Exercise 4: Kernels, decision trees, model selection, and statistical validation

Introduction to Machine Learning

University of Barcelona

December 20, 2015

Authors:
Camps, Julià
Serra, Xavier

Contents

1 Question Block 1 **3**

2 Question Block 2 **8**

3 Question Block 3 **10**

4 Question Block 4 **12**

5 Question Block 5 **17**

6 Question Block 6 **18**

7 Question Block 7 **19**

8 Question Block 8 **20**

1 Question Block 1

1 *Load the dataset 'example_dataset_1.mat'.*

We load the required dataset into Matlab. This dataset is non-linearly separable. Its elements are described by 2 attributes and a class label, which can be $+1$ or -1.

2 *Create the Gram matrix for $\sigma = 1$ and plot it (you may use imagesc to display the matrix and L2_distance for computing distances between points).*

We use the gram matrix formula in order to describe it:

$$\text{gram matrix} = \exp\left(\frac{-\|x_i - x_j\|^2}{\sigma^2}\right)$$

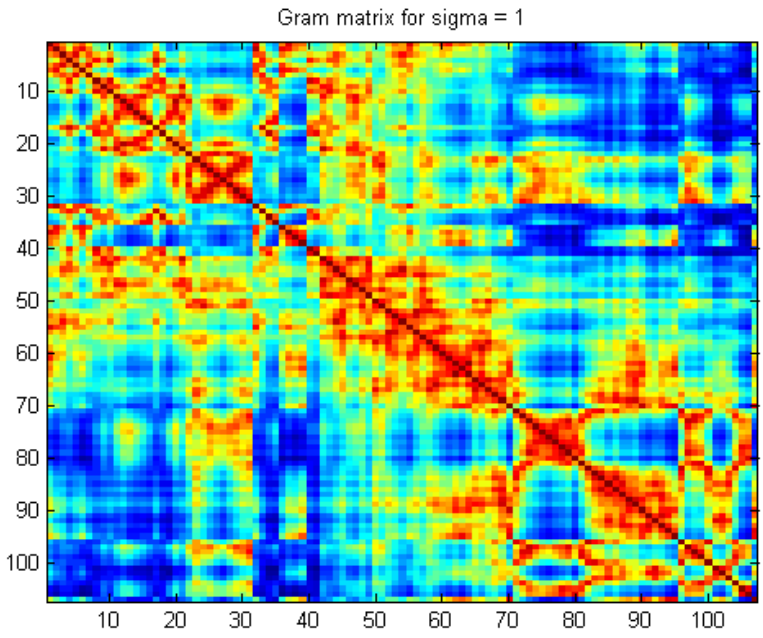


Figure 1: Gram matrix for rbf kernel with $\sigma = 10$

In figure 1 can observe than this is a nice gram matrix for applying rbf, due that the higher values are concentrating on its diagonal, and we have the lowest values concentrating on the antidiagonal.

3 *Describe the Gram matrix displayed. Which are the maximum values of the matrix? And the minimum? Is it positive definite (check if all the eigenvalues are positive)?*

As we commented in figure 1, we have obtained a nice gram matrix for $\sigma = 1$ with 'example_dataset_1.mat' dataset.

Its maximum values are placed on the main diagonal and the minimum are on antidiagonal, but not on the common part between both. As we can see on figure 2, which is showing where are some of the absolute maximum and minimum values placed on the covered space of the gram matrix, but without the gram matrix values themselves.

We can affirm that it is positive definite. From observing the eigenvalues of a matrix, we can affirm that it is positive definite if, and only if, all eigenvalues are positive and greater than zero.

Let K be our gram matrix, eig a function than gives all eigenvalues of a matrix and PD all positive definite matrix set, so we can express it as

$$K \in PD \iff \forall eig_i \in eig(K), \quad eig_i > 0$$

As our gram matrix, K, fulfils this property

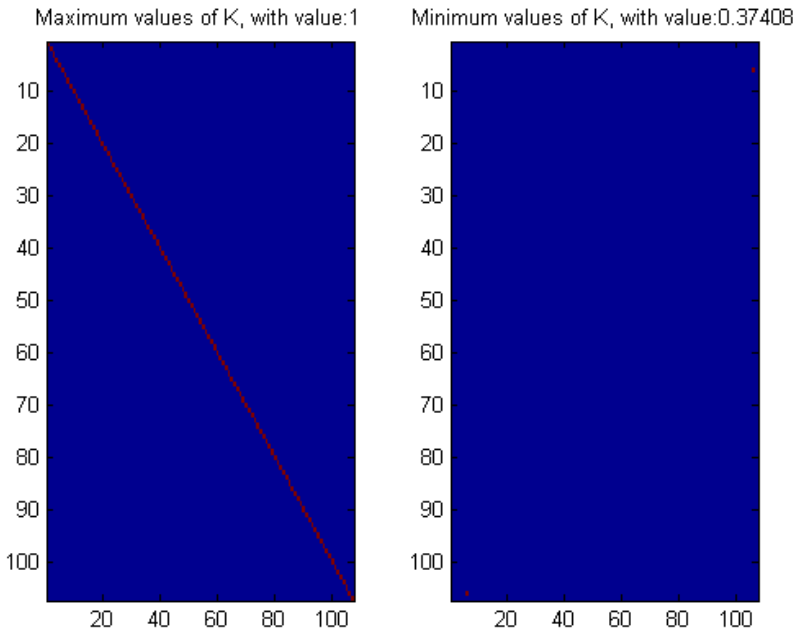


Figure 2: Gram matrix for rbf kernel with $\sigma = 10$

4 *Modify the learning code of the dual SVM to handle the kernel. Do not use the offset b.*

For changing this part, first we implemented the dual SVM function, which retrieves a solution as a function of the SVs in the SVM. Then we switched the kernel part of the function.

From the "Representer's theorem" we know that if we represent our any non-linearly separable on a very high dimensional space, this will become linearly separable by an hyperplane in that space, and than we can re-project back the hyperplane solution to the original space.

However, computation in this feature spaces can be very costly due their high dimensionality, which are, typically, inifinite-dimensional feature spaces.

Then we know than in SVM dual representation the dimensionality is only relying on the inner product of the dataset data. So this allows to introduce the concept of "The Kernel Trick", which by simply switching an inner product of the data to a kernel than retrieves a PDS[1] matrix, because a PDS matrix can be represented as the inner product of a matrix in "some" (unknown) dimensional space, allows us to separate our data on a inifinite-dimensional feature space.

We first implement the Dual SVM:

$$W(\nu) = \sum_{i=1}^{n} \nu_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \nu_i \nu_j y_i y_j x_i^T x_j$$

, then we use "The Kernel Trick", so we switch the inner product of our data, $(x_i^T x_j)$ to a kernel K, defined as $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$. In this case we use the rbf kernel $K = \exp\left(\frac{-\|x_i - x_j\|^2}{\sigma^2}\right)$.

So finally we change our dual implementation to use a the rbf kernel

$$W(\nu) = \sum_{i=1}^{n} \nu_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \nu_i \nu_j y_i y_j K(x_i, x_j), \quad K = \exp\left(\frac{-\|x_i - x_j\|^2}{\sigma^2}\right)$$

5 *Run your training algorithm on that dataset with $\lambda = 1$ and $\sigma = 1$.*

In Dual SVM, the representation of the SVM is expressed as a function of the SVs, rather than as a function of the attributes of the data.

Models obtained from the dual have as many weights as SVs we have in our SVM. So the number of parameters in the dual is not bounded to the dimensionality of the data, this clarifies the model in high dimensional spaces.

Now, as we are using a non-linear kernel, we can not recover the primal linear representation as we did, when playing with linear SVM models, on the third practicum.

We do not consider relevant to show explicitly the values of ν neither the SV values for this exercise, and we do not have the b value, because we

[1] PDS: Positive Semi-Definite.

have taken it apart from the formulation of the SVM as a requirement of the previous question. Instead, we will explain explain some other relevant information of the obtained dual SVM model, such as the total number of SVs N_{SVs}, the number of SVs that are laying on the margin N_{SVm}, the margin distance to the hyperplane m_{dist} and the training error $trainErr$.

After training with $\lambda = 1$ and $\sigma = 1$ we obtained this model:

- N_{SVs} : 36
- N_{SVm} : 36
- m_{dist} : 0.0095
- $trainErr$: 0

As we can observe from the information reported above:

- All SVs in our SVM are laying on the margin, so our margin will wiggle with our data and try to adapt to it. The number of SVs on the margin is an indicator to take into consideration when trying to detect complexity overfitting on SVMs, which is indicated by the σ value in the formulation, but as we can see in figure 3 this model is not still overfitted from a complexity point of view.

  It also indicates than the λ value is high enough to, make it worth to, classify all values out of the margin.
- The train error is zero, this means than the model complexity specified by σ, is high enough to correctly classify all the training instances. And than the λ value is high enogh to make it worth to classify them.

6 *Plot the dataset and the separating hyperplane. What do you observe?*

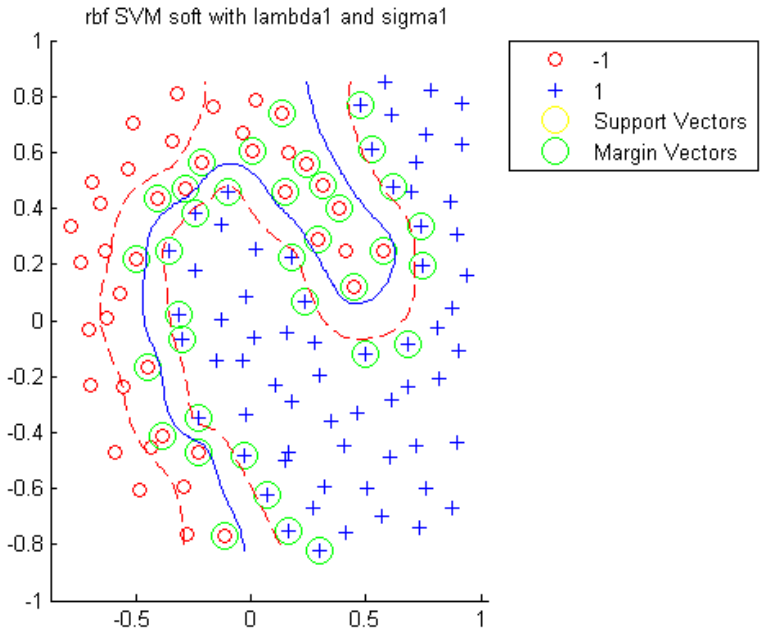


Figure 3: Plot of the data and its separating hyperplane with $\lambda = 1$ and $\sigma = 1$

In figure 3 we observe that as we mentioned when describing the obtained model, all instances are correctly classified, and that we only have SVs on the margin, what already knew from looking at ν, and seeing than we had no ν values equal to λ.

2 Question Block 2

1 *Consider that you train a full decision tree. What is the expected training error to obtain? Why?*

The expected training error to obtain is zero. We know than a tree will perform binary decisions on each node, in order to split the instances represented by it, into nodes only represented by one single class elements.

If we apply this process to a deterministic dataset[2] we will always find a representation of this type than separates all our data. Even if we have to go the extream and have one particular case for each instance on a dataset that determines its class.

So if we build a full decision tree, we are just saying than we will continue splitting the nodes until all final nodes represent one single class elements. Moreover, the training error on this model will be always equal to zero, because we have fitted it completely to the training dataset.

2 *Load the dataset 'example_dataset_1.mat'.*

We load that same dataset than in the previous block, **Question block 1**. Which as we said, is a non-linearly separable dataset.

3 *Train a tree using the default parameters and plot the result on the training set. Compute and report the training error? Is the result what you expected?*

For training the tree we used the Matlab function *tree=classregtree(X, Y)*.

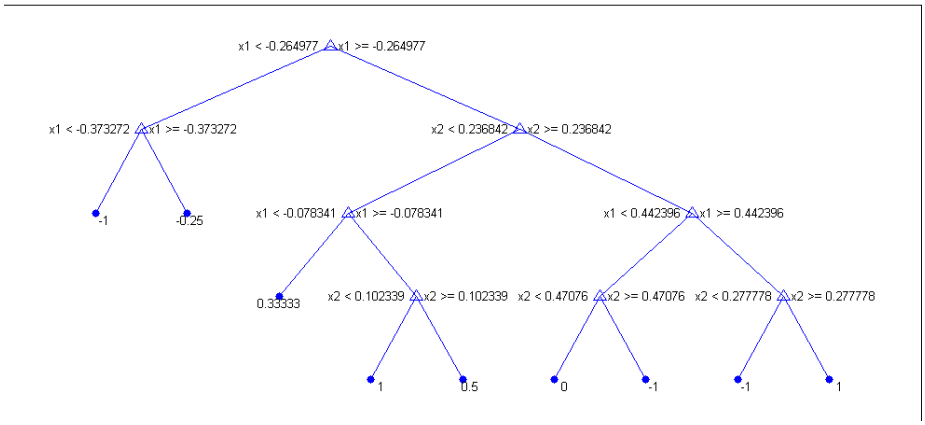


Figure 4: Plot of the tree model with default parameters for the dataset 'example_dataset_1.mat'

[2]Deterministic dataset: A dataset where all instances than share all attributes values, will also be from the same class.

In figure 4 we can see than this tree is not fully build. Due than not all the final tree leafs are represented by one single class instances.

The training error on this tree is 0.2430, this is because the default settings of the tree do not specify it to be a full decision tree, in fact the default settings specify than it only can split a node if there at least ten instances represented on it. This is a post-pruning constrain, which are set in order to avoid the tree from growing silly and overfitting every single element when there are many outliers on the training dataset.

As we already knew than the default value of the *minparent* value was 10, and we knew how the dataset elements distributed from the previous block, **Question block 1**. We expected to have an training error different greater than zero, as it out performs.

4 *Find the parameterization of the tree that allows you to build the full tree and plot the result on the training set. Which is the value of the training error?*

For training the full tree we used the Matlab function *tree=classregtree(X,Y,'minparent',1)*, with the new parameter 'minparent', which specifies the minimum size of a node in order to split it, set to 1. So if a node contains at least one element represented it can split it if necessary, in order to separate elements from the classes represented on it.

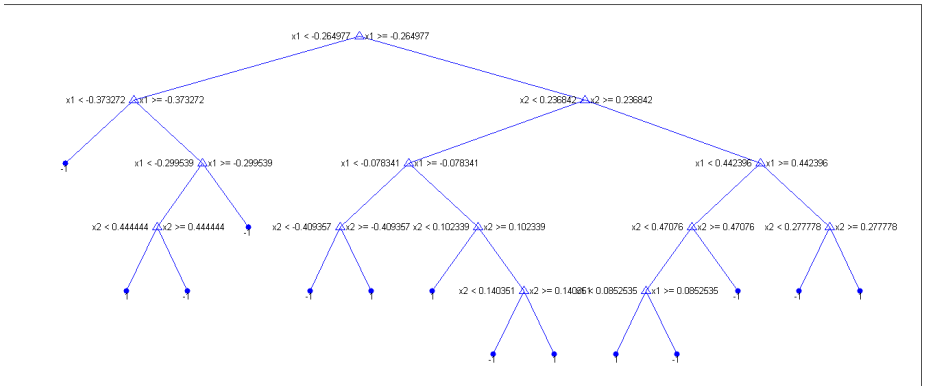


Figure 5: Plot of the tree model with default parameters for the dataset 'example_dataset_1.mat'

In figure 5 we can see than this tree is fully build. Due than all the final tree leafs are represented by one single class instances.

In this case the training error obtained is, indeed, zero.

3 Question Block 3

1 *Load the dataset 'example_dataset_1.mat'.*

We load that same dataset than in the previous blocks, **Question block 1** and **Question block 2**. Which as we said, is a non-linearly separable dataset.

2 *Create a function that given a dataset creates the K folds by storing the indexes corresponding to training and test for each of the folds.*

We created a new Matlab function named 'kfoldIndexer', it performs as *kfolds = kfoldIndexer(data,k)*, where 'kfolds' are the 'data', dataset, instances indexes separated in 'k', equal-sized (approx), blocks randomly.

3 *Compute and report each class frequency value for the original problem and for each of the training and test partitions executing your code using K=10.*

Class frequencies for the whole 'example_dataset_1.mat' dataset:

- Positive: 0.6168
- Negative: 0.3832

We can observe than the Positive class has a higher presence, almost double, on the loaded dataset than the Negative class.

Class frequencies for the generated k-folds using the 'kfoldIndexer' code with 'k' set to 10:

	Generated K - folds from the loaded data, K = 10									
Class	1	2	3	4	5	6	7	8	9	10
Positive	0.4545	0.6364	0.7273	0.3636	0.5455	0.3636	0.6364	0.8	0.9	0.8
Negative	0.5455	0.3636	0.2727	0.6364	0.4545	0.6364	0.3636	0.2	0.1	0.2

Table 1: Obtained 10-Folds class frequency table.

In table 1 we can see how each class instances are distributed among the randomly generated data-folds.

As this distribution is random, it is important to check and look at this frequencies in order to avoid undesirable distributions that could lead us to wrong assumptions. In our case we find that the data has distributed "good enough" to proceed using this distribution. Although we know that some folds are balanced the other way around than the original dataset and that fold 9 has really small presence of Negative class elements. On the other hand, we have that most folds have more presence of class Positive,

as the original dataset, and that all folds have presence of both classes greater or equal to 20%, with the exception of fold 9, which we have already mentioned.

4 Question Block 4

1 *Load the dataset 'example_dataset_1.mat'.*

We load that same dataset than in all previous blocks. Which as we said, is a non-linearly separable dataset.

2 *Use the cross-validation function to create a set of 5-fold indexes*

Class frequencies for the generated k-folds using the 'kfoldIndexer' code with 'k' set to 10:

	Generated K-folds, $K = 5$				
Class	1	2	3	4	5
Positive	0.6364	0.6364	0.6190	0.6190	0.5714
Negative	0.3636	0.3636	0.3810	0.3810	0.4286

Table 2: Obtained K-Folds class frequency table, on the 'example_dataset_1.mat' dataset, for $K = 5$.

In table 3 we can see how each class instances are distributed among the randomly generated data-folds. We can observe than this time the frequencies match very well with the original dataset class frequencies (Positive: 0.6168; Negative: 0.3832).

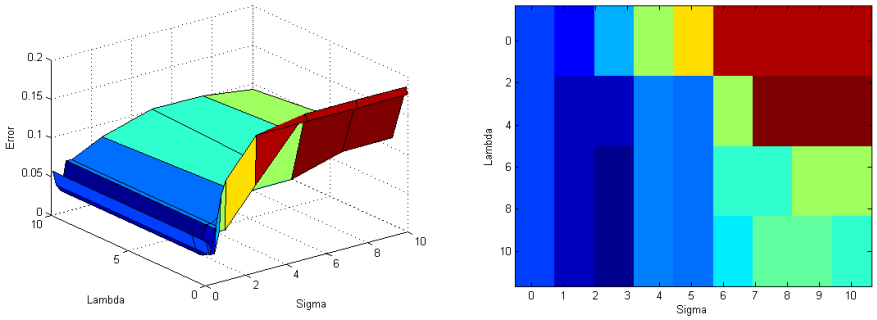
3 *Use the cross-validation set created for finding the best parameters of an RBF SVM. Plot the average error surface for each set of parameters. You may use a grid search, i.e. define a set of parameters to be tested and select the best one.*

The parameters than we used for the grid search on the cross-validation, with the folds proposed in the previous question, were:

$$\lambda = [0.01, 0.1, 1, 10] \quad \sigma = [0.1, 0.25, 0.5, 0.75, 1, 2.5, 5, 7.5, 10]$$

λ values \ σ values	0.1	0.25	0.5	0.75	1	2.5	5	7.5	10
0.01	0.0558	0.0468	0.0749	0.1117	0.1303	0.1775	0.1775	0.1775	0.1775
0.1	0.0558	0.0377	0.0377	0.0654	0.0649	0.1121	0.1866	0.1866	0.1866
1	0.0558	0.0377	0.0281	0.0654	0.0649	0.0931	0.0935	0.1117	0.1121
10	0.0558	0.0377	0.0281	0.0654	0.0649	0.0840	0.1022	0.1022	0.0931

Table 3: Mean error obtained from cross-validation using the previous presented K-folds, for $K = 5$.



(a) 3D plot

(b) 2D plot

Figure 6: Plot of the average errors obtained for the different values of the parameters λ and σ used in the cross-validation.

In case that more than one combination of parameters has the same error, we will choose as the best the one that generalises the best, than will be the less complex model and less constrained.

For this we defined a two steps criteria:

1. Simpler model: the one using the highest σ.
2. Less constrained: from the previously selected, the one using minimum λ.

In figure 6 we can observe than the first setting than minimizes the average error obtained from the cross-validation process is $\lambda = 1$ and $\sigma = 0.5$.

4 *Use the cross-validation set created for finding the best value for minparent. Plot the average error surface for each parameter value.*

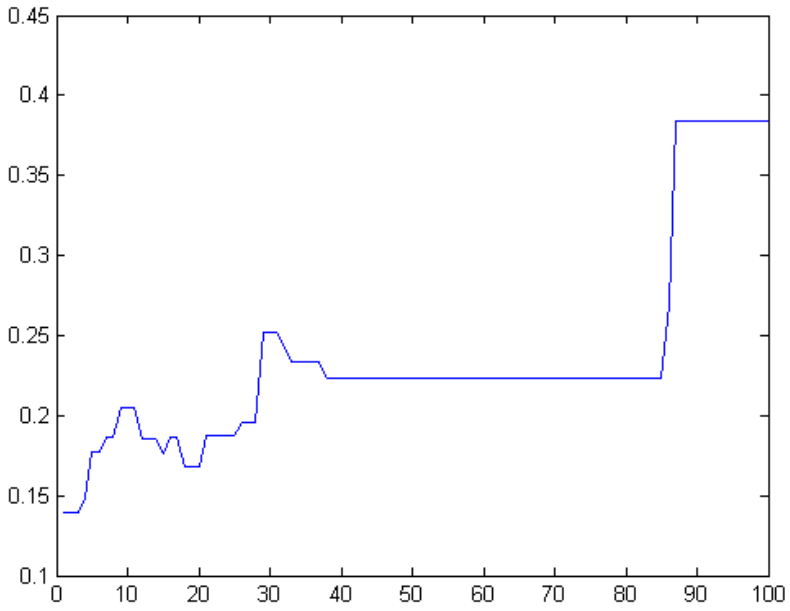


Figure 7: Plot of the average errors obtained for the different values of the parameter minparent used in the cross-validation.

In case that more than one combination of parameters has the same error, we will choose as the best the one that generalises the best, the less complex model, which will be the one using the higher 'minparent' value.

In figure 7 we can observe than first settings are minimizing the average error obtained from the cross-validation process.

minparent values

	1	2	3	4	5	6
Err	0.1394	0.1394	0.1394	0.1489	0.1771	...

Table 4: Mean error values from the best values of minparent, according to the information shown in figure 7.

Finally from table 4 we can observe that the best minparent setting is $minparent = 3$, because is the highest than maintains the minimum error value.

5 *Show the best validation errors obtained.*

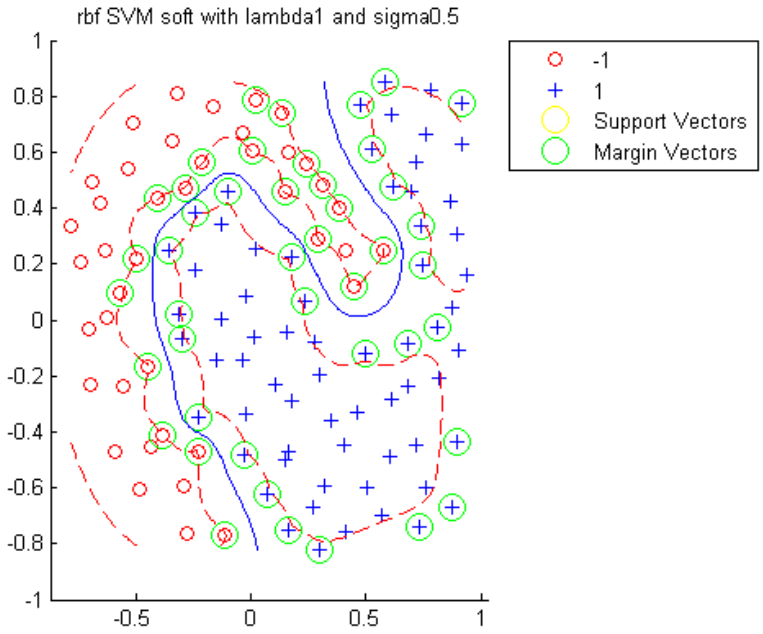


Figure 8: Plot of the data and a rbf SVM separating hyperplane with $\lambda = 1$ and $\sigma = 0.5$

In figure 8 we can see how outperforms the rbf SVM with the best parameters combination.

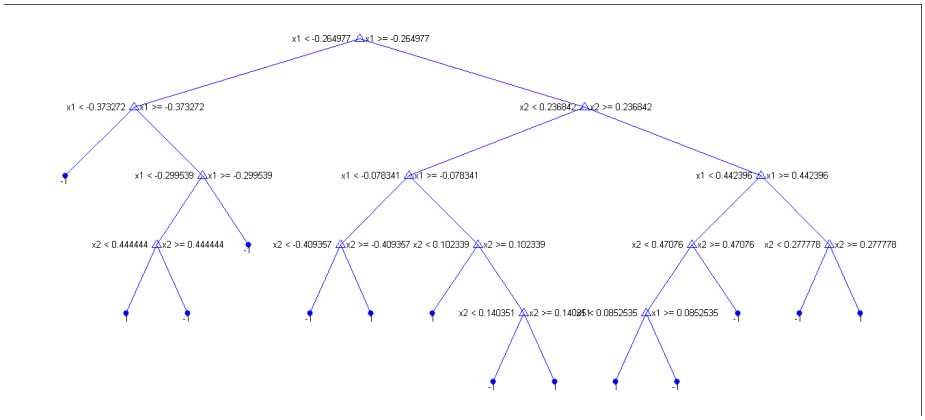


Figure 9: Decision tree model obtained when training with $minparent = 3$.

In figure 9 we can see how is the tree build with the best value for the minparent parameter found with the cross-validation process.

	Folds evaluated				
	1rst	2nd	3rd	4th	5th
SVM	0	0.0455	0.0476	0	0.0476
Tree	0.1818	0.1818	0.2381	0.0476	0.0476

Table 5: Validation errors obtained with the best settings chosen for each classifier during the cross-validation process, for $K = 5$.

In table 5 we can observe than for this classification problem, the rbf SVM outperform much better than the decision tree for the different combinations of parameters used in both cases.

5 Question Block 5

1

6 Question Block 6

1

7 Question Block 7

1

8 Question Block 8

1