Introduction

In this homework we are trying to classify the data in 3 different classes, using 2 out of 13 features in order to visualize it in a 2D plot.

We start by randomly splitting the data in 3 sets: Train, Validation and Test, in proportion 5:2:3.

We can then plot the data in order to try to get a glimpse of how it is organized.

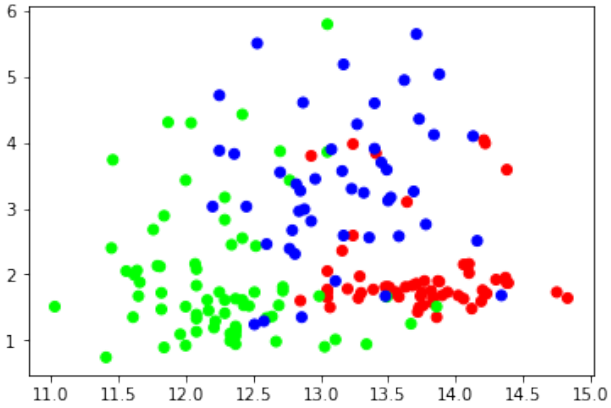


Figure 1: 3 classes plotted using first two features.

K-Nearest Neighbors

kNN has an hyper-parameter k, which is a parameter that has to be set by the programmer. k represents the number of closest samples to check in order to classify a sample; we're going to evaluate different values (1, 3, 5, 7) on the validation set.

In Figure 3 one can see the boundaries and the accuracy on the validation set for each value of k.
Accuracy results are shown in Figure 2, for the given validation set k=3 seems to be the best choice.

With smaller k we can see that sometimes data could be classified incorrectly due to outliers, but, while this problem tends to go away with a larger k, if we increase its value more and more we reach a point in which all samples are classified as the most common class.

If we evaluate the test set with the best value for k we obtain a success percentage of 76%,

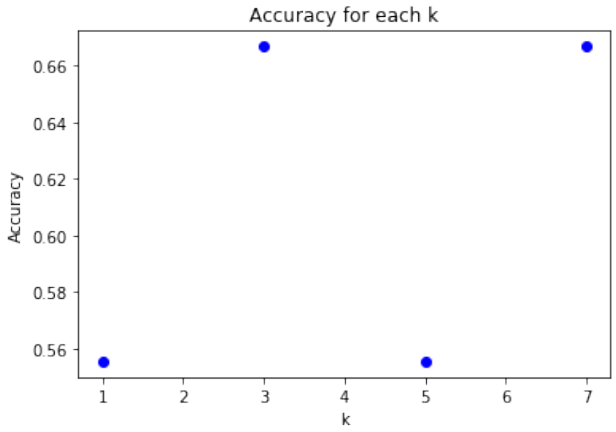
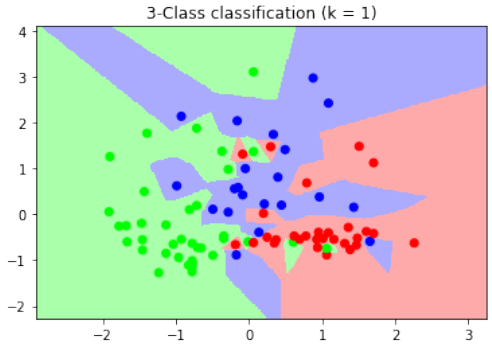


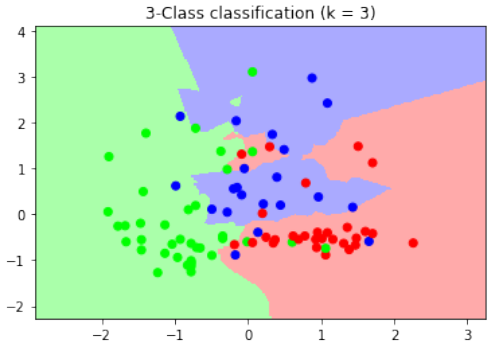
Figure 2: Accuracy for each value of k

we could try to increase this value by using more features, choosing the two features that give us better results.

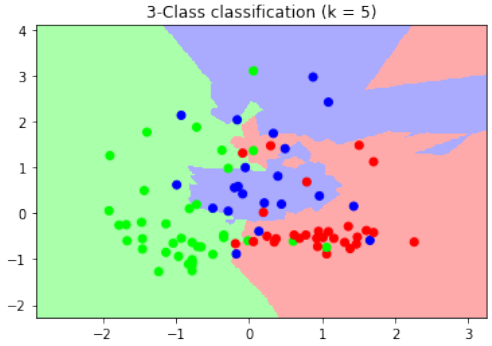
However, one can argue that with k equal to the number of classes sooner or later there will be ties.



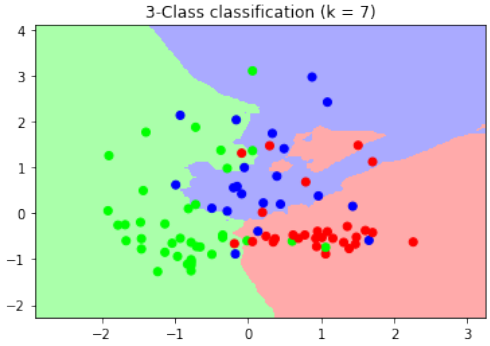
(a) k=1, accuracy=55.6%



(b) k=3, accuracy=66.7%



(c) k=5, accuracy=55.6%



(d) k=7, accuracy=66.7%

Figure 3: Boundaries and train data for different values of k; accuracy on the validation set

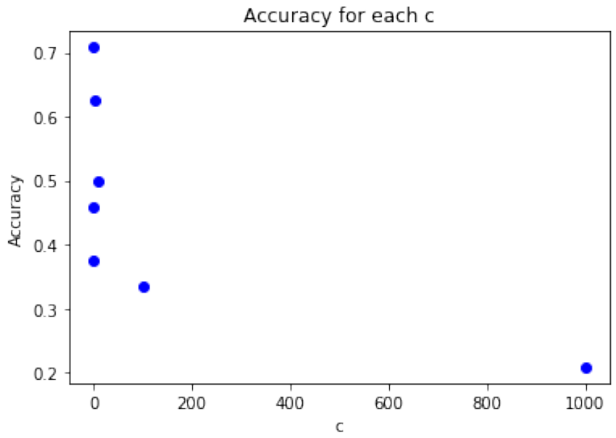


Figure 4: Accuracy for each value of c

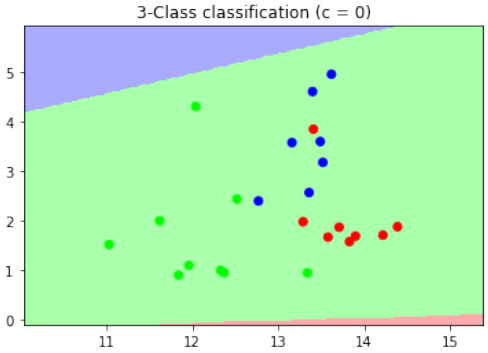
Linear SVM

In this section we are classifying the data with a linear support vector machine.

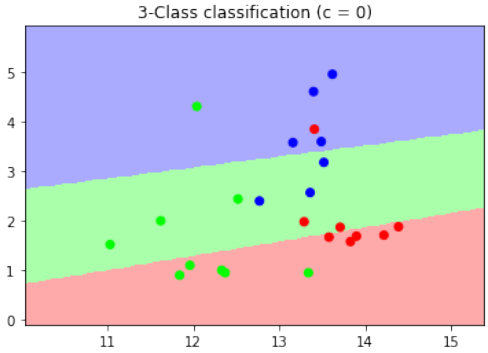
In Figure 5 one can observe the decision boundaries for each value of $C \in [0.001, 0.01, 0.1, 1, 10, 100, 1000]$; decision boundary are linear and differ from one another only in angles between them or in the bias.
This can be explained by the fact that a linear SVM makes the assumption that the data is linearly separable and thus cannot catch non-linearity.
Due to this characteristic, and to the fact that we are taking into consideration just 2 out of 13 variables, the classifier has a lots of variance for the accuracy on the test test, with percentages going from 49% to 71% on the same set.

In figure 4 are shown the results of the evaluations for all of the possible value of C, from there we can take the best value for c, which is 0.1.

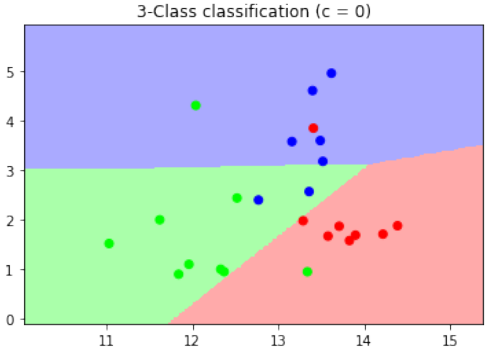
Using C=0.1 we can then evaluate on the test set, in order to know the performance of our classifier on unknown data.
The accuracy classified successfully 71% of the samples.



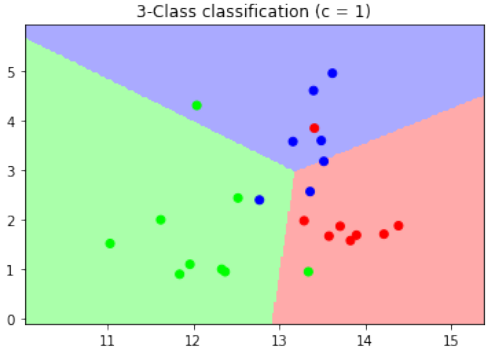
(a) c=0.001, accuracy=38%



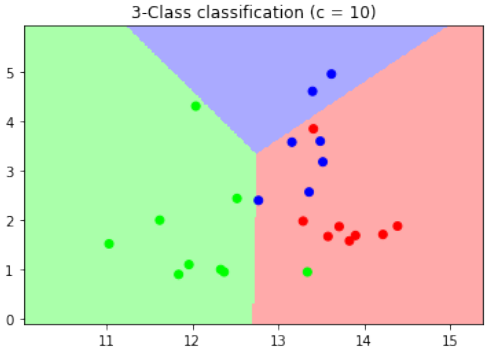
(b) c=0.01, accuracy=46%



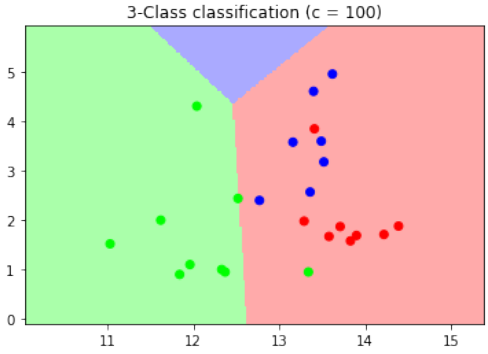
(c) c=0.1, accuracy=71%



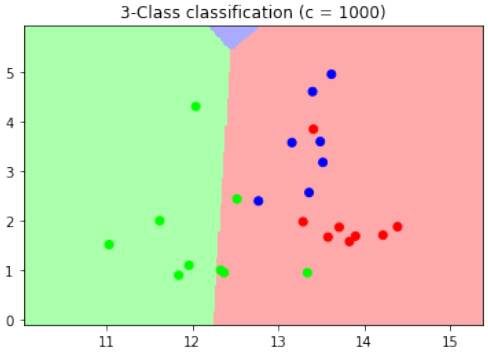
(d) c=1, accuracy=63%



(e) c=10, accuracy=50%



(f) c=100, accuracy=33%



(g) c=1000, accuracy=21%

Figure 5: Boundaries and train data for different values of k; accuracy on the validation set

RBF Kernel

RBF Kernel can be use to better classify non-linear data, in this section we keep changing only the hyper-parameter C and evaluate the performance on the validation set for the same values used above.
Decision boundaries can be seen in Figure 7.

It can be noticed that they are not linear and that they should perform better than those of the linear SVM with this data, however it can also be seen that for large C this classifier tend to overfit and to give more and more importance to outliers.

In fact C represents how much we want to avoid misclassifying samples, so for large C the hyperplanes will have smaller margins, if these will lead to better performances on the train set.

The best value found for C is 1; using this value for C the classifier classifies correctly 81.5% of the samples in the test set.

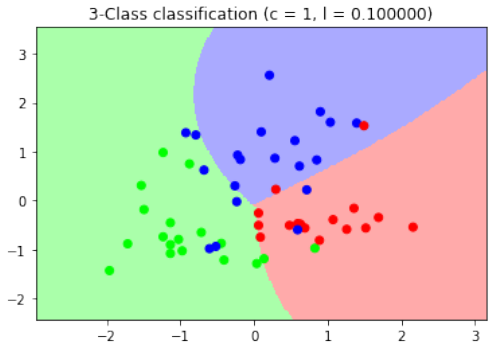
But for this type of classifier one can set another hyper-parameter γ, which is correlated to the tradeoff between error due to the variance and the error due to the bias in the model: $k(x_i, x_j) = \exp -\gamma(x_i - x_j)^2$.
In particular, with large γ the SVM is able to better capture the complexity of the data, but if it become too large there's the risk of overfitting the data.

I set, as possible values for γ, [0.7, 0.4, 0.1, 0.07, 0.04, 0.03, 0.01, 0.001, 0.0001] and built a grid with accuracy values for each pair of C and γ.
The grid is shown in Table 1.

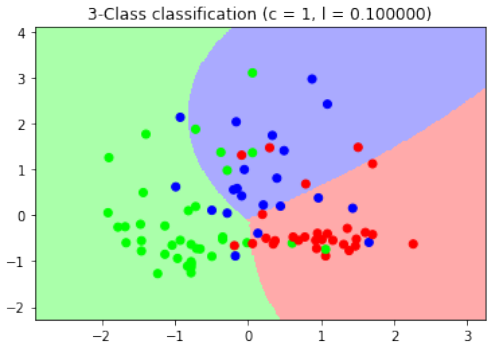
From the grid we can read the best values for C and γ, between those considered, and use them to evaluate the SVM on the test set.
The decision boundaries for this value can be seen in Figure 6.
With C = 1 and $\gamma = 0.7$ 74.0% of the test samples are classified correctly.

C/γ	0.7	0.4	0.1	0.07	0.04	0.03	0.01	0.001	0.0001
0.001	41.7%	41.7%	41.7%	41.7%	41.7%	41.7%	41.7%	41.7%	41.7%
0.01	41.7%	41.7%	41.7%	41.7%	41.7%	41.7%	41.7%	41.7%	41.7%
0.1	38.9%	38.9%	41.7%	41.7%	41.7%	41.7%	41.7%	41.7%	41.7%
1	72.2%	72.2%	77.8%	77.8%	77.8%	66.7%	44.4%	41.7%	41.7%
10	72.2%	66.7%	72.2%	72.2%	72.2%	72.2%	77.8%	44.4%	41.7%
100	63.9%	72.2%	72.2%	72.2%	72.2%	72.2%	72.2%	77.8%	44.4%
1000	52.8%	58.3%	72.2%	72.2%	72.2%	72.2%	72.2%	77.8%	77.8%

Table 1: Accuracy for each pair of C and γ

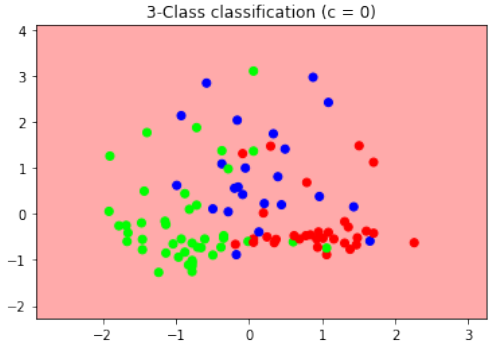


(a) Decision boundaries and test set.

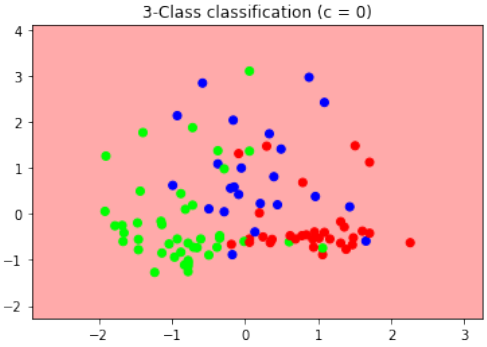


(b) Decision boundaries, train and validation set.

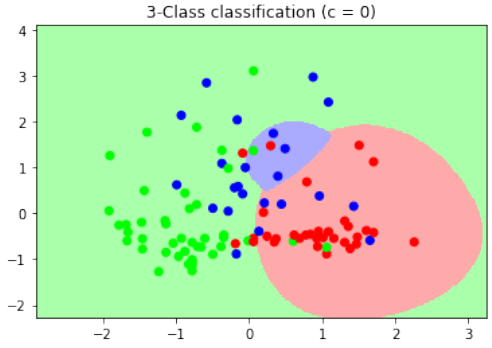
Figure 6: Boundaries, train and test set for best values of C and γ found



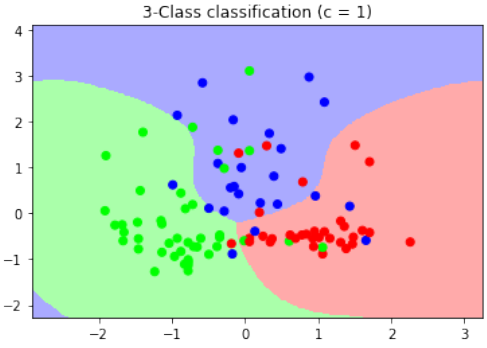
(a) c=0.001, accuracy=36%



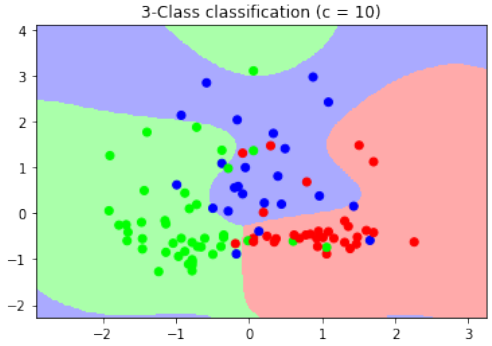
(b) c=0.01, accuracy=36%



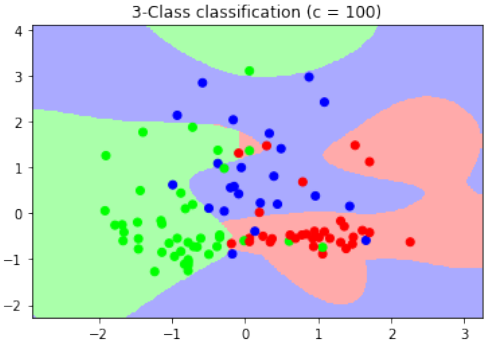
(c) c=0.1, accuracy=48%



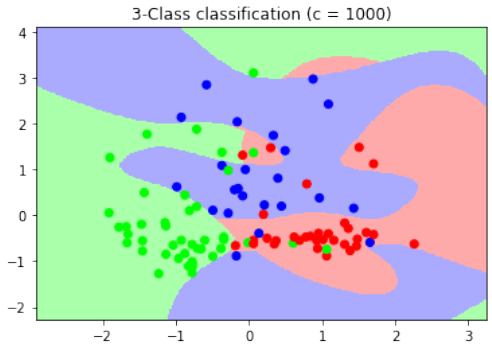
(d) c=1, accuracy=72%



(e) c=10, accuracy=72%



(f) c=100, accuracy=72%



(g) c=1000, accuracy=64%

Figure 7: Boundaries and train data for different values of k; accuracy on the validation set

C/γ	0.7	0.4	0.1	0.07	0.04	0.03	0.01	0.001	0.0001
0.001	41.7%	41.7%	41.7%	41.7%	41.7%	41.7%	41.7%	41.7%	41.7%
0.01	41.7%	41.7%	41.7%	41.7%	41.7%	41.7%	41.7%	41.7%	41.7%
0.1	51.7%	51.7%	39.2%	38.3%	40.9%	41.7%	41.7%	41.7%	41.7%
1	64.2%	64.2%	60.9%	65.0%	60.0%	60.0%	47.5%	41.7%	41.7%
10	65.9%	62.5%	62.5%	62.5%	62.5%	60.9%	60.0%	47.5%	41.7%
100	61.7%	64.2%	64.2%	64.2%	64.2%	62.5%	62.5%	62.5%	47.5%
1000	55.0%	56.7%	65.8%	62.5%	64.2%	64.2%	64.2%	60.9%	62.5%

Table 2: Mean accuracy for each pair of C and γ

k-Fold Cross Validation

In order to do a 5-fold cross validation we need to merge our train and validation sets and then split them again in two for each internal iteration (20% validation set and 80% train set).

In fact we will iterate through all the possible values for C and for γ and for each pair perform the cross validation; part of the code is written below, in order to give a better explanation.

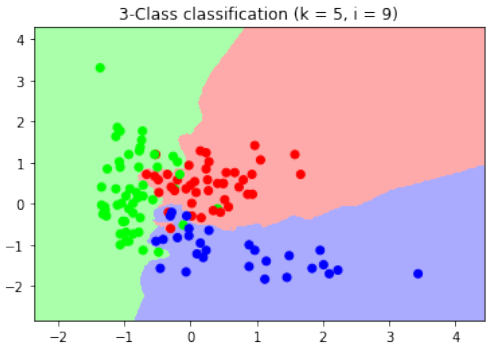
Algorithm 1: k-Fold Cross Validation iterations

```
for c in [0.001, 0.01, 0.1, 1, 10, 100, 1000]:
    for l in [0.1, 0.03, 0.01, 0.001, 0.0001]:
        for i in range(5):
            #1)split validation and train set (proportion 2:8),
            #different samples at each iteration
            #2)fit end evaluation
        #3)mean of all evaluations for current pair of c and l
```

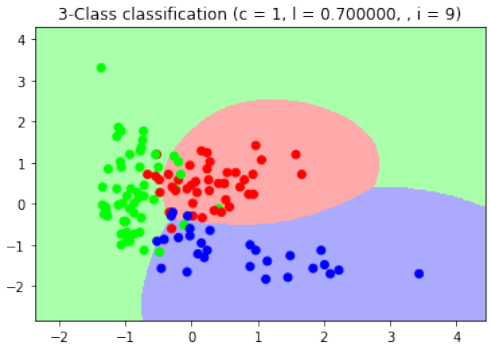
If we perform again a search for the best values of c and γ through a grid search, we find that with c=10 and γ=0.7 the SVM classifier has better performances, as shown in Table 2.

Consequently we use these values to fit a model and evaluate it on the test set, achieving correct classification for about 79.6% of the samples.

The score obtained with cross validation is slightly better than the one reached before, this can be explained by the fact that by applying cross validation we are reducing the risk of overfitting the data.



(a) Decision boundaries and train set for kNN, features 9 and 10.



(b) Decision boundaries and train set for SVM, features 9 and 10.

Figure 8: Boundaries and train set for best features and best hyper-parameters found.

kNN vs SVM

Both kNN and SVM reach percentages of success, on the test set, near 80%; but, while the former doesn't need a training phase and for each sample of the test set search the k nearest training samples, the latter uses the train set to build the decision boundaries.

This, in the case of kNN, leads to an high computational cost in the test phase and to an high memory usage, because it needs to store all of the train set.

However kNN makes no assumption on the data, while SVM was born as a linear classifier and can be use, with higher percentages of success, in case of non-linearity only if associated with some appropriate kernel (RBF in this homework).

Search for best pair of features

We can now search for the best pair of features (search limited to consecutive features, in order to reduce the computational cost).

In the case of kNN and SVM with RBF kernel the best couple of features is 9-10, which for kNN with k=5 results in 96.3% of correctly classified samples of the test set and for SVM with c=1 and l=0.7 results in 94.4%. (Figure 8)