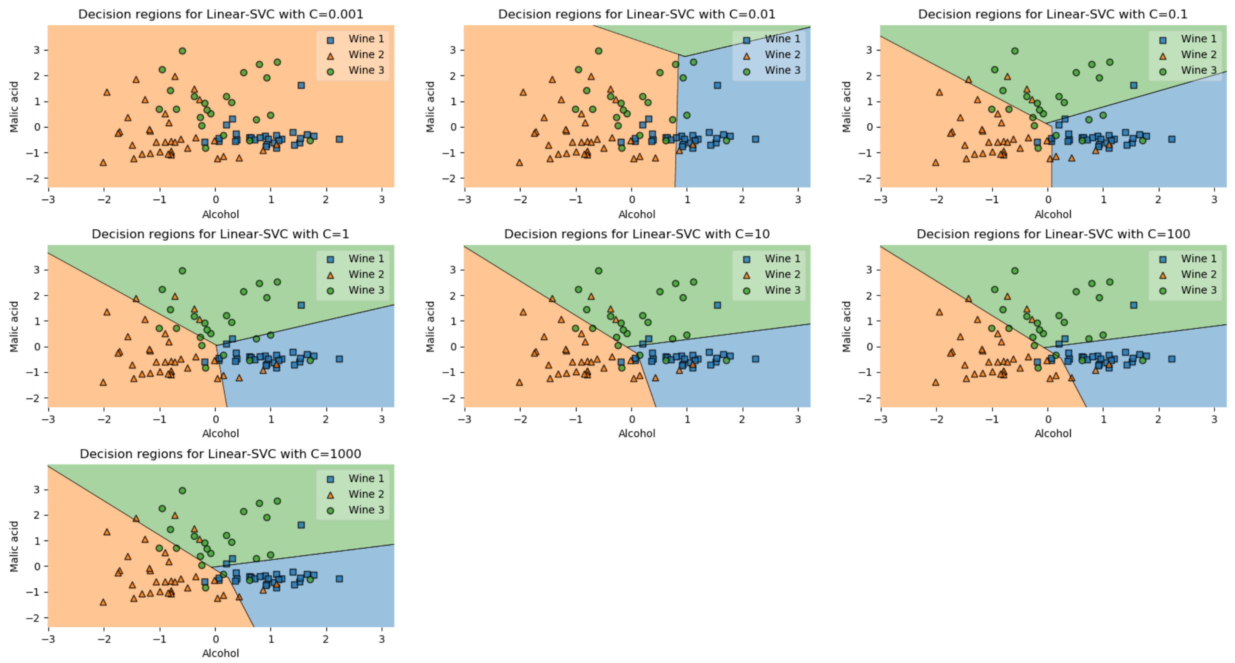


Figure 8: Decision regions of Linear-SVC.

As can be seen from the figure 8, for different values of C the decision boundaries change a lot. As said before, for low values of the misclassification cost its impact in the definition of the boundaries is really negligible, indeed in particular for $C = 0.001$ the classification is completly wrong, but also for $C = 0.01$ the classification doesn't work very well since the boundaries are defined but most of the points belonging to the third class are not correctly classified.
Whereas for values of $C >= 0.1$ the decision boundaries are defined in such a way that the misclassification is minimized, since more and more importance is given to this element facing on the other hand the risk of overfitting.
Nevertheless the higher the value the less are the number of points that are not correctly classified reaching more or less stable boundaries for $C >= 10$.
The boundaries obtained for this classifier are completly different with respect to the one obtained for the KNN, due to the fact they are built in different ways.
Here the boundaries are represented by straight lines, instead the boundaries of KNN have not this kind of behaviour.

In order to choose and so define the "best" model, an evaluation on the validation set was done and the accuracies for different values of C are reported in the accuracy plot and in the output below:

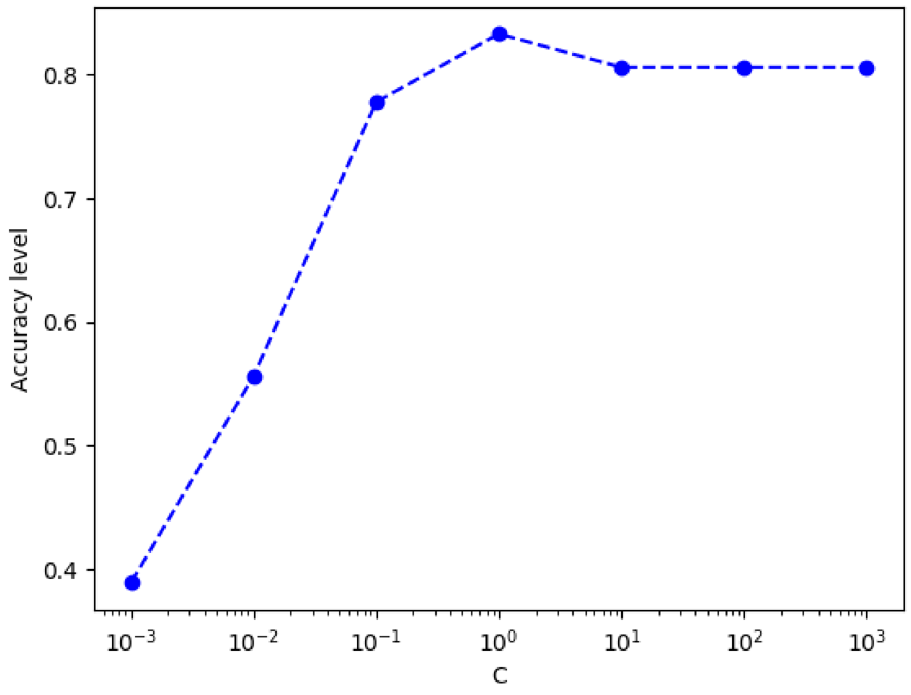


Figure 9: Accuracies of Linear-SVC

```
The accuracy of the model with C equal to 0.001 is: 0.389
The accuracy of the model with C equal to 0.01 is: 0.556
The accuracy of the model with C equal to 0.1 is: 0.778
The accuracy of the model with C equal to 1 is: 0.833
The accuracy of the model with C equal to 10 is: 0.806
The accuracy of the model with C equal to 100 is: 0.806
The accuracy of the model with C equal to 1000 is: 0.806
```

According to the figure 9 can be seen that only for value of $C >= 0.1$ the algorithm is behaving in a good fashion, reaching good values of accuracies. Whereas with lower values of C the accuracy reached is pretty low, 55% in the best case.
The optimal value of accuracy is found where $C = 1$ and this means that it is a good trade-off between maximizing the margin and minimizing the mistakes, while for higher values the accuracy stays more or less stable on 80%.

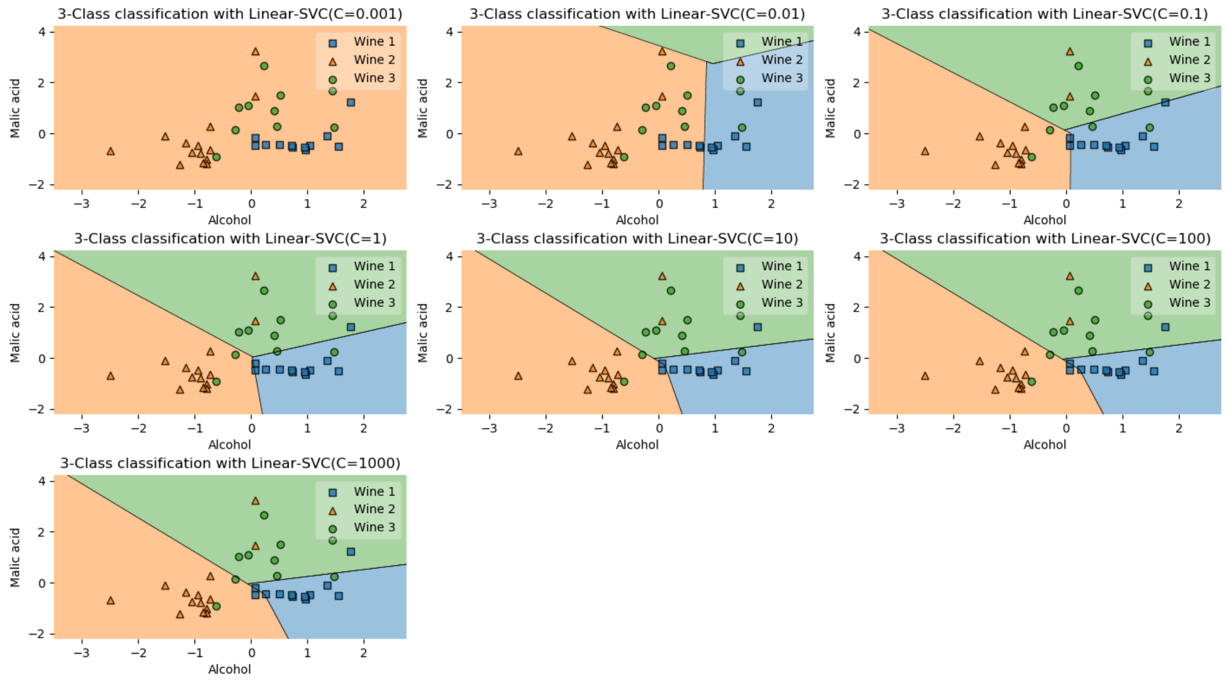


Figure 10: Linear-SVC evaluation on the validation set.

In figure 10, the validation set was scatter to have a better idea of how well the different models classified the points.

Once the best parameter is found and selected according to the score obtained by the model on the validation set, a new model is built, fitted on both the training and the validation sets and then eventually evaluated on the test set.
Here is reported an image of the final decisions boundaries and an output highligthing its accuracy:

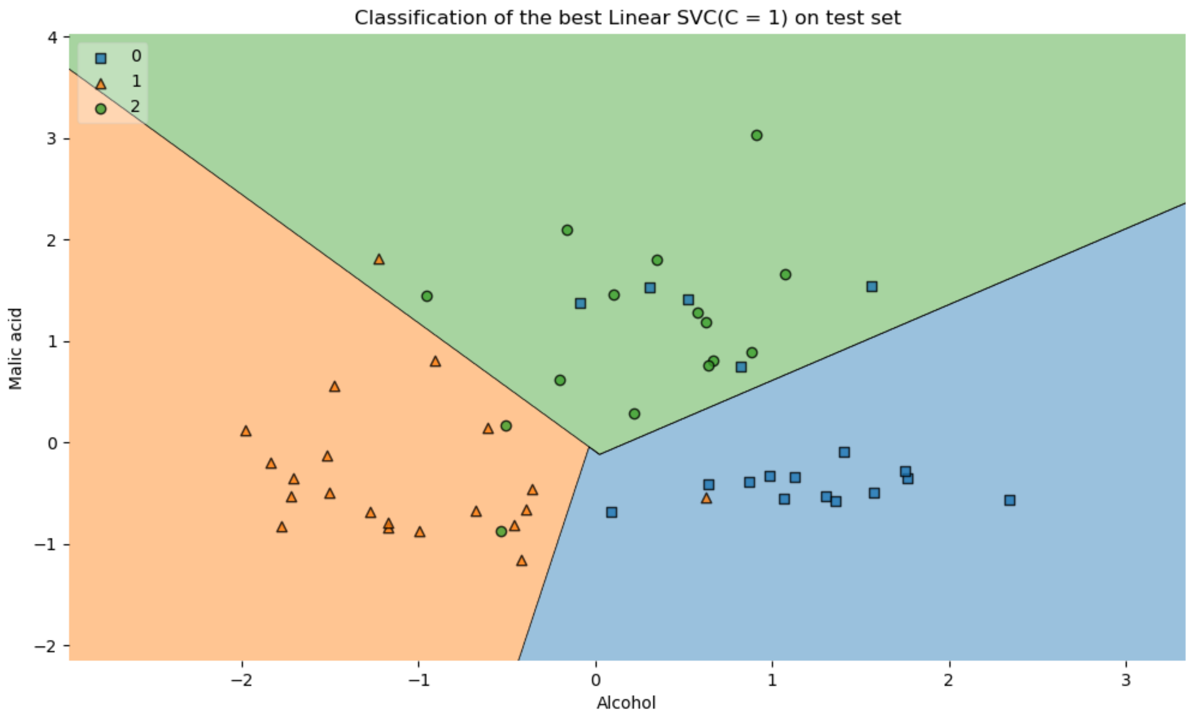


Figure 11: Linear-SVC with test set

The SVM built with C = 1 has an accuracy on the test set up to: 0.833

In the figure 11, the test set was scattered in order to have an idea of how well the model correctly classified the points.
It can be notice that the decision regions defined in the last plot are different from the ones generated in figure 8, since the classfier was trained on the entire train set.

3.2.2 Radial basis function Kernel

The same steps are faced for the SVM with a rbf kernel [3] defined as:

$$K(x, x') = \exp(-\gamma \|x - x'\|^2)$$

Using this kernel, one more parameter γ needs to be set. At the beginning γ will be automatically set by the algorithm itself, then when it's required a grid search we will tune both the hyperparameters C and γ.

According to the sklearn library [4] $\gamma = \frac{1}{\#features}$, so in this case it's equal to 0.5.

The models were built using the same values for the parameter C used for the creation of the Linear SVC, and at each step a visualization of the decision boundaries and some highligths of the accuracies obtained are reported.
The models were built according to the following piece of code:

```python
from sklearn.svm import SVC
C = [0.001, 0.01, 0.1, 1, 10, 100,1000]
model=[] # vector of the clfs (classifiers)
for i in C :
    model.append(SVC(C=i,kernel='rbf')) #definition of the clfs
```

After the creation the models were fitted on the train set and the obtained decision regions are reported in the following image:

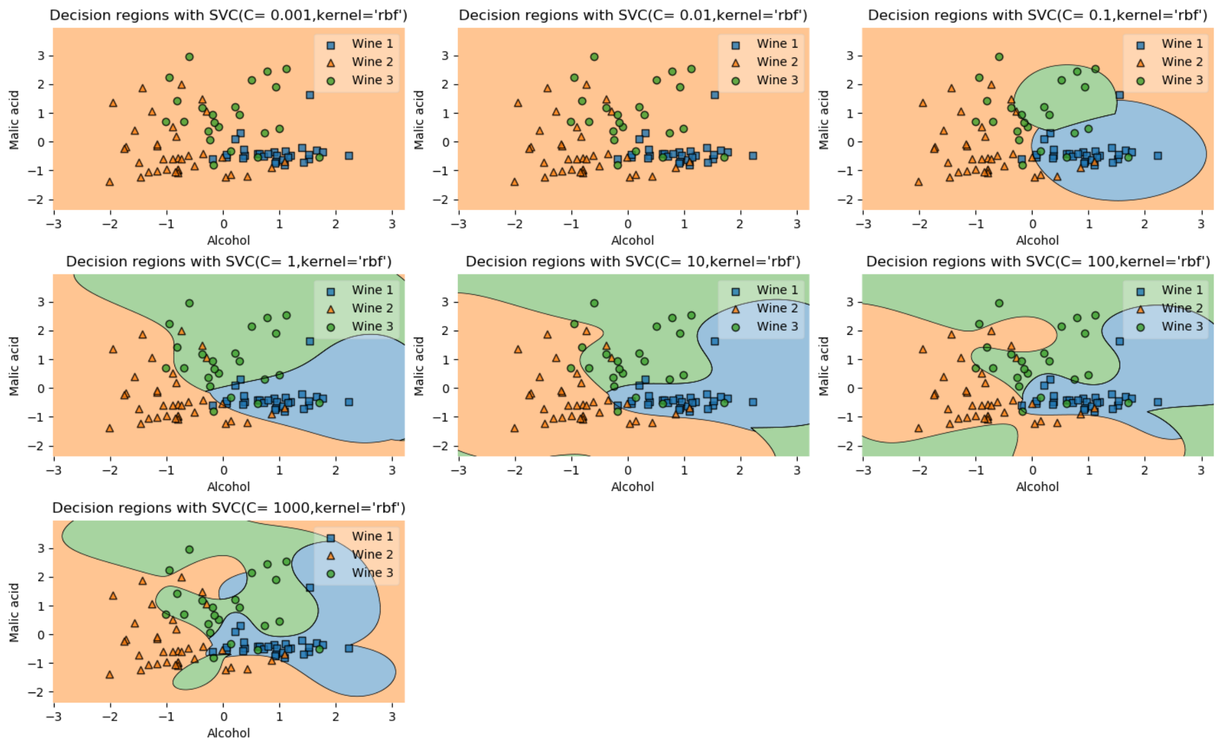


Figure 12: Decision regions of rbf-SVC.

With respect to the linear kernel obviously here the boundaries are a bit different in terms of shape, indeed if the separator before were lines, here the boundaries have a more globular shape, due to the usage of the radial basis function as kernel.
As can be seen from the image 12, the model starts to create representative decision boundaries only for values of the hyperparameter $C >= 0.1$.
This can be explained as before in the Linear Kernel, since the parameter C represents the cost of the misclassification and its value for the first two plots is really low, its impact on the definition of the decision boundaries is really poor, indeed the model doesn't worry about misclassifing the points coming from class 1 and class 3.
On the other hand, when the value of C becomes huge, for the model misclassifing a point has big cost, for this reason it tends to correctly classify as more as possible the points, probably doing overfitting. This phenomenon can be registered for a value of $C >= 100$.

The models were then evaluated on the validation set and the reported image is in the following section:

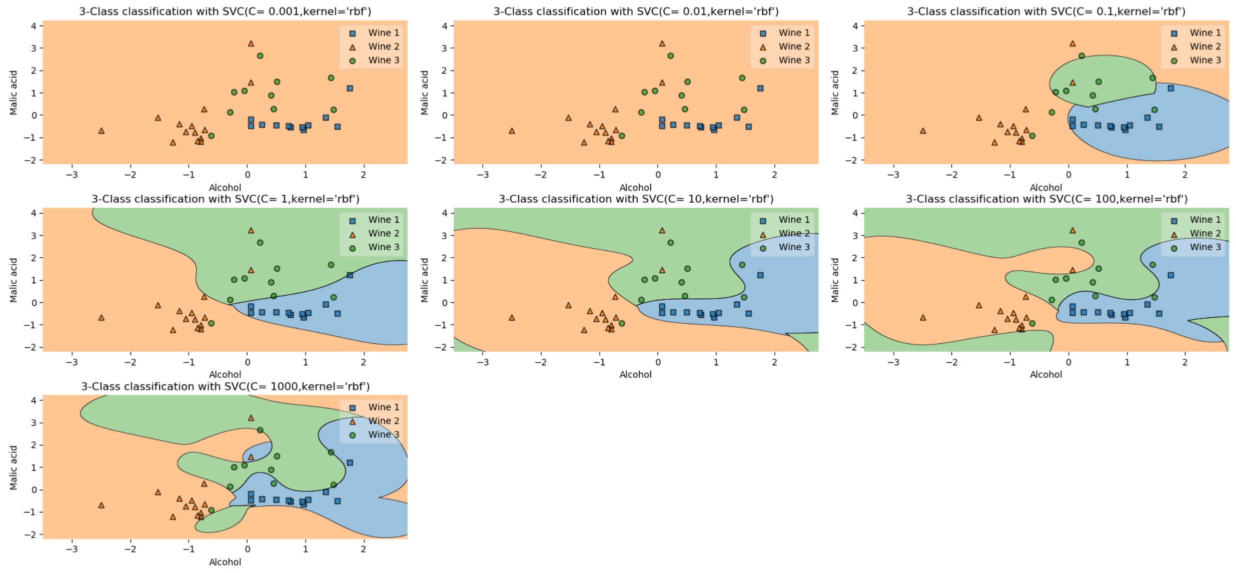


Figure 13: Classification of rbf-SVC built with only parameter C on validation set.

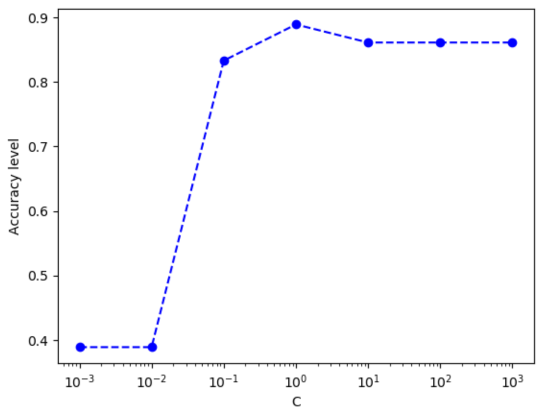


Figure 14: Accuracy plot of rbf-SVC built with only parameter C.

```
The accuracy of the model with C equal to 0.001 is: 0.389
The accuracy of the model with C equal to 0.01 is: 0.389
The accuracy of the model with C equal to 0.1 is: 0.833
The accuracy of the model with C equal to 1 is: 0.889
The accuracy of the model with C equal to 10 is: 0.861
The accuracy of the model with C equal to 100 is: 0.861
The accuracy of the model with C equal to 1000 is: 0.861
```

As can be seen in the image 13, the classificator seems to have a good behaviour in correctly classifying the labels only for values $C >= 0.1$ and this is also confirmed from the output of the program and from the plot of the accuracies reported in image 14.
Indeed the accuracies obtained remark this phenomenum: for $C >= 0.1$ the mean accuracy is 86% with an optimum 88.9% obtained for $C = 1$ and a "worst" 83% obtained instead for a value $C = 0.1$.

The model that scored the highest accuracy, the one with $C = 1$ was selected to be evaluated on the test set. For this reason it was trained according to the entire train set, considered also the validation one, and then scored on the test set.

An output highligthing the accuracy obtained and a plot are reported here:

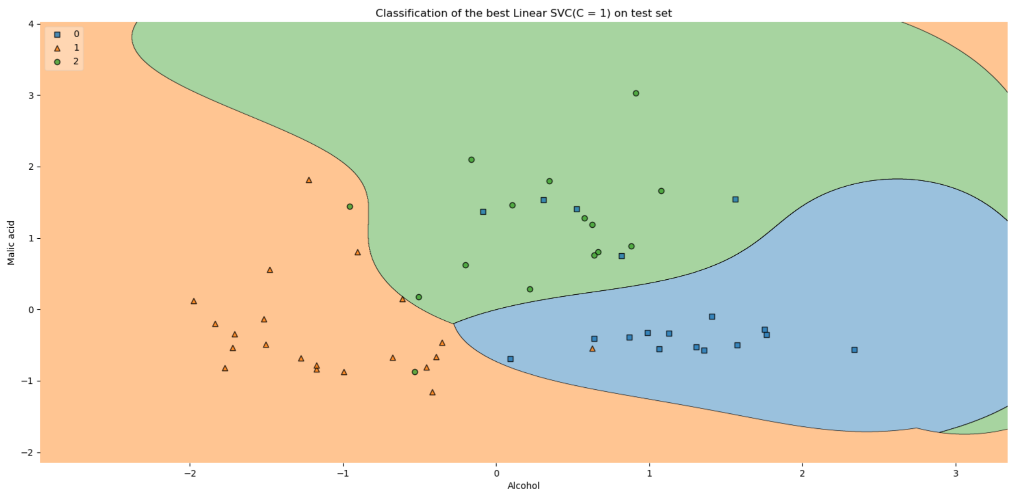


Figure 15: Best SVC with rbf kernel evaluated on the test set

---

The SVM built with C = 1 has an accuracy on the test set up to: 0.852

The final score obtained on the test is aligned to the ones obtained on the validation set, but it is possible to notice that the decision boundaries are slightly different with respect to the ones produced in the evaluation on the validation set, because the algorithm is now trained on both the training and the validation sets.

3.2.3 Grid Search

The purpose of this section is to tune both the hyperparameters C and γ.
The hyperparameter C has the same meaning as in the previous cases, it's the cost of misclassifing points, γ instead intuitively defines how far the influence of a single training sample reaches, with low values meaning 'far' and high values meaning 'close'. The gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors.

The values proposed for the parameter C were the same used in the creation of the previous classificators, instead for γ there was the freedom to assign any values.

The chosen values were:

$$C = [0.001, 0.01, 0.1, 1, 10, 100, 1000]$$

$$\gamma = [10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10, 100, 1000]$$

A large spectrum was chosen also for γ in order to extract as much information as possible from the behaviour of the models.

The same steps as in the previous sections were faced so the models were firstly built fitted on the train set, evaluated on the validation set and the best model evaluated also on the test set.
The model are built and then the grid search is done according to the following piece of code:

```
C = [0.001, 0.01, 0.1, 1, 10, 100,1000]

gamma = [10**(-8),10**(-7),10**(-6),10**(-5),10**(-4),10**(-3),
        10**(-2),10**(-1),10,100,1000]

for c in C:
       for g in gamma:
           model.fit(X_train,y_train) #training phase

           #accuracy obtained on validation set
               currentValue = round(model.score(X_Validation,y_Validation),3)

              # save the best values of C and gamma
               if currentValue > optimalAccuracy:
                      optimalAccuracy = currentValue
                      bestC=c
                      bestG=g
```

The output of the grid search is really long since 77 different models are created and evaluated on the validation set, for this reason it was decided to not report such a long log here, but it can be checked by simply running the python script or it can be seen in the file `log.txt`. From the output it is possible to notice that for $C <= 0.1$ regardless of the value of γ the accuracy of the model is 39% and this is due to the fact a too low value of C corresponds to a too low cost of misclassification and this will make the model misclassifing regardless of the value of γ. The behaviour starts to change when $C = 1$ indeed here the accuracy is raised up for values of γ in the range 10^{-2} - 10 with a mean accuracy of 77%.

```
The SVM built with C = 1 and gamma = 0.01 scored an accuracy: 0.694
The SVM built with C = 1 and gamma = 0.1 scored an accuracy: 0.806
The SVM built with C = 1 and gamma = 10 scored an accuracy: 0.806
The SVM built with C = 1 and gamma = 100 scored an accuracy: 0.694
```

The higher the value of C, the wider it's the range of γ in which the mean accuracy increase:

```
The SVM built with C = 10 and gamma = 0.001 scored an accuracy: 0.694
The SVM built with C = 10 and gamma = 0.01 scored an accuracy : 0.806
The SVM built with C = 10 and gamma = 0.1 scored an accuracy: 0.889
The SVM built with C = 10 and gamma = 10 scored an accuracy: 0.806
The SVM built with C = 10 and gamma = 100 scored an accuracy: 0.722
The SVM built with C=1000 and gamma=1e-05 scored an accuracy: 0.694
The SVM built with C=1000 and gamma=0.0001 scored an accuracy: 0.806
The SVM built with C=1000 and gamma=0.001 scored an accuracy: 0.778
The SVM built with C=1000 and gamma=0.01 scored an accuracy: 0.889
The SVM built with C=1000 and gamma=0.1 scored an accuracy: 0.861
The SVM built with C=1000 and gamma=10 scored an accuracy: 0.806
The SVM built with C=1000 and gamma=100 scored an accuracy: 0.722
```

It can be seen that the behavior of the model is very sensitive also to γ. If γ is too large for instance $\gamma = 1000$, the radius of the area of influence of the support vectors only includes the support vector itself and no value of the hyperparameter C will be able to prevent overfitting. When gamma is very small, the model is too constrained and cannot capture the complexity of the data. The region of influence of any selected support vector would include the whole training set, so the resulting model behaves similarly to a linear model with a too low value of C, so it's not able to capture the shape of the data.
The optimum is found in two couples:

1. $C = 10$ and $\gamma = 0.1$
2. $C = 1000$ and $\gamma = 0.01$

Both of them reached the maximal accuracy with a value of 89%, but among the two, the first solution, the one characterized by $C = 10$ and $\gamma = 0.1$, was used to be evaluated on the test set.

Both the plots, the one with the validation set scattered on it, and the one with the test set, were reported here in order to have a better idea of how the decision boundaries changed and of how the model performed in the classification:

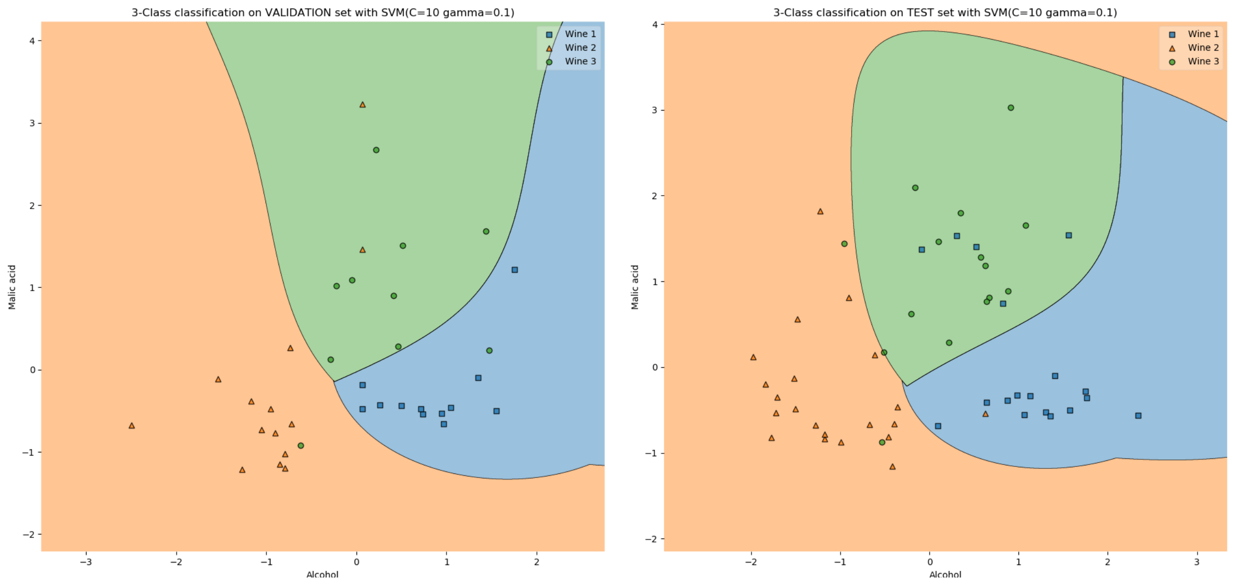


Figure 16: Best SVC with rbf kernel evaluated on the test set

As can be notice in the image 16, the boundaries have changed, and this can be explained from the fact that the model before being evaluated on the test set, is fitted on the entire train set, so considering also the validation one. On the other hand it can be seen that an higher number of points of the test were misclassified by the model with respect to the one in the validation set, so a little bit lower accuracy should be expected.
An output highligthing the accuracy of the model is reported:

```
The model scored on the test an accuracy equal to: 0.852
```

The fact that we obtained a little lower accuracy can be also explained by the fact that the evaluation, done on the validation set, was in a way biased from how the sets were built.
This could have possibly been avoided with a cross validation procedure, since the accuracy of the model would have been tested more times with different portions as validation set and a more significant value, a mean accuracy, would have been extracted as final accuracy.

4 K-Fold

The purpose of this section is to build a final SVC model with a kernel based on the radius basis function, fitted on the entire train set, tuning its parameters with a cross validation procedure. Once the best model was created, it was required to evaluate it on the test set. An output highlighting which were the parameters used to create the model is reported here:

The model was created with C = 1000 and gamma = 0.01

It can be notice that the couple chosen in the grid search is the second solution that reported an optimum in the previous grid search without a cross validation approach.
Previously the couple chosen was the first one, since according to the algorithm, the first best solution had to be taken in consideration. Here instead the model was built with this couple of parameters since the grid search with a cross validation procedure makes the model, built with the parameters, be tested more than one time on different portions of the train set, considered as validation ones. The results brought to the conclusion that the second couple of hyperparameters was able to make the model better in classifying samples.
Furthermore, the latter solution is more robust thanks to the cross validation procedure.

The model was then evaluated on the test set and an output is reported here:

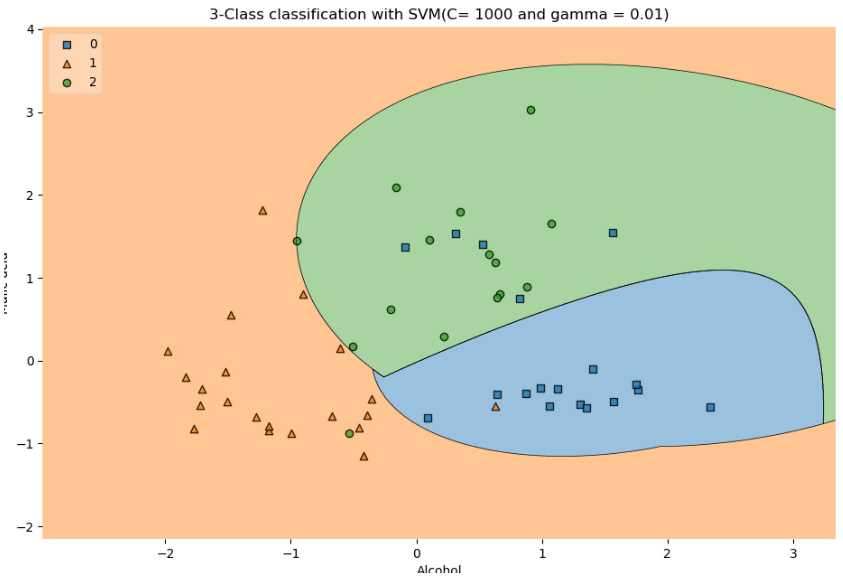


Figure 17: Best rbf SVC obtained with a grid search with a cross validation approach

---

The model scored on the test an accuracy equal to: 0.852

It can be seen that even though the model was built with a more exaustive and robust research it didn't bring the model to have an higher value of accuracy.

5 Difference between KNN and SVM

Both KNN and SVM are classification algorithms, but the idea behind them is deeply different. KNN classifies data based on the majority among the votes made by the K nearest neighbours in the training set.

To do so, it requires the definition of two important parameters:

1. K
2. Distance Measure

K will define how many neighbours will be involved in the vote, and the definition of this parameter is not simple. Its definition can be obtained based on some heuristics, but the idea is that a too high value will make the decision biased towards the most common class, on the other hand a too low value will make the model more affected by noise and outliers in the collection.
The model is very simple but the results that can be obtained are really effective, moreover since the model has not a real training phase, if new labeled data are added to the training collection, no adjustments are required to make the model able to work properly for future classifications. This means that by having more labeled data, the training phase will not be affected since there will be no time overhead.
The classification time instead is affected from the dimension of the training data and from the parameter K since more computation is required whenever a new sample needs to be classified, due to the fact that for each sample in the train set a distance measure needs to be computed from the new unlabeled sample. So having more precision costs in terms of time in the classification phase.

SVM instead is a model that classifies new data defining one or more hyperplanes that separates optimally the samples in the training set. The number of hyperplanes that will be defined, depends on the complexity of the problem, if it's a binary classification only an hyperplane will be defined, otherwise more than one.
Originally it was born to solve only linearly superable problems with hard margins finding an optimal solution, but using the kernel trick or by using the so called soft margins this limitation has been overcome.
When the soft margin are used, possibly two parameters need to be set:

1. C
2. γ

C represent the cost of misclassification, while γ represent the inverse of the radius of influence of the samples considered by the model as support vectors.
Any type of SVM needs to be trained over some labeled data (supervised-learning) and if new data are added to the train set, the training phase needs to be redone from scratch.
This is a disadvantage in the training phase with respect to the KNN model, but an advantage comes in the classification, since with this approach after the training, the prediction is faster.

6 Different pairs of attributes

It was decided to use the attributes "OD280/OD315 of diluted wines" and "Proline", to assign the class label according to the composition of these two components.
Here are reported just the final evaluations of the different models on the test set, but further informations and plots can be find in the attachment or by simply running the scripts related to the analysis with a different pair of attributes.

6.1 KNN

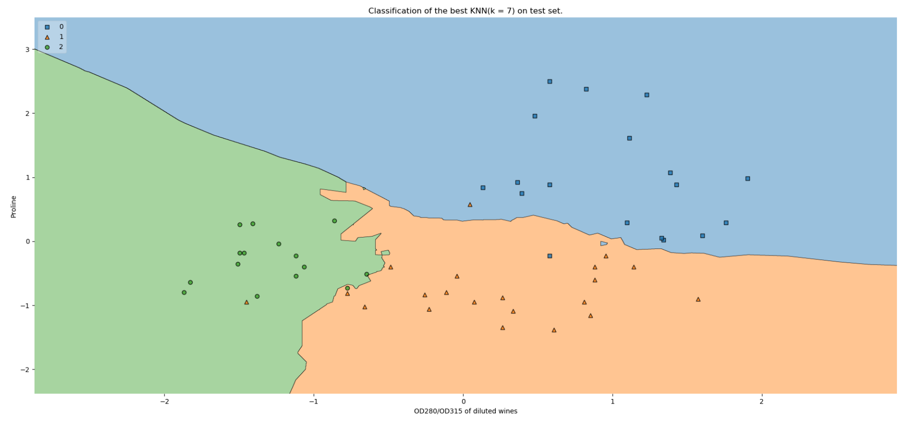


Figure 18: KNN evalaluated on the test set.

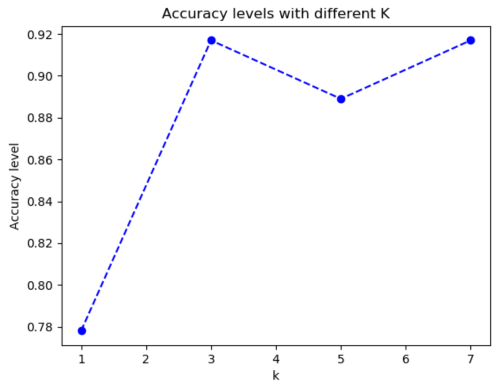


Figure 19: Accuracy plot for KNN model.

The KNN model with k = 7 has an accuracy on the test set up to: 0.926

As can be notice the decision boundaries are completly different from image 6, this can be explained by the fact that two completly different features are used to train the model. Moreover it can be seen that both the accuracy on the train and on the test sets are higher with respect to the previous case, reaching an optimal accuracy in the test equal to 92.6%.

6.2 Linear SVM

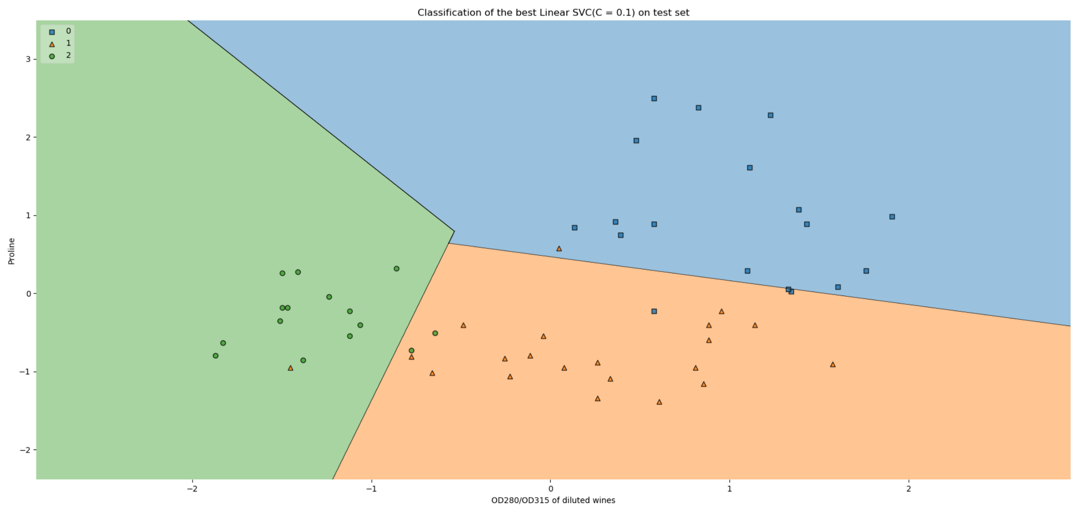


Figure 20: Linear SVC evalaluated on the test set.

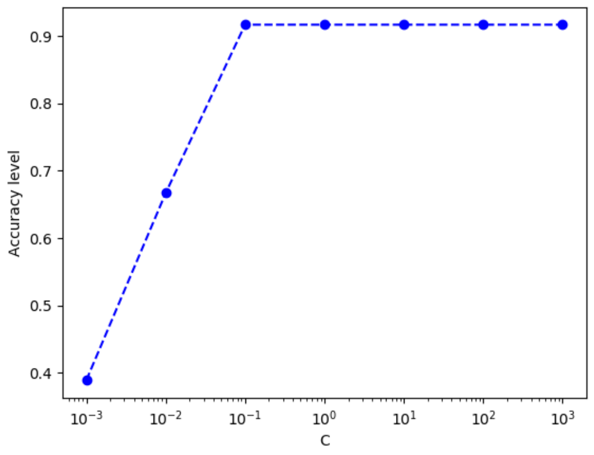


Figure 21: Accuracy plot for Linear SVC model.

The SVM built with C = 0.1 has an accuracy on the test set up to: 0.87

It can be notice that also here the decision regions are different from image 11 for the same reason as before. The model built has an higher accuracy for both the evaluations on the training and the test sets with respect to the previous analysis.
The accuracy on the training saturates to an high value equal to 91.7%, but in the test set a little bit lower accuracy was scored.

6.3 SVM with radial basis function Kernel

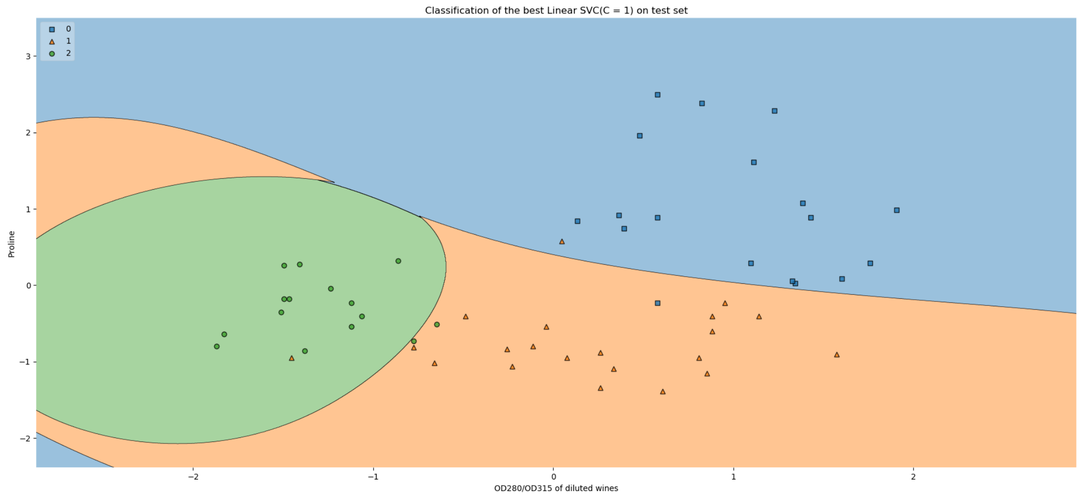


Figure 22: SVC with rbf kernel evalaluated on the test set.

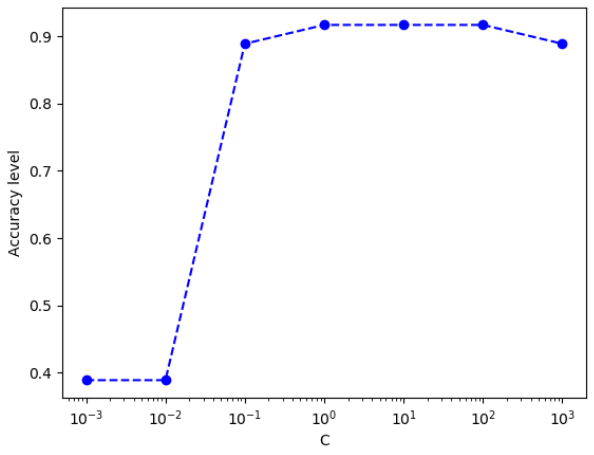


Figure 23: Accuracy plot for SVC model with rbf kernel.

The SVM built with C = 1 has an accuracy on the test set up to: 0.907

The accuracy plot obtained is very similar to the one obtained in the previous analysis (image 14), but it is relevant to see that the accuracy obtained is higher than before.
This phenomenum has happened also in the test set where an higher accuracy equal to 90.7% was registered.

6.4 Grid Search

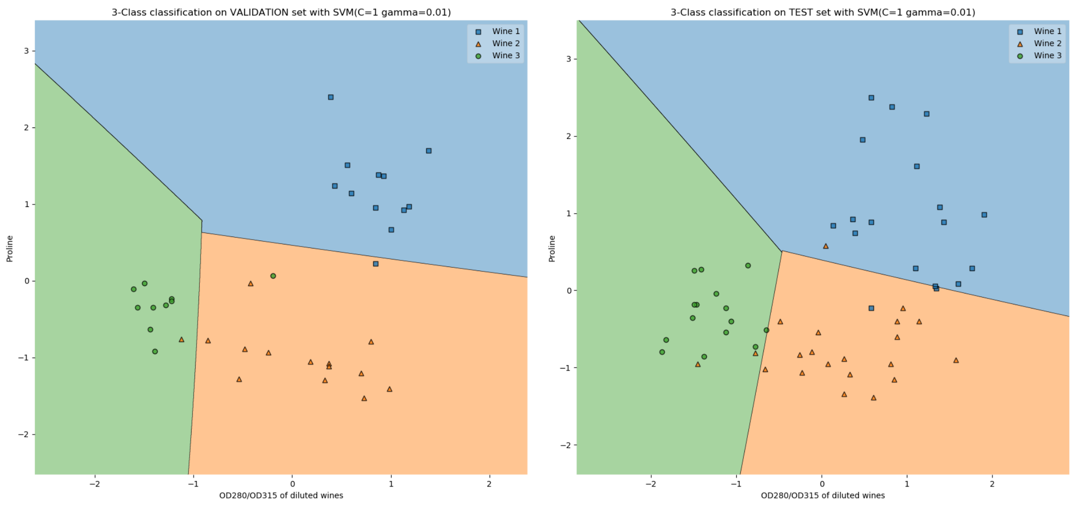


Figure 24: SVC with rbf kernel evaluated on validation and test sets.

The model was created with C = 1 and gamma = 0.01
The model scored on the validation an accuracy equal to: 0.917
The model scored on the test an accuracy equal to: 0.907

The decision boundaries from the evaluation on the validation set to the test set, have changed a bit since the model was previously fitted only on the train set, then on both the train and the validation ones. The hyperparameters used to build the model are different from the one used in the previous analysis. The boundaries, even though the kernel is rbf, seem to be described by more or less straight lines, this is a huge hint that the best kernel to be used should be the linear one. Indeed it can be also seen by plotting simply the data that the problem can be considered as linearly superable with soft margins.

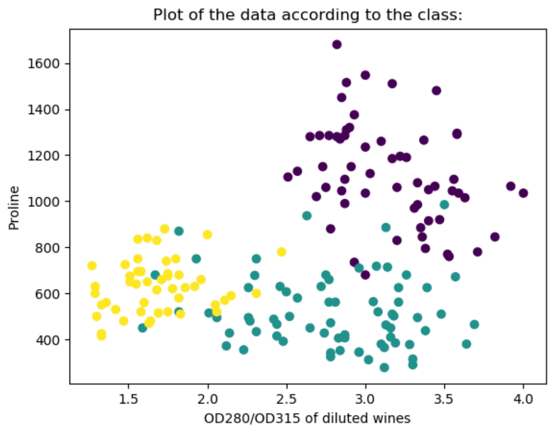


Figure 25: SVC with rbf kernel evaluated on validation and test sets.

6.5 RBF SVM with 5-Fold Cross Validation

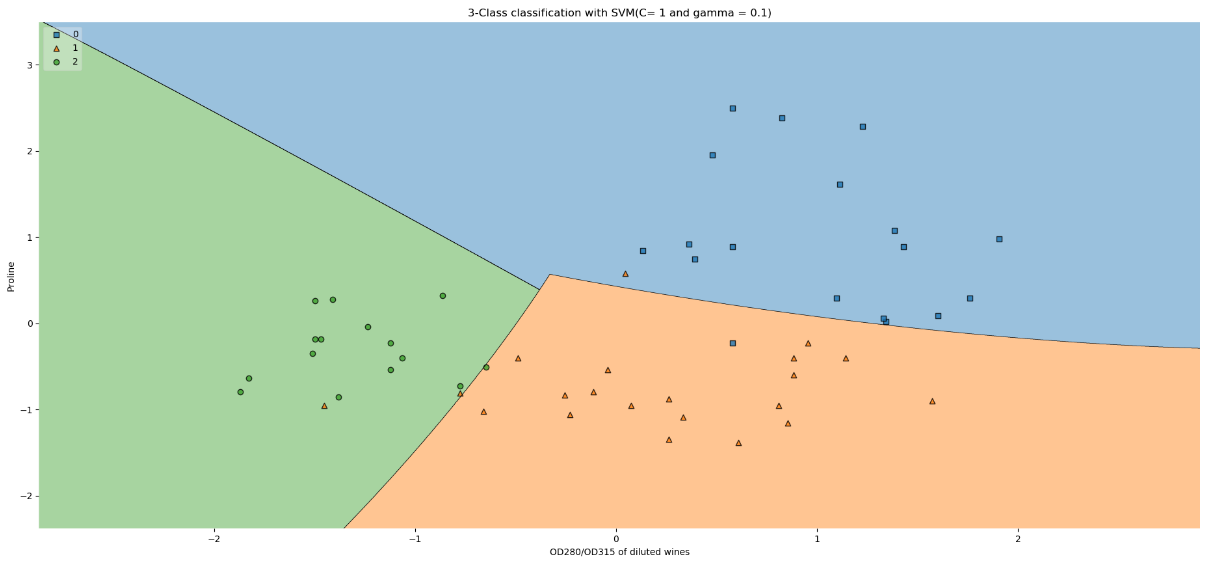


Figure 26: RBF SVM with 5-Fold Cross Validation.

The model was created with C = 1 and gamma = 0.1
The model scored on the test an accuracy equal to: 0.907

The hyperparameters used to create the model are different from the one obtained in the previous analysis as can be seen in image 17, and also here an higher accuracy value was obtained.

In conclusion, it seems that these two features are better with respect to the previous ones, since every possible model that have been trained came up with an higher accuracy.
This can be explained by the fact that the considered features better represents the variety of the data in the collection.

References

[1] Wine Dataset,
`https://archive.ics.uci.edu/ml/datasets/Wine`

[2] Support Vector Machines - Soft Margin Formulation and Kernel Trick,
`https://towardsdatascience.com/support-vector-machines-soft-margin-formulation-and-kernel-trick-4c9729dc8efe`

[3] Radial basis function kernel, `https://en.wikipedia.org/wiki/Radial_basis_function_kernel`

[4] sklearn.svm.SVC,
`https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`