ID: s267647
Name: Giorgio
Surname: Maritano

Homework 1 – Machine Learning And Artificial Intelligence

Note: Note that point 3. requires data to be splitted '*into train, validation and test sets in proportion 5:2:3*' and consequentially I assumed that all training phases in further computations are done on the train set only (that is 50% on the data). Besides point 16. says to '*Merge the training and validation split. You should now have 70% training and 30% test data*' so I assumed that only for further points the training phase will be done on the previous train+validation set.

The code with this document is commented widely to provide explanation that can be better understood if seen with the code they relate to.

K – Nearest Neighbors

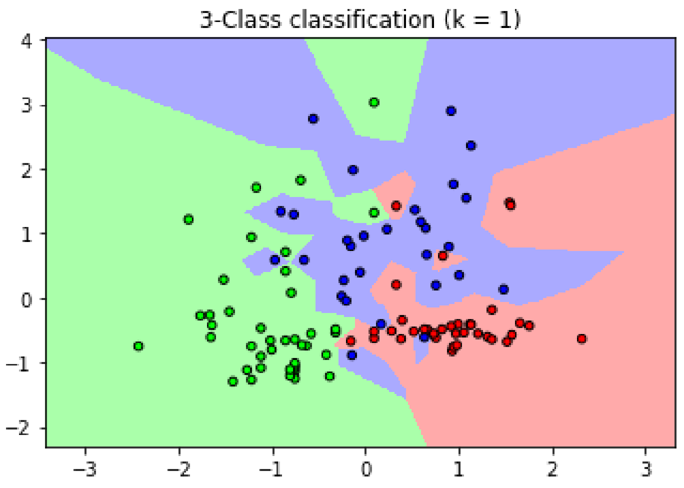
In this section was required to use the KNN algorithm to classify points imported from the wine dataset.

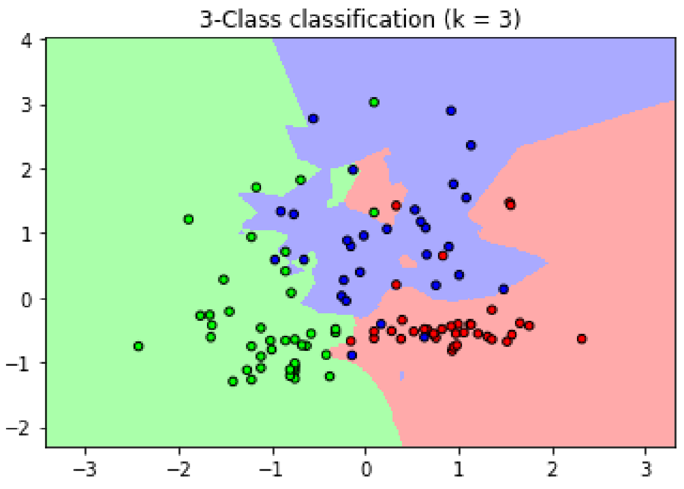
Since many Machine Learning algorithms work better with standardized data we use a Standard Scaler to preprocess data, but only once they are split into train, validation and test sets.

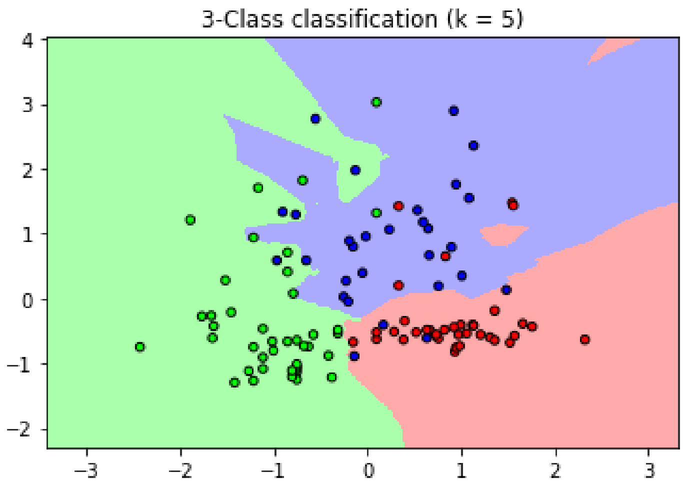
The split of which above happens randomly, and since the function *train_test_split* divides data in two parts, one of test_size% and the other into (1-test_size)%, I needed to apply it twice using the buffer ndarrays X_t and y_t (which anyway I'll be using later when I will be required to train classifier on the train+validation set).

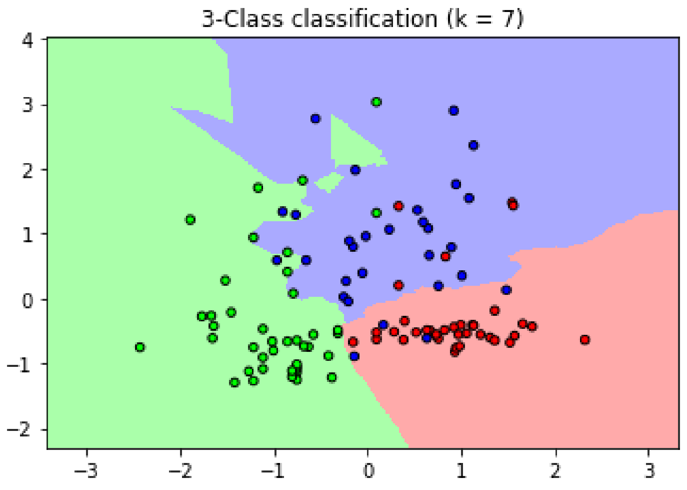
I create an array with various given values of K for the KNN algorithm and into a for loop I train a classifier and I plot data and decision boundaries for each K.

Here I will present some example after a run of the script. Note that since data are split in a pseudo-random way the output will likely change every time we run the code.


3-Class classification (k = 1)
4
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3


3-Class classification (k = 3)
4
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3

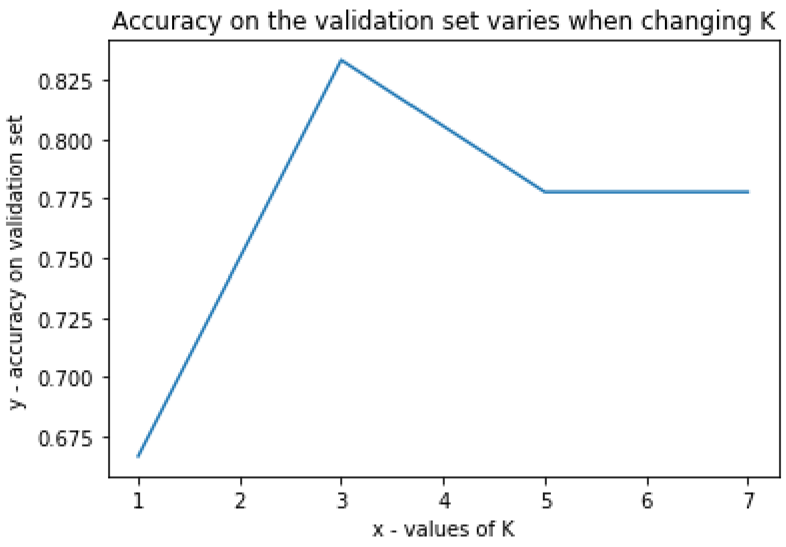

3-Class classification (k = 5)
4
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3


3-Class classification (k = 7)
4
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3

Now, after plotting the graph of data and decision boundaries I was required to show how the accuracy changes when we change the value of K.

In the KNN algorithm the value of K tends to be less accurate when closer to 1 and more accurate when increasing, but only up to a certain limit. So the best K will be likely in the middle.

Here here a graph showing the accuracy on the validation set when changing K:



As we can see the best value of K is 3, so we evaluate it of the test set. Here I will present the output I got from this run:

Use the best value of K and evaluate the model on the test set. How well does it works?

Accuracy with K = 3 evaluated on the test set: 0.741

6. How the boundaries change? Why?

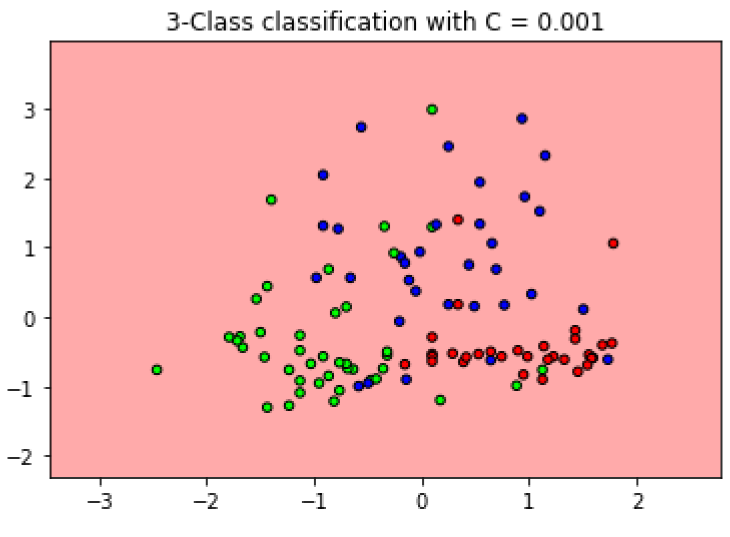
With this algorithm decision boundaries become more and more smooth and uniform and less scattered with the growing of neighbours took into considerations. In this case decision boundaries are determined by the nearest neighbours of the various classes in that region, but of course they change when you move around the instance space.

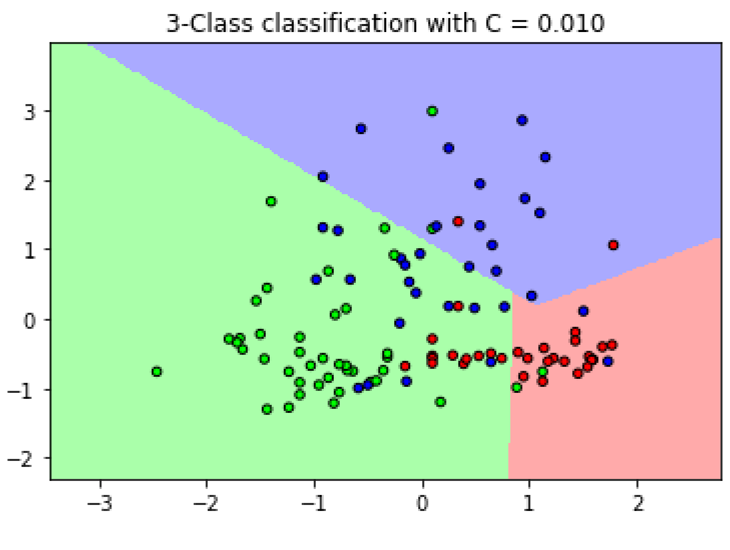
Linear SVM

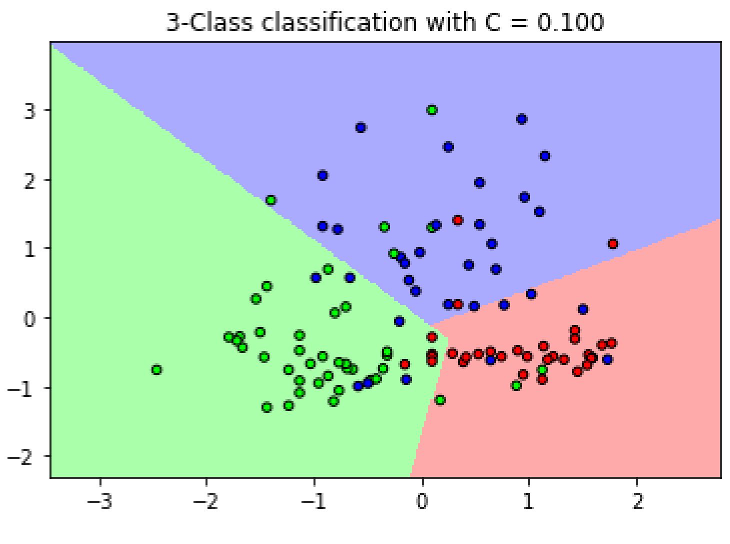
Here I was asked to do train a classifier on the same data as before, but with the difference that this time I had to use a Linear SVM.

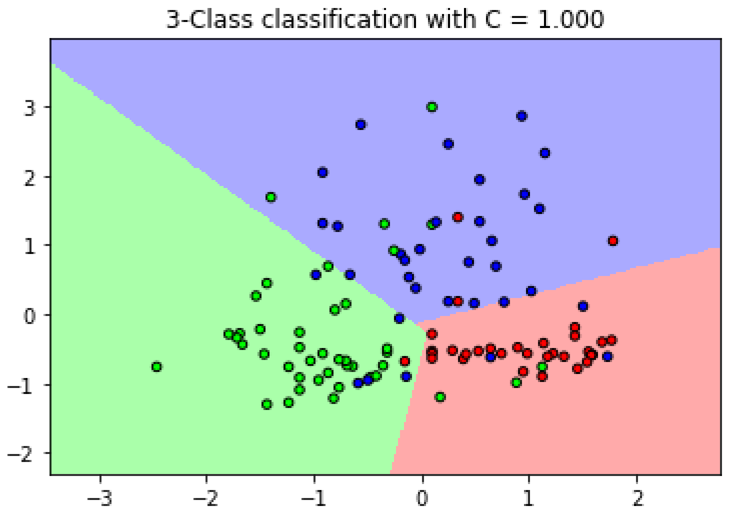
I was given various values of C, which is the cost of misclassification and thus indirectly the parameter which pilots the size of the margin: smaller values of C means that we want our classifier to keep larger margin, thus misclassification will be certain even for easily separable data.
On the other hand larger values of C means a smoother and tinier margin, which will provide better classification results but will increase a lot the cost.
My aim was in fact to find a trade off value of C which will work well on test/real value data and wouldn't have a too much high cost.

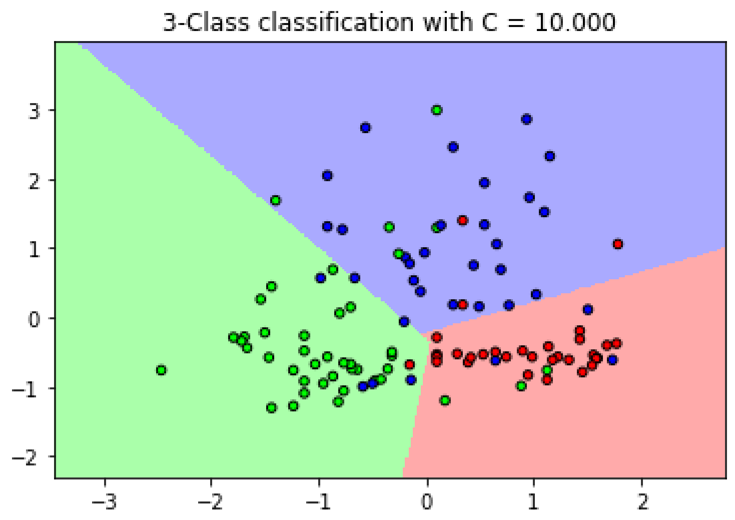
After that I had to plot the data and decision boundaries as before, and then also the accuracy on the validation set, always as a graph.

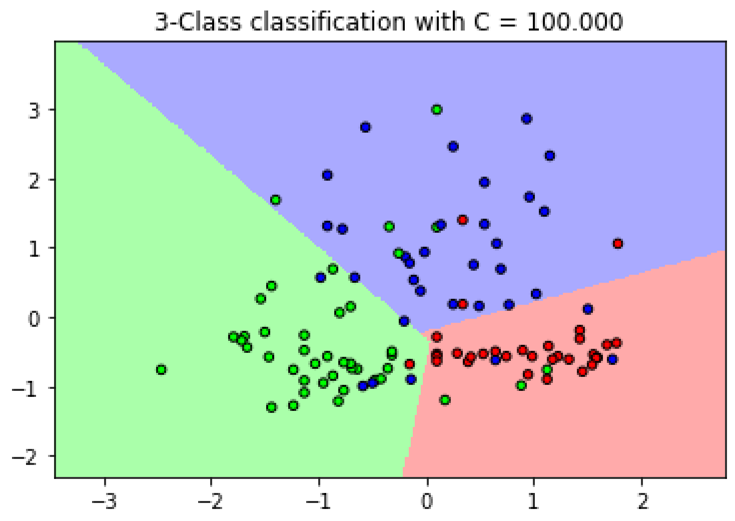


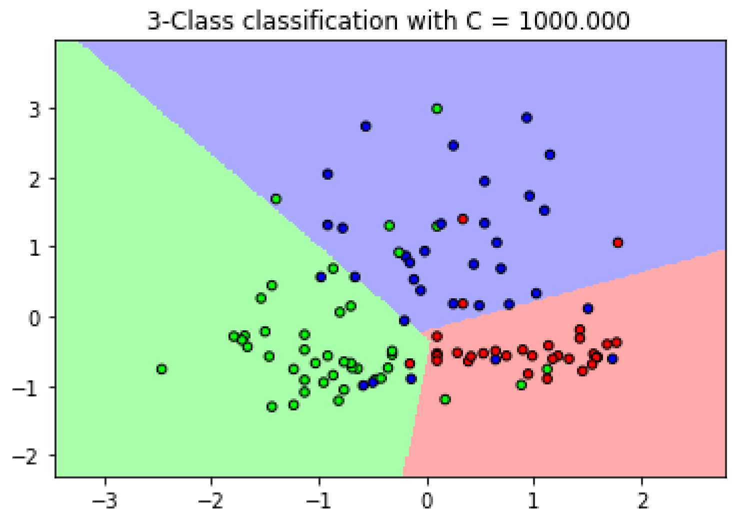




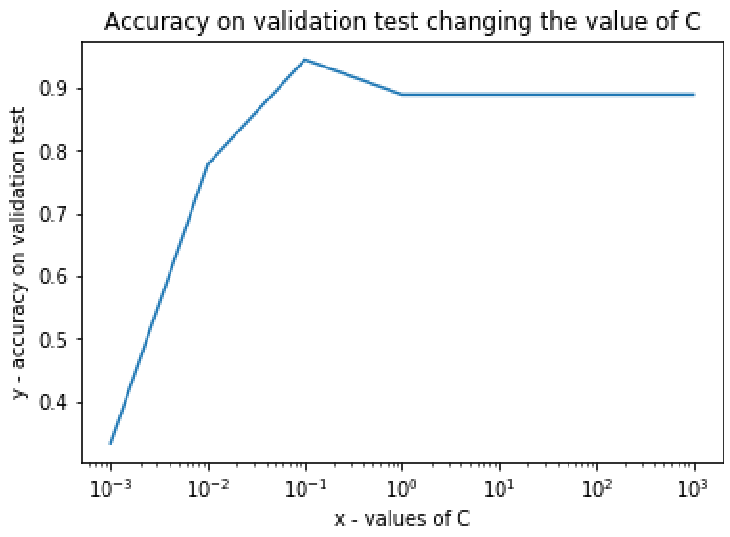



3-Class classification with C = 10.000
3
2
1
0
−1
−2
−3
−2
−1
0
1
2


3-Class classification with C = 100.000
3
2
1
0
−1
−2
−3
−2
−1
0
1
2


3-Class classification with C = 1000.000
3
2
1
0
−1
−2
−3
−2
−1
0
1
2

Now I'll show the accuracy graph, with also the score on the test set.



Use the best value of C and evaluate the model on the test set.
How well does it works?
Accuracy with C = 0.1 evaluated on the test set: 0.722222

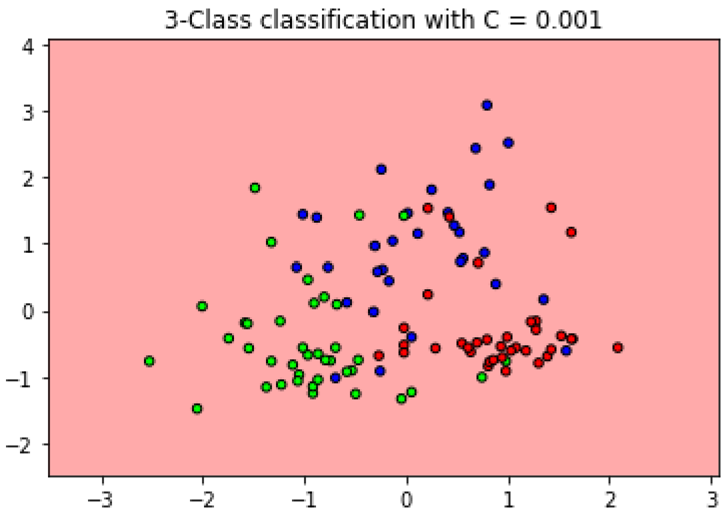
The best value of C is decided based on the accuracy score made on the validation set.

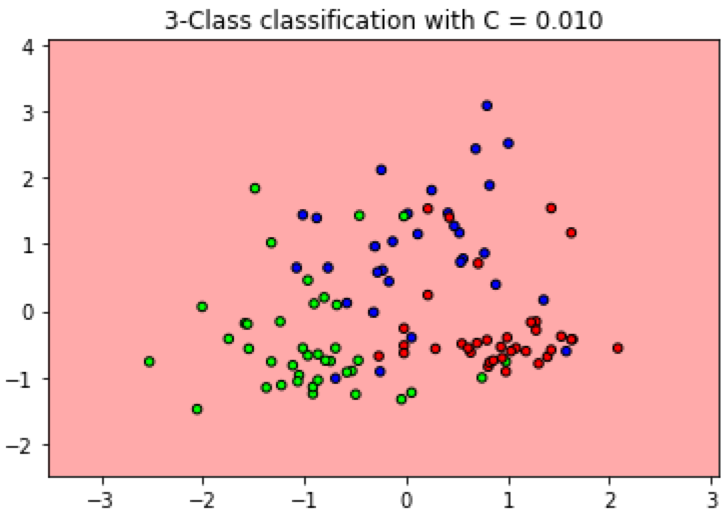
10. How the boundaries change? Why?

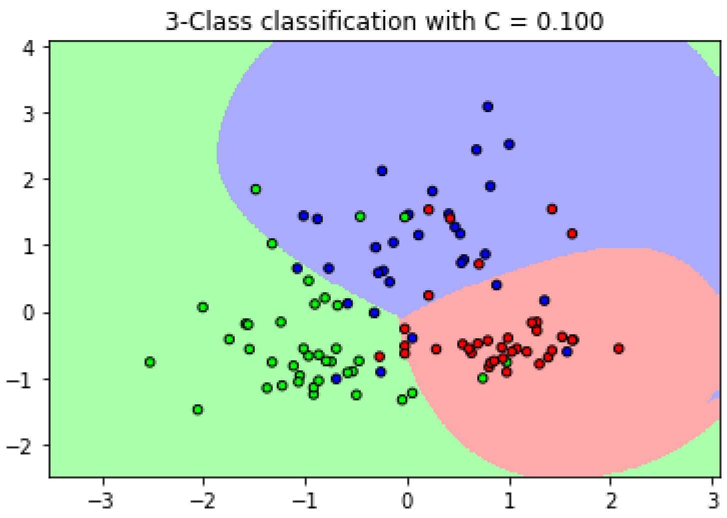
According to what I said before about the C parameter now we can observe that for the lowest C there is a complete misclassification with just one class instead of 3.
With the growing of C we can see an increase in accuracy and overall the presence of all 3 decision regions, but of course that comes with an higher computation cost.
The boundaries themselves change their orientation because of the more 'flexibility' they have.

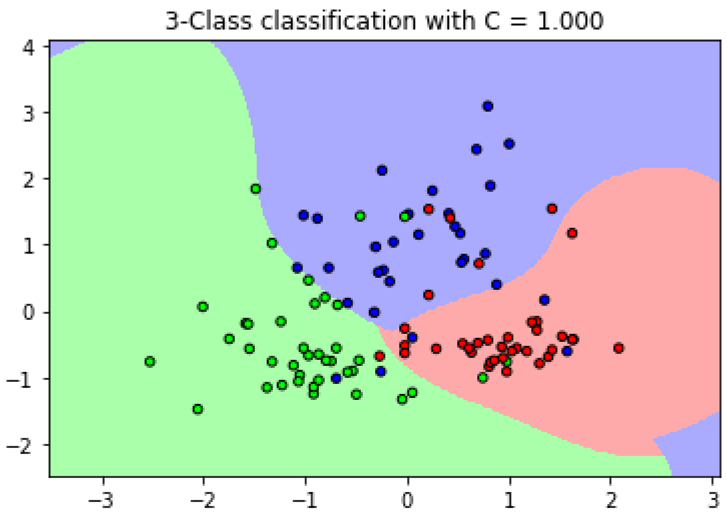
RBF Kernel

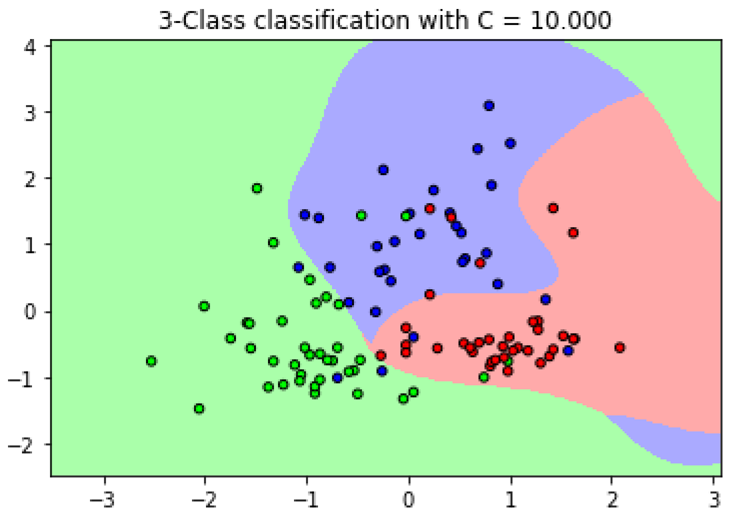
The procedure of training and plotting data and decision boundaries is the same as for the Linear SVM, but with the difference that here we specify the parameter kernel = rbf.

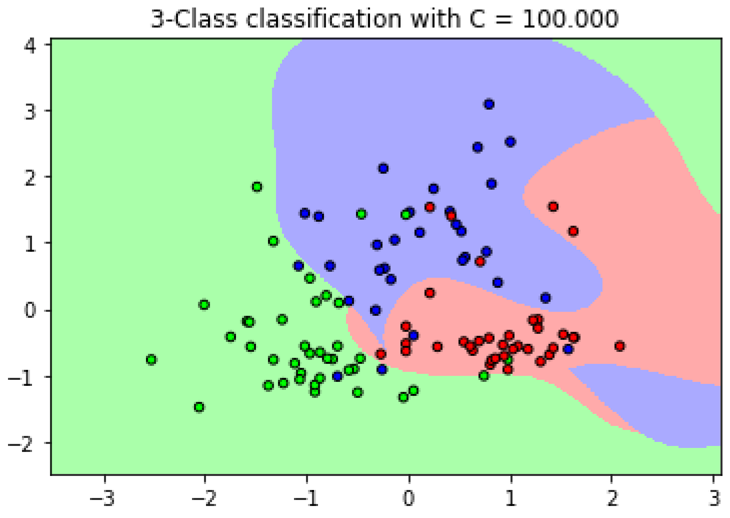


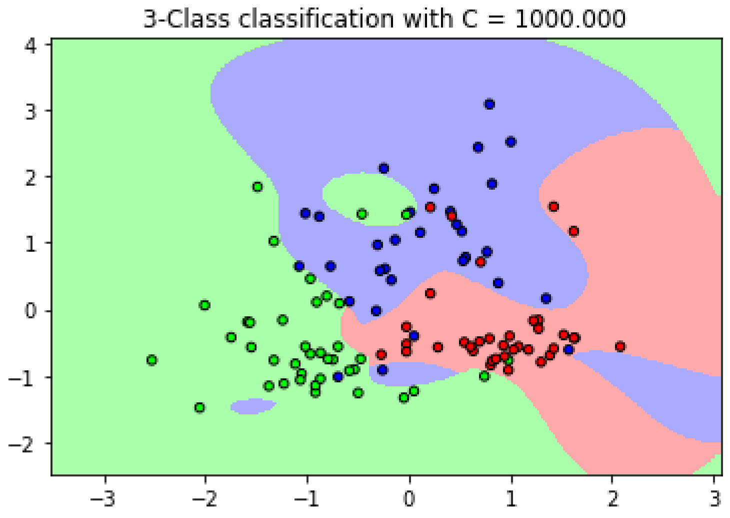







3-Class classification with C = 10.000


3-Class classification with C = 100.000


3-Class classification with C = 1000.000

Now I will report the output for the 13rd point:

We have the best score on the test set with the value of C = 10.000 and the score is: 0.852

14. Are there any differences compared to the linear kernel? How are the boundaries different?

The difference is that in the linear kernel the boundaries are straight lines, while here they are curves and regions are more scattered: indeed we can observe not only 3 regions, but there are green areas inside the blue region, for example. This is because the kernel is not linear.

To perform the grid search requested by the 15th point of the homework I used the function

clf = GridSearchCV(estimator=svm.SVC(), param_grid=parameters, n_jobs=-1)

where the parameters are:

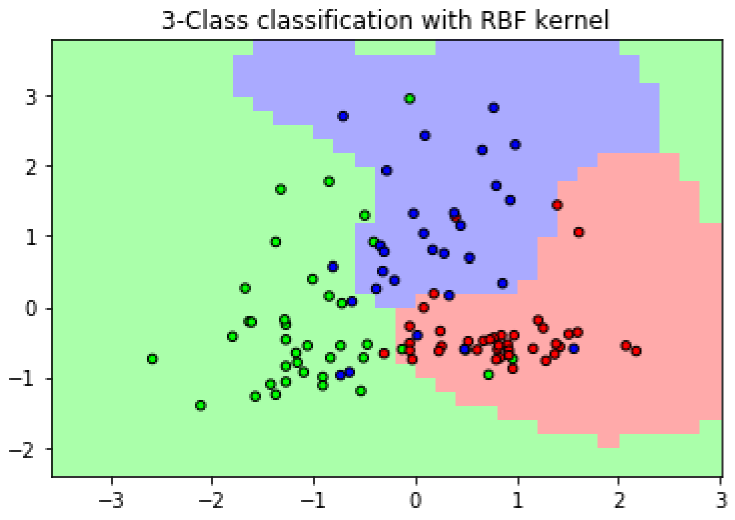
parameters = [{'C': [0.1, 1, 10, 100, 1000], 'gamma': [0.0001, 0.001, 0.01, 0.1, 1], 'kernel': ['rbf']}]

I chose values for gamma which are quite low, whereas for C value I took the previous ones given by the homework, excluding the lowest two since the accuracy was anyway too low for those values.

Now I will report the output of this classifier on the validation set, plus the accuracy of the classifier with the best C and gamma parameters but this time **on the test set**. In the end I will show the data and decision boundaries of the classifier with the best parameters.

Score on validation set: 0.9444444444444444

Best score of the RBF kernel with the best parameters: 0.8148148148148148



<u>K-Fold</u>

I now had to merge the validation and training data into one set only. To do so I simply kept the set resulting from the first split and made the Grid Search for gamma and C as before for the RBF kernel, but this time with also the cv=5 option to perform a 5-fold cross validation.

Eventually I evaluated the parameters on the test set and got this result:

Score with 5-fold cross validation on the test set: 0.7777777777777778

18. Is the final score different? Why?

Since data are pseudo-randomly split every time we run the script data can change every time and so accuracy.
I observed that usually accuracy for the RBF kernel with and without the cross validation is different most of the times and there isn’t a classifier which works always better respect to the other.
The score is different because of how it works the K-fold cross validation: since with it the classifier takes out 1/K of the training data and trains on the remaining (1-K)/K of the data for K times, at the end it averages all the accuracies and because of this the final accuracy should be more precise and gives us a more realistic evaluation of the model.