Exercise 3: Linear Support Vector Machine

Introduction to Machine Learning

University of Barcelona

December 12, 2015

Authors:
Camps, Julià
Serra, Xavier

Contents

1 Question Block 1

1 *Load the dataset "example_ dataset_ 1.mat".*

We load the required dataset, which is linearly-separable, so we will be able to obtain a solution when applying hard-linear SVM.

2 *Run your training algorithm on that dataset.*

The applied algorithm does not accept misclassified points, all have to be at least at projection distance '1'. Let X be the set of all instances being classified, and let Y be the set of labels associated to each of these instances, so we can write that

$$\textbf{projection distance} = y_i(w^T x_i + b) \geq 1, \quad \forall x \in X$$

, with respect to the solution hyperplane.

The obtained model will be composed by a set of weights, each corresponding to one of the instances attributes, so that for two attributes, like in the used datasets, the solution will be a function of the attributes like the following: $ax_1 + bx_2 + c = 0$, where a and b are the weights of the attributes of the instances, and c is the offset of the hyperplane.

After running the training algorithm with the dataset we obtain a SVM solution model with strict boundaries, resulting in the following two dimensions hyperplane: $7.6950x_1 + (-4.8511)x_2 + 0.9752 = 0$.

3 *Plot the dataset and the separating hyperplane.*

In Figure 1 we can see as, indeed, the obtained model is completely fitting the training data. We have also drawn the margins of the nearest SVs to the hyperplane obtained.

4 *Identify the support vectors and explain how you know they are support vectors.*

For identifying SVs[1],in a hard margin limits linear SVM, we had to assume that our SVs will be all the instances placed at **projection distance** $= 1$, from the SVM generated hyperplane.

This approach will work for our example. However, we know that in definition of the dual approach of SVM, a support vector is defined as an instance with $\alpha \neq 0$, meaning that this instance has an influence over the obtained solution. This definition is much more precise than ours, because if all instances from each class are placed in parallel rows, using the first definition we would consider all of them as SVs, whereas if we used the dual solution only those that are actually weighted would be considered SVs.

[1]SVs: support vectors.

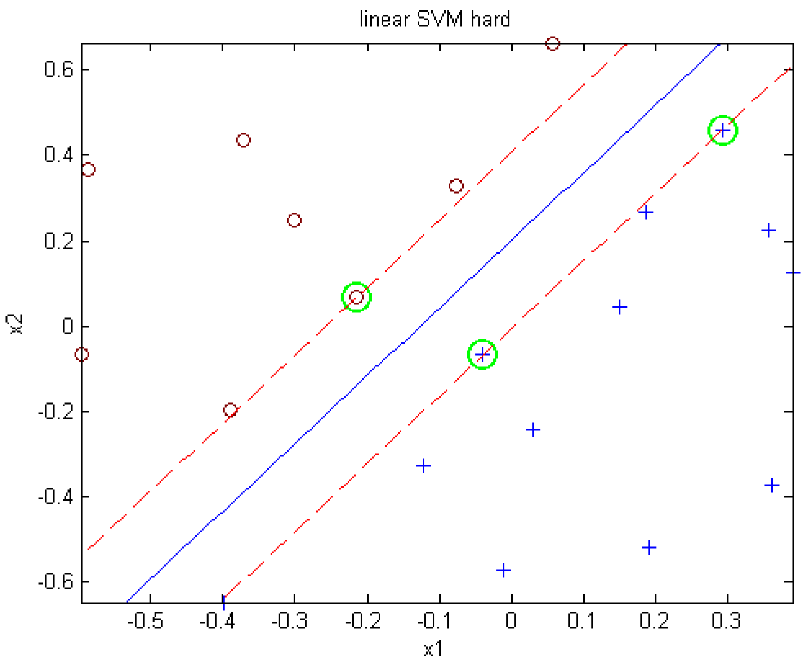

Figure 1: Hard lineal SVM on “example_dataset_1.mat” dataset

We also considered defining as SVs those instances that, when removed from the dataset, would lead to a different solution hyperplane. However, in the previously proposed case, with two parallel rows of instances, removing a single instance at a time would not modify the solution hyperplane, as the rest of instances of its class would keep the same solution. This would lead us to the false conclusion that there are no SVs.

From our method we obtained that the SVs for the hard lineal SVM, using the “example_dataset_1.mat” dataset, are the ones highlighted in green in the plot shown in Figure 1.

2 Question Block 2

For this block of questions we implement the soft margin linear SVM function, that will consider the number of **errors** in the optimization function as a parameter to minimize, trying at the same time to to maximize the margin of the model. Thus, we will have to find a trade-off between errors and margin size. This trade-off will be balanced by a λ weight parameter multiplying sum of the errors values.

So as smaller we place λ less taken into account will be the errors, and the contrary will happen when placing higher λ values.

In this section we will find SVs of two different types:

- The ones placed on the margin limits, as when visualizing the plots using hard-margin linear SVM
- Some values inside the margin limits, due to us allowing "errors" in this formulation.

So, from now on, we will see two different colors highlighting classification: green for the SVs placed over the margin limit, and yellow for the SVs inside the margin limit or misclassified.

Once a point falls in the margin, we will paint it with error color, without marking them independently on the fact if they are well or wrong predicted. We paint them with the same color because from the formulation point of view, are errors and the only difference is that as more badly are misclassified, greater will be the value that are adding to the errors sum value.

We are also asked to use a non-linearly-separable dataset for the third question and, as the used dataset up to this point is linearly-separable, we used a function proposed in the practicum description to create a simple non-linearly-separable dataset. We have named it "non_separable_dataset_1.mat", and we have used in for the questions that required a non-linearly-separable dataset in this **Question block**.

1 *Load the dataset "example_ dataset_ 1.mat".*

We load the same dataset that in the previous section.

2 *Consider the soft-margin formulation for $\lambda = 0$. Is it reasonable to think that the resulting hyperplane should be very similar to the case of hard-margin SVM in a separable problem? Why?*

No, it's not reasonable, because λ indicates the error penalty in our SVM. Therefore, if we set λ to a negligible value, or even to zero, the function won't care about minimizing the number of errors, and will only focus on maximizing the margin and, thus, minimizing the second norm of the solution.

From this we can conclude that, for $\lambda = 0$, we will get a wrong hyperplane, because it will not be separating our labeled data.

Here we add a plot of how does it look when we run the function with $\lambda = 0$ for the "example_dataset_1.mat" dataset.

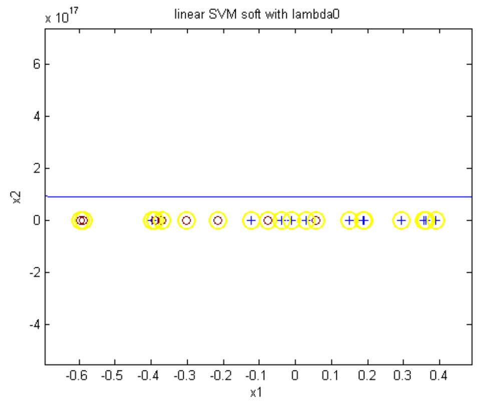

Figure 2: Plot scaled in the whole data and SVM solution range

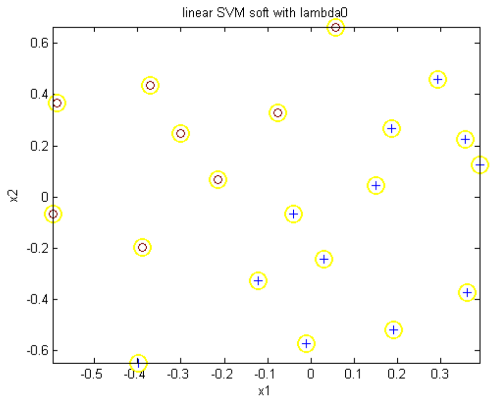

Figure 3: Plot scaled in the dataset data range

- In Figure 2 we can see as, indeed, it's not separating the data, as we have mentioned. This is due to the fact that we have integrated the restrictions in the minimization formulation. So when being multiplied by zero, the restrictions will simply not be taken into account at all.
- And in Figure 3 we can clearly see that the solution is not even in the visual range of the data being used for building it. This is due to the part of the formulation that uses the data being multiplied by 0, so it's not taken into account for building the solution.

3 *Run the training algorithm for non-separable data sets with $\lambda = 0$. Plot the*

dataset and the separating hyperplane. What do you observe? Hypothesize a reasonable explanation.

From here on we will be using the aforementioned custom dataset that we have constructed, named “non_separable_dataset_1.mat”.

Here we add a plot of how does it look when we run the soft-margin SVM with $\lambda = 0$.

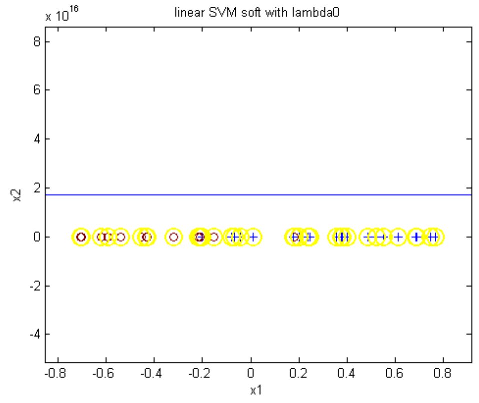

Figure 4: Plot scaled in the whole data and SVM solution range

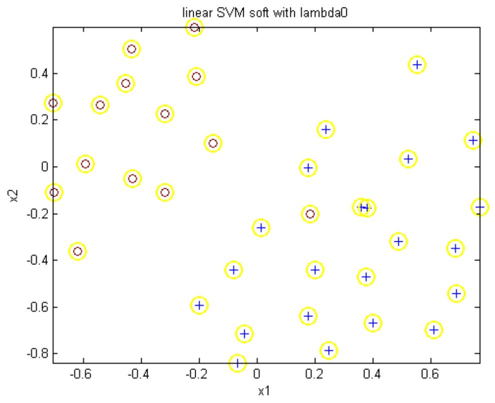

Figure 5: Plot scaled in the dataset data range

- In Figure 4 we can see as, indeed, there were not any significant difference with the previous plots, because only change was the data used, which is not taken into account since we set $\lambda = 0$.
- And in Figure 5 we can see it behaves exactly the same as the linearly-separable dataset, so our conclusions are confirmed from the closer view plot.

As we can observe, if we do not penalise miss-classification, the results obtained using non-linearly-separable data are equivalent to those using linearly-separable data.

4 *Plot the dataset and the separating hyperplane when training with* $\lambda = 10^{-2}$, $\lambda = 1$ *and* $\lambda = 10^2$.

Here we have the requested plots for the different λ values:

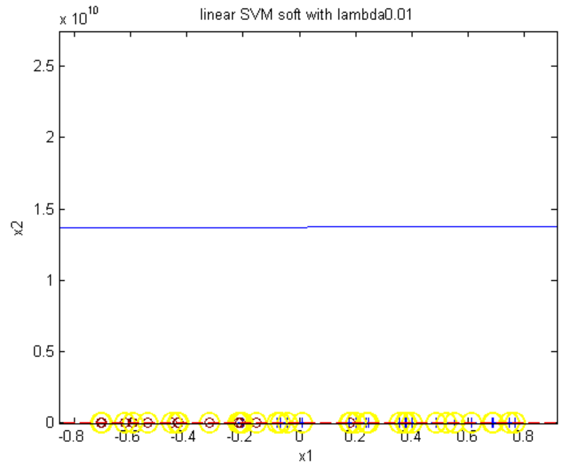

Figure 6: Plot of the data and its separating hyperplane with $\lambda = 0.01$

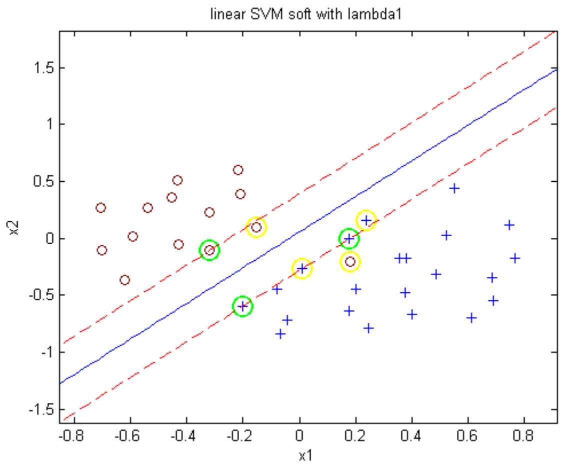

Figure 7: Plot of the data and its separating hyperplane with $\lambda = 1$

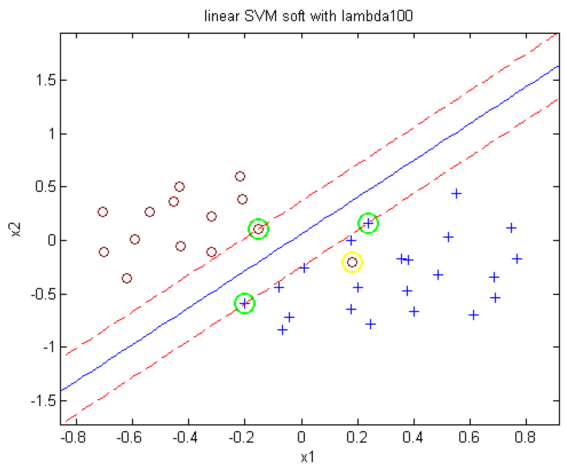

Figure 8: Plot of the data and its separating hyperplane with $\lambda = 100$

- In Figure 6 we can see that $\lambda = 0.01$ is a still too low value, so we have a similar situation as when using $\lambda = 0$.
- In Figure 7 we can see that, for $\lambda = 1$, we obtain a better shaped solution. This one has a still large margin, and, therefore, many instances fall within the margin boundaries. However, we can observe that it provides a soft-linear SVM acceptable solution, as it's trying to separate the elements of both classes.
- In Figure 8 we can see that, as we continue increasing the λ value, the provided solution resembles more to the one provided by an hard-linear SVM. The margin size is reduced in order to allow the hyperplane to classify more elements correctly. However, we know that, in this case, a hard-linear SVM would not be able to find a solution, as it's a non-linearly-separable dataset.

5 *Which is the expected value of u_i for the support vectors with margin equals 1?*

Their expected value is 0. So we can say that $E[u(x_i)] = 0$, $\forall x_i$ with **projection distance** $= 1$.

6 *Observe the values of u_i for $\lambda = 1e2$. Can you identify the SVs by observing those values? Describe the rule you use for identifying them.*

As we have already mentioned in the previous question, we can say that we are not able to do so, because some of them will have values equal to zero, due to the fact that they are lying in the margin boundary.

From the u_i values we can only deduce the support vectors that are inside the margin of the solution SVM. The reason is that the vector u is the slack vector, with a value for each instance in the dataset, which is 0 if it's at distance greater than 1 and in the correct side of the hyperplane. On the other hand, it has values greater than 0 if the instance is in the incorrect side of its corresponding margin. This positive value is the projection distance from the instance to it's corresponding margin.

So the method we will use in order to identify SVs consists in considering the projection distance vectors over the hyperplane from **Question block 1**, instead of the slack values as u_i does. Concretely, we take all those instances whose projection distance is lower than 1 as SVs of the solution SVM (in **Question block 1** we just considered values $= 1$, because we knew that errors were not allowed, but now we allow them as a part of the optimization problem).

3 Question Block 3

1 *Load the dataset "example_ dataset_ 2.mat".*

We load the requested dataset, which is a non-linearly-separable one, just like the one we were using in the previous block. However, in this case a lot more instances are placed in conflicting positions for the SVM algorithm.

2 *Run your training algorithm on that dataset for* $\lambda = 10^{-2}$*,* $\lambda = 1$ *and* $\lambda = 10^2$*.*

We run the algorithm as we did in the block 2.

3 *Plot the dataset and the separating hyperplane for each value of lambda.*

Here we have the requested plots for the different values of λ:

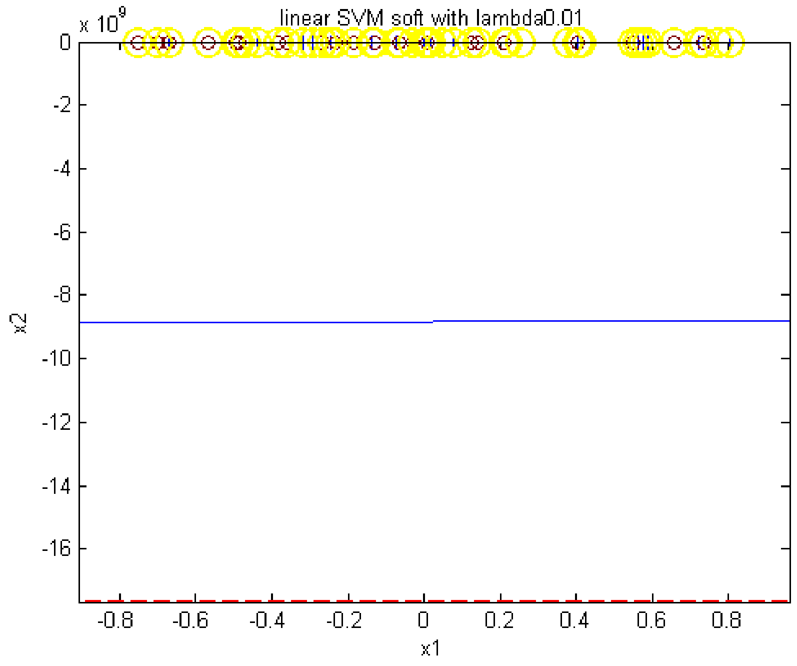

Figure 9: Plot of the data and its separating hyperplane with $\lambda = 0.01$

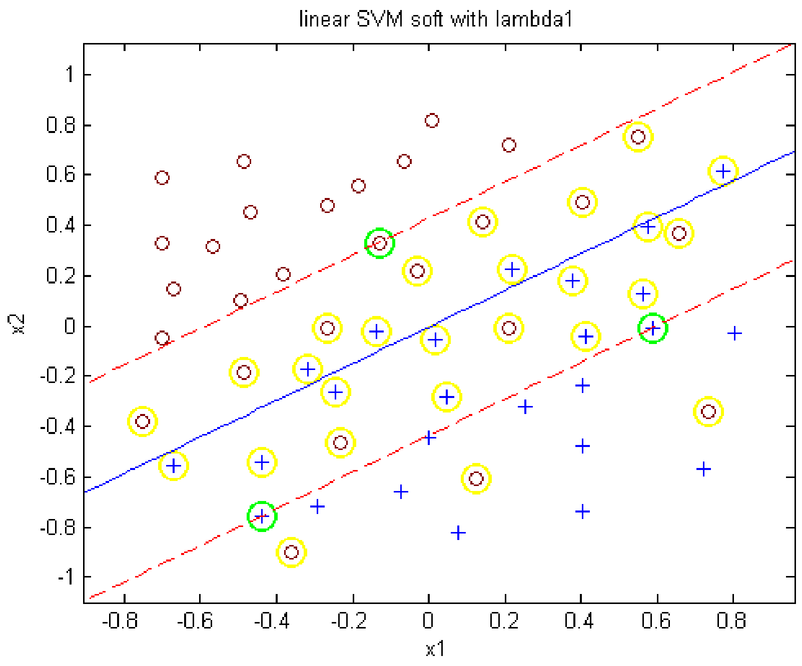

Figure 10: Plot of the data and its separating hyperplane with $\lambda = 1$

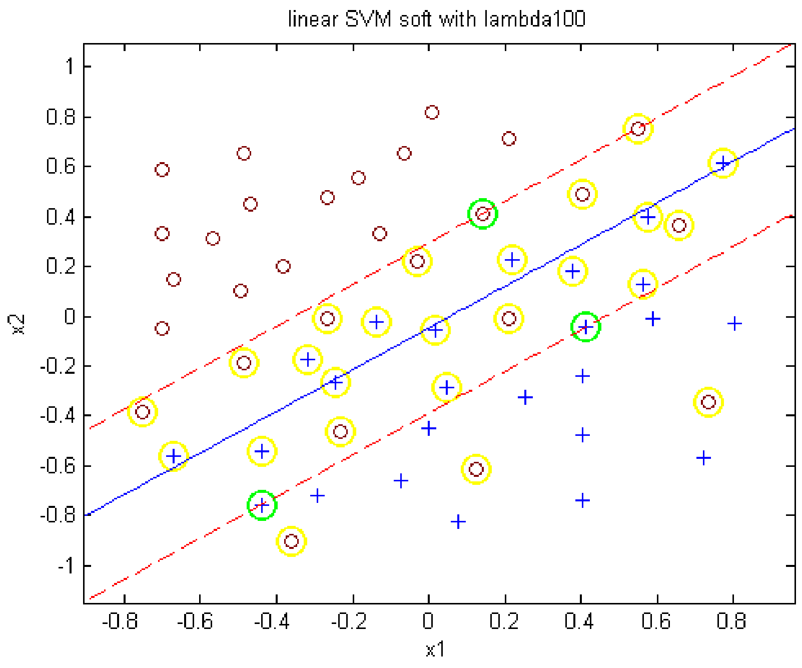

Figure 11: Plot of the data and its separating hyperplane with $\lambda = 100$

4 *Observe the values of u_i for $\lambda = 10$. Can you identify the SVs simply by observing those values? Describe the rule you use for identifying them.*

As we have explained in **Question block 2**, we are not able to determine all SVs from the u vector, as it is only showing the slack values.

For identifying the SVs we have used the same method that we described in **Question block 2**, and the support vectors are highlighted in green and yellow on the output plot when we run the SVM code.

Here we attach the obtained plot when running soft-linear SVM with $\lambda = 10$:

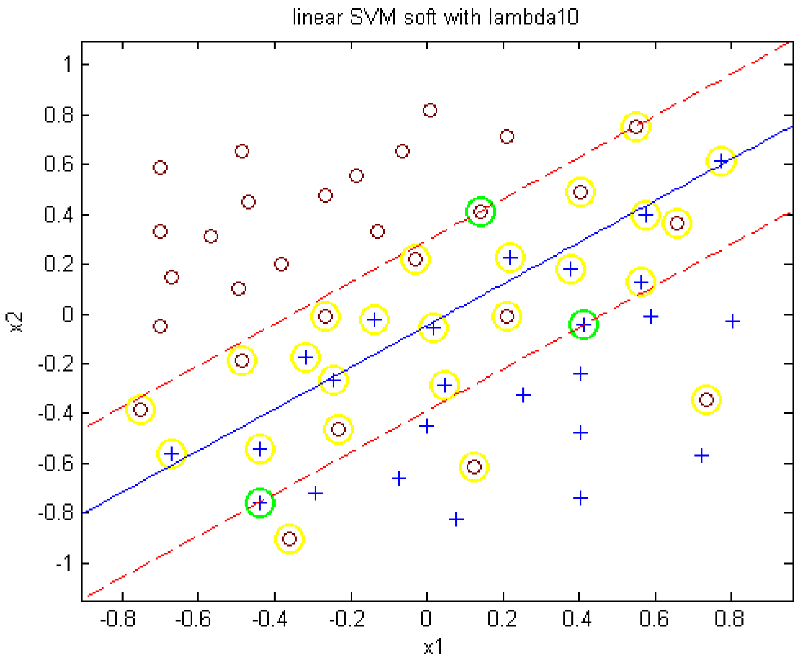

Figure 12: Plot of the data and its separating hyperplane with $\lambda = 10$

4 Question Block 4

1 *Load the dataset 'example_ dataset_ 2.mat'.*

We load the requested dataset, which is the same used in the previous block.

2 *Run your training algorithm on that dataset for $\lambda = 1e-2$, $\lambda = 1$ and $\lambda = 1e2$.*

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. But as our data has only two dimensions we decided to transform the representation of the obtained model, back to the primal representation, $y = w^T x + b$, in order to have a more explicative and simple representation, and to avoid having very long lists of weights in the report.

We will also explain some relevant information of the dual representation, such as the total number of SVs N_{SVs} and the number of SVs that are laying on the margin N_{SVm}.

After training, we obtained a model for each different value of λ:

- $\lambda = 0.01$:
  - N_{SVs} : 38
  - N_{SVm} : 3
  - w_1 : 0.0231
  - w_2 : -0.0335
  - b : 0.0183
- $\lambda = 1$:
  - N_{SVs} : 27
  - N_{SVm} : 3
  - w_1 : 0.0462
  - w_2 : -0.0552
  - b : 0.0197
- $\lambda = 100$:
  - N_{SVs} : 27
  - N_{SVm} : 3
  - w_1 : 0.0462
  - w_2 : -0.0552
  - b : 0.0197

As we can observe from the primal representation of the models, the model is not changing from changing λ from 1 to 100, this may mean that the margins are stuck in a zone than if they decrease they are increasing the distance of many fully misclassified, so minimization process won't change the provided solution for values of $\lambda > 1$.

This can be clearly seen in the plots presented on the next section 14 and 15, were we can see that, indeed, the plotted bounders are the same when increasing the λ from 1 to 100.

3 *Plot the dataset and the separating hyperplane for each value of lambda.*

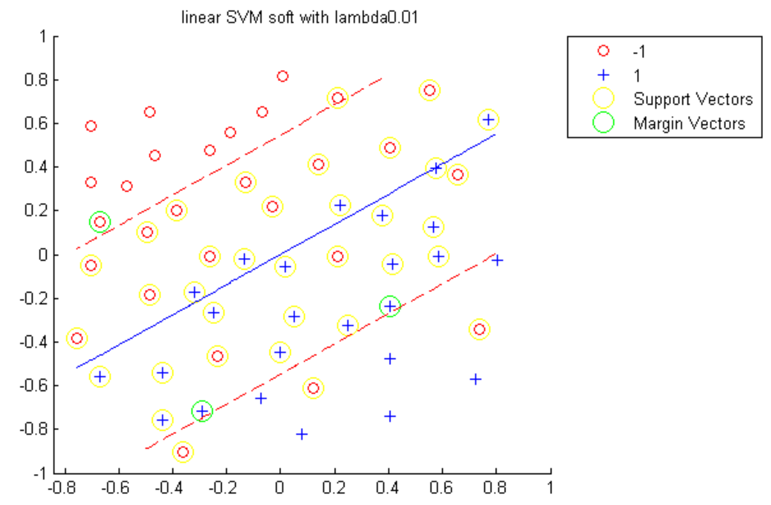

Figure 13: Plot of the data and its separating hyperplane with $\lambda = 0.01$ using the Dual

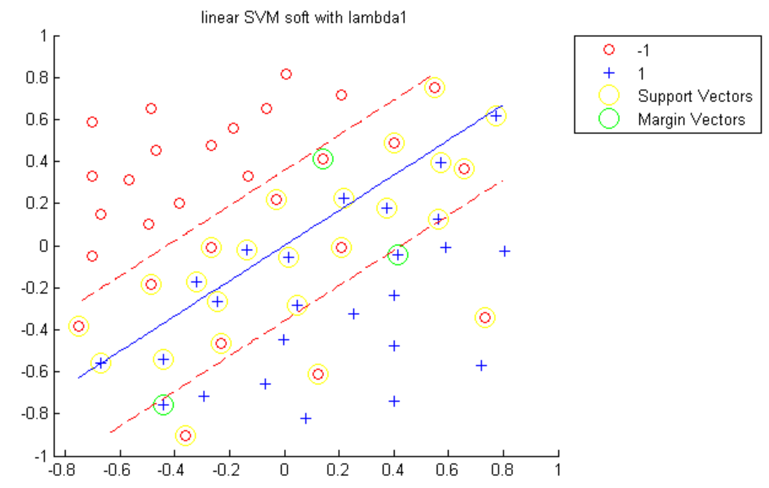

Figure 14: Plot of the data and its separating hyperplane with $\lambda = 1$ using the Dual

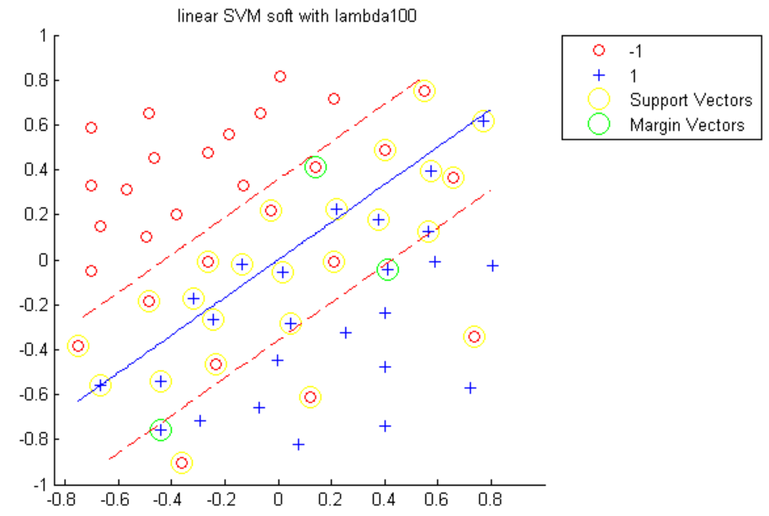

Figure 15: Plot of the data and its separating hyperplane with $\lambda = 100$ using the Dual

As we mentioned on the previous question, we can now see that the when we increase the λ value form 0.001 (13) to 1 (14) the retrieved SVM is different one from each other, in despite that when performing a second increase on λ to 100 (15) we realize that it had stuck previously as the model coincides with the one retrieved for $\lambda = 1$.

4 *Which are the expected values of ν_i for the SVs?*

The expected value will always be greater than zero, what can be written as: $\mathbf{E}(\nu_i) \quad \forall x_i \in SV_s, \quad \nu_i > 0$

Apart from this, we can extract more information just from the ν_i values for the SVs. We can distinguish between two different cases:

- $\nu_i = \lambda$: When the instances are on the wrong side of the margin.
- $0 < \nu_i < \lambda$: When the instances are exactly on the margin, and at projection distance $= 1$ from the hyperplane.

5 *Observe the values of ν_i for $\lambda = 10$. Can you identify the SVs simply by observing those values? Describe the rule you use for identifying them.*

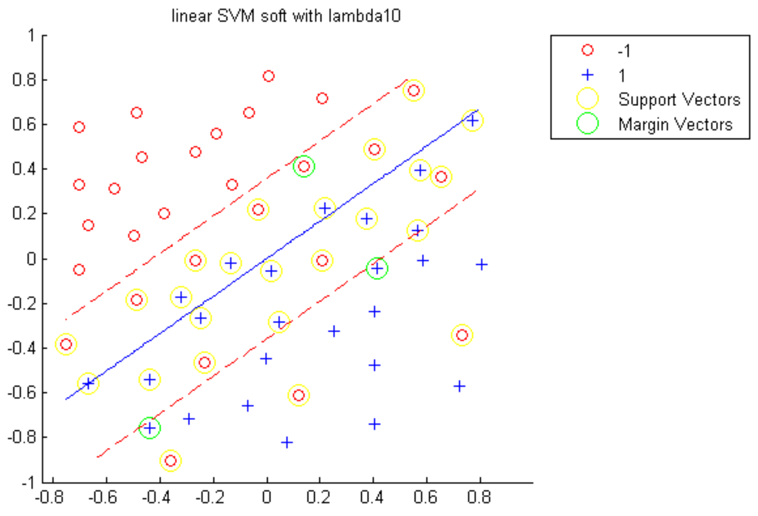

Figure 16: Plot of the data and its separating hyperplane with $\lambda = 10$ using the Dual

Yes we can. The rule that we defined is that a SV is a x_i (instance) for which it's ν_i value is greater than 0. So we can write that

$$\forall x_i \in X, \quad x_i \in SVs \iff \nu_i > 0$$

.

Note: When programming this part, we found that there were some problems, as some instances that should have a ν equal to 0 did not fulfill this condition. The reason was that their ν was actually a residual value resulting from the limited precision of the previous operations, extremely close to 0, but not exactly 0. Therefore, we had to relax a little bit the condition, and round slightly the values in order to address this issue.

6 *Is it easier to identify the SVs in the primal or in the dual? Why?*

We consider easier to identify SVs in the dual. With the dual we do not have any of the ambiguous cases that we had with the primal representation. Due the obtained vector ν, in comparison to the vector u obtained with the primal.

Recall: for the primal solution in the soft-linear SVM, we considered SVs those vectors, x_i, with projection distance less or equal than '1'.

With the dual, the solution obtained can be seen as a function of our SVs, and we have in ν the weights that indicate the relevance of the instances for our SVM.

So, just analyzing the values of ν, we are able to determine which instances are relevant for the SVM and, consequently, SVs.

5 Question Block 5

This question block and the following one are related with the treatment of unbalanced data, that is, a dataset in which one of the classes has sensibly more instances than the other one.

1 *Load the dataset* ***example_dataset_3.mat*** We do so by means of the usual instruction *load* and we split in between **data_5** and **labels_5**, which are later transposed in order to have an instance for each row. We do so because, later, it will be easier for us to work with the data in this form:

Listing 1: Data Loading

```
dataset_5 = load('../Data/example_dataset_3');
data_5 = dataset_5.data';
labels_5 = dataset_5.labels;
```

2 *Check how many examples we have for each class. Is the problem unbalanced? Why?*

We have:

- For the negative class: 13 instances
- For the positive class: 110 instances

This problem is clearly unbalanced, as there are almost 10 times as many positive examples as negative.

In case it is not linearly separable, working directly with an unbalanced problem may be dangerous. Usually, unbalanced data occur when it is difficult to obtain instances of one of the classes, and most often this implies that these few instances are of the utmost importance: e.g patients with an extremely rare disease. Therefore, it is more important to detect correctly those patients having the disease than diagnosing incorrectly a healthy person with it.

When using a typical SVM in an unbalanced dataset in which the small class is the most important one, it will treat both of them equally and we are likely to have as many errors of both classes. This is an undesired effect, which should be fixed.

3 *Search for the optimum value of δ for this problem. Plot the separating hyperplane and justify your choice of δ.*

We have decided to use the previously implemented function $train\_linearSVMsoft$, which applies a soft margin SVM to the given dataset, and has as an adjustable parameter λ. In order to find the best lambda, we have decided to try with different ones and evaluate the error of each of the obtained models. The values tried are:

$$\Lambda = \{1, 10, 100, 1000, 10000\}$$

Even though it is not the most accurate trial, working with different orders of magnitude is probably enough to give an intuition. In order to determine the goodness of a certain lambda we have counted the number of instances within both margins. Another possible goodness indicator would be the number of instances not in their half of the space: this indicator is not used, but it is also shown as additional information. The results obtained are:

λ	**#SVM errors**	**#Instances misclassified**
1	17	8
10	14	7
100	14	7
1000	14	7
10000	14	7

Table 1: Lambdas and errors

As we can see, once a certain lambda is achieved, the number of errors keep being the same. Therefore, we take $\lambda = 10$.

The resulting hyperplane, as well as both margins, are plotted in Figure 17

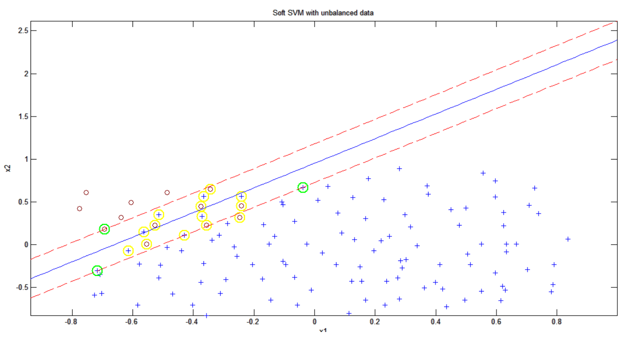

Figure 17: Hyper-plane with unbalanced data

4 *Is the result satisfying? Why?*

It depends. Objectively speaking, we have obtained less than 6% classification errors, and only the 11% of the instances are within both margins. Taking into account we faced a non-linearly separable problem, without additional information, we may consider it a quite good model.

However, we may consider, as already pointed out, that it is more important to correctly classify the instances in the minor class than in the major one. If this was the case, we should take a look at the number of instances of each class misclassified in Table 2:

Class	**#Misclassified instances**
+1	3
-1	4

Table 2: Num misclassified instances

As we can see, SVM gives the same importance to the errors of both classes, and, therefore, of the 7 misclassified instances, 4 of them belong to the minor class. Thus, if our goal was to correctly classify this class, this result is not satisfying at all, most of the misclassified instances belong to the priority class.

5 *Compute and report the training error rate.*

As already shown in Tables 1 and 2, we have, with the chosen lambda, 14 instances between the margin (11.4%) and 7 misclassified ones (5.7%).

6 Question Block 6

Theoretical advice: In general, the cost of an error in a critical class is set by the user/client according to the application. However if we don't have a priori knowledge it is sensible to use a balancing weight so that an error on the majority class has less importance. A possible balancing weight is to use the quotient between the cardinality of the minority class set over the cardinality of the majority class set.

1 *Modify your code to take into account the balancing weight.* ***Hint:*** *instead of using the compact constraint $y_i(a^T x_i + b) \geq 1 - u_i$, use separate constraints for the class $+1$ and -1. Look at the first slides regarding* ***A convex optimization view to linear classification****.*

In order to take into account the balancing weight, we have modified the original soft margin SVM into:

```
cvx_begin
    minimize $||a||_2 + \lambda \times (u^T\mathbf{1} \times weight_1 + v^T\mathbf{1} \times weight_2)$
    subject to
        $r \times a_i + b \leq -1 + v$
        $s \times a_i + b \geq 1 - u$
        $u \geq 0$
        $v \geq 0$
cvx_end
```

This way, the penalizing effect obtained by means of u and v can be regulated using $weight_1$ and $weight_2$. Each of these weights is assigned to either the positive or the negative classes, and the one with the highest weight will have its errors more severely punished. This way, in case we want to prioritize the minor class, we can assign it a weight ten-fold the other. In our case, we have decided to follow the given advice, and assign each weight according to the cardinality of its class.

2 *Search for the optimum value of λ for this problem. Plot the separating hyperplane and justify your choice of λ.*

In this case we have decided to test the same lambda values as in **Question 5** in order to make the conclusions more reliable. Therefore, we have used:

$$\Lambda = \{1, 10, 100, 1000, 10000\}$$

And the errors obtained are shown in Table 3:

λ	**#SVM errors**	**#Instances misclassified**
1	51	22
10	27	15
100	22	13
1000	22	13
10000	22	13

Table 3: Lambdas and errors

Once again, the best value for *lambda* is 100, although any of the greater values have obtained the same results.

At first sight it seems as it is clearly worse than the previous case, as the number of errors of both types is significantly greater. However, the aim of this modification was a very specific one: to prioritize one of the classes. In order to check if this condition has been fulfilled, we must see the number of misclassified of each class, in Table 4:

Class	**#Instances misclassified**	**#SVM errors**
+1	13	20
-1	0	2

Table 4: Num misclassified instances

This time we have considered both kinds of errors in order to make it more informative. We can see that no negative instance has been misclassified, but two of them are within the margin. The conclusions we extract from this are:

- The total number of errors has greatly increased (approximately a 50% more).
- The negative class (the minor one) is almost entirely well classified.
- We have fulfilled our goal, to correctly classify most of the minor class.
- If we wanted to maximize this bias, and leave more negative instances outside the margin, we could increase the difference between weights.

In the real example aforementioned, we would have incorrectly diagnosed many healthy people as having the disease but, in exchange, all those actually ill would have been properly diagnosed. It is just a matter of priorities, depending on the goal being pursued.

In order to provide a more graphical approach, we provide the projection of the hyperplane, as well as both margins, in Figure 18:

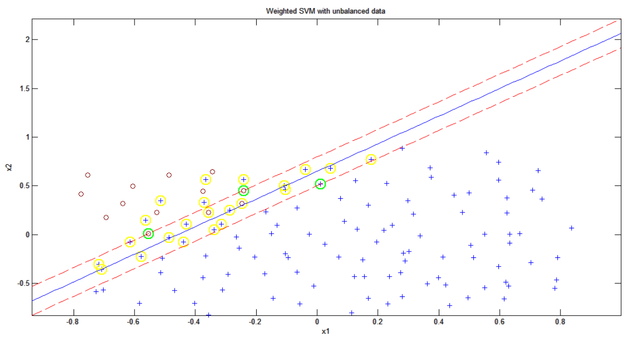

Figure 18: Hyper-plane with weighted SVM

In this image we can see the difference between this hyperplane and the one obtained with the normal soft SVM in Figure 17. It has moved in order to place every single negative instance in one side, and most of the positive ones in the other.

3 *Is the result satisfying? Why?*

Once again, it depends. Generally speaking, there are both more classification errors and instances within the margin than before, so, in terms of accuracy, this result is less satisfying than the one in **Question block 5**. However, as already mentioned, if our goal is to properly classify all negative instances, this result is clearly more satisfying.

Therefore, we can not really classify a certain result using a global criteria, as each problem has its own goals, which are evaluated in a different way.

In this concrete case, we wanted to create a weighted SVM which gave more importance to one of the two classes, and we have clearly succeeded in doing so, so we are satisfied with the result.

4 *Compute and report the error rate. Is this error rate smaller than the one obtained in block 5? Why?*

As already shown, we have 22 instances within the margin (17.9%) and 13 misclassified ones (10.6%). Both of them are higher than the ones in **Question block 5**. The reason is that, by forcing the SVM to give much more importance to a certain class, it has not been able to attain the global optima in terms of misclassified instances. So, the gain of accuracy in the minor class has been obtained by means of an overall loss of accuracy.

5 *Use the balancing weight to define a weighted error rate and compute the weighed training error rates for the models in block 5 an 6. Is this error rate smaller than the weighted error for the model in block 5? Why?*

In this last question we were asked to create a new error measure which took into account the weight of each misclassified instance. Therefore, for each misclassified instance we have added the weight corresponding to its class to the total error, which has been later divided between the weight of the whole dataset in order to provide a percentage of error. Then, assuming a dataset $\mathcal{D}$, divided into two classes $\mathcal{D}_N$ and $\mathcal{D}_P$ such that: $(\forall a_i \in \mathcal{D}_N | label(n_i) = -1)$, $(\forall p_i \in \mathcal{D}_P | label(a_i) = +1)$ and $\mathcal{D} = \mathcal{D}_N \bigcup \mathcal{D}_P$, we can express the error as:

$$error_w = \frac{\sum\limits_{n_i \in \mathcal{D}_N} weight(n_i) \times miss(n_i) + \sum\limits_{p_i \in \mathcal{D}_P} weight(p_i) \times miss(p_i)}{\sum\limits_{a_i \in \mathcal{D}} weight(a_i)} \tag{1}$$

, where

$$miss(a_i) = \begin{cases} 1 & \text{if } a_i \text{ misclassified} \\ 0 & \text{if } a_i \text{ is properly classified} \end{cases}$$

We have calculated this error for both models, the one used in **Question block 5** and the one used in this current question block. The results are shown in Table 5:

Question block	Weighted error
5	34.41%
6	16.78%

Table 5: Weighted error

We can now compare these errors with the other two types of errors used. In order to make it easier to compare, we have unified all of them in a single Table 6:

Question block	SVM error	Misclassified instances	Weighted error
5	13.0%	5.7%	34.4%
6	18.7%	10.6%	17.2%

Table 6: Error comparison

We can see here that in the first two errors, the most *typical* ones, the model obtained in **Question block 5** obtains better results. It is a logical outcome, as the model aims at the data split with less misclassifications. However, in the new weighted error it is the other way around, and the weighted SVM of **Question block 6** obtains better results. Once again, it is the predicted outcome, as this model aimed to minimize the weight of the misclassified instances.