1 Perceptron and Margins

In this problem we will be working exclusively with a small dataset S in Figure 1. Consider all red (diamond) training examples as negative instances ($y_\diamond = -1$) and all blue (cross) training examples as positive instances ($y_\times = +1$). We will explore two variations on the standard perceptron algorithm you learned in class, and relate the perceptron to a hard-margin SVM.

(a) In class you learned the perceptron algorithm for finding an *unbiased* hyperplane that separates two classes with 0 training error. There is no such hyperplane, however, for the dataset S in Figure 1. A simple way to incorporate a bias term b is to augment the weight vector $\mathbf{w}$, such that $\mathbf{w}_{biased} = \langle \mathbf{w}, b \rangle$, and augment all training examples such that $\mathbf{x}_i := \langle \mathbf{x}_i, 1 \rangle$. Implement a biased perceptron learning algorithm and train it on S, following the order of training examples given in Figure 1. Compute $\mathbf{w}_{biased}$. Plot the resulting hyperplane. Does the resulting hyperplane maximize the margin between the two classes? (Note for all perceptron implementations in this problem: initialize your weight vector $\mathbf{w}$ to the zero vector.)

Solution

No, the hp doesn't maximize the margin, as it's closer to one class than another

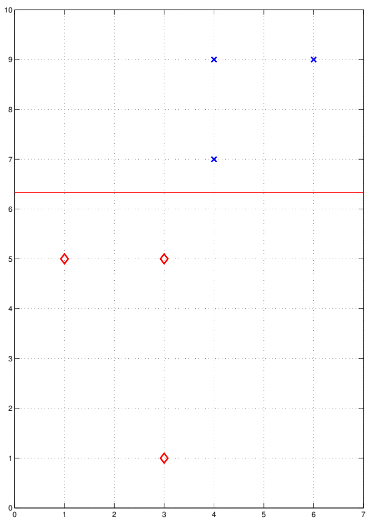


Figure 1: Solution part *a*

(b) Sketch a hyperplane that achieves the largest hard margin in S *without* using an SVM package. Clearly mark all support vectors in your diagram. What is the weight vector $\mathbf{w}_{opt}$ and bias b_{opt} of the maximum margin hyperplane that achieves a functional margin of 1 for the support vectors? What is the geometric margin γ_{opt}? Also indicate which dual variables α_i are non-zero. Show all work leading you to your answer.

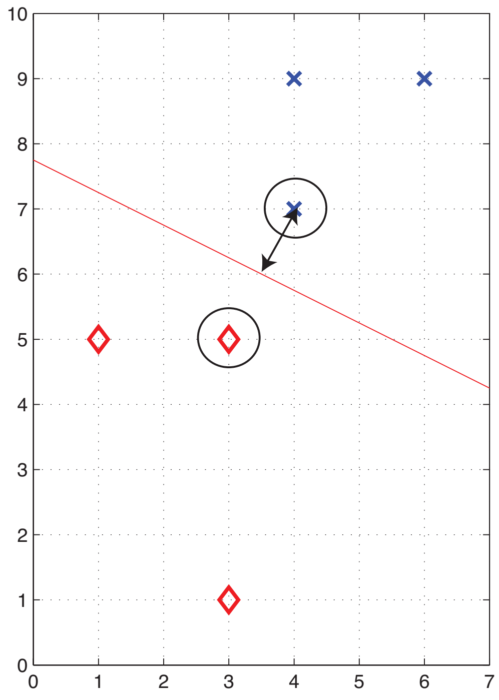


Figure 2: Solution part *b*

Solution

$$\mathbf{w}_{opt} = < \frac{2}{5}, \frac{4}{5} >$$

$$b_{opt} = -\frac{31}{5}$$

$$\gamma_{opt} = \frac{\sqrt{5}}{2}$$

Support vectors are marked in Figure 2 with circles. They are the only instances with non-zero alphas.

(c) All of a sudden you get a strong and unstoppable urge to implement your own SVM. Unfortunately you have no background in convex optimization, so you decide to try something very simple instead. Being clever, you think up a way to modify the biased perceptron algorithm to allow you to 'tune' its margin. One simple way to accomplish this is to update the weight vector whenever a training example is within a user-specified fraction $\beta \in [0, 1)$ of the maximum margin γ.

$$\frac{y_i(\mathbf{w}^T\mathbf{x}_i + b)}{\|\mathbf{w}\|} < \beta\gamma \tag{1}$$

Set $\beta = 0.5$ and run your modified perceptron algorithm until convergence (train on the examples in the same order as indicated in Figure 1). Plot the resulting hyperplane (you found γ in part b). What do you observe?

Solution

The hp for $\beta = 0.5$ is not better than the non-margin version in part a. For other initial conditions, the original hp could be worse, however, the $\beta = 0.5$ is guaranteed to have the margin of at least 0.5γ.

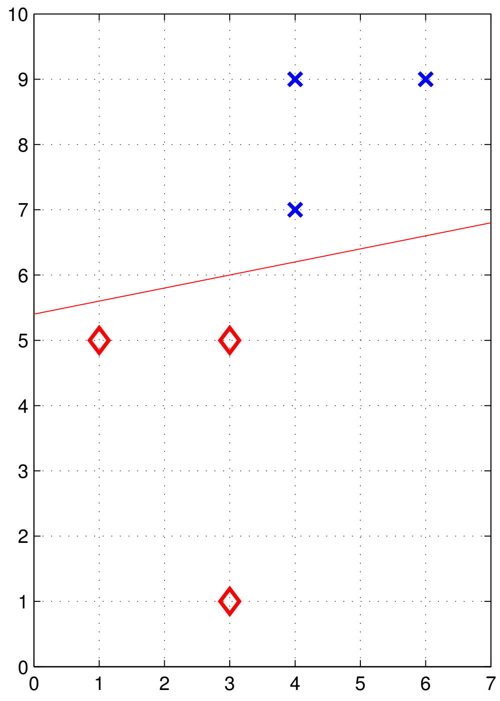


Figure 3: Solution part c

(d) Rerun your algorithm until convergence for a range of values $\beta \in${*0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95*}. Plot (on one graph) a hyperplane for each value of β (10 values total) overlaying the optimal hyperplane you found in part b.

Solution

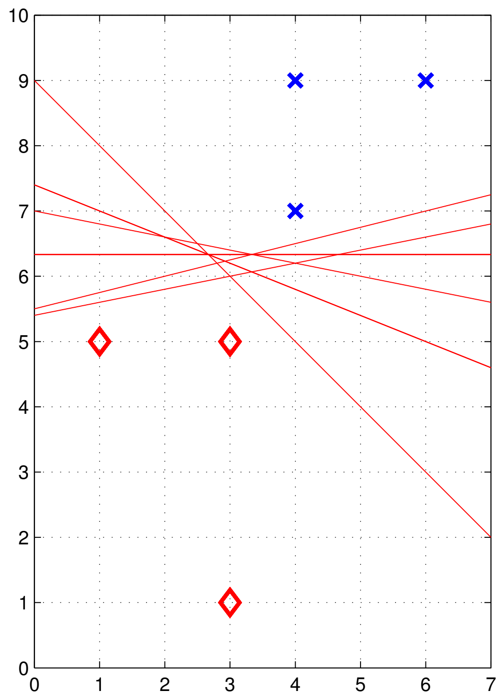


Figure 4: Solution part *d*

(e) What do you observe? What do you expect to happen if you rerun this modified perceptron learning algorithm with different initializations of the $\mathbf{w}$ vector, and different ordering of the training examples, for different values of β, as β approaches 1? You don't need to perform the experiments, but justify your answer.

Solution

Observation 1: For different orderings and initial conditions and small values of β we expect larger variance in the learned hyperplanes. For β close to 1, we expect the learned hyperplanes to be fairly consistent and close to the optimal hyperplane. **Observation 2:** For larger values of β, it takes longer for the perceptron learning algorithm to converge.

In the rest of the problem we will try to prove the mistake bound of this modified perceptron algorithm with $\beta = 0.5$. Each remaining part of the problem is designed to lead you to the final bound. If you are unable to complete any single part, you can use the provided result as a given for the parts that follow.

Assume that all datapoints are rescaled into a ball of radius 1, so that $R^2 = 1$ in all of the bounds you have seen in class. This does not affect the generality of the proofs.

(i) You have already seen in class, that we can bound the *squared length* of the weight vector in the standard perceptron on every mistake update as follows:

$$||\mathbf{w}_{k+1}||^2 \leq ||\mathbf{w}_k||^2 + 1$$

Show that this bound implies the following bound on the *length* of the weight vector in the standard perceptron:

$$||\mathbf{w}_{k+1}|| \leq ||\mathbf{w}_k|| + \frac{1}{2||\mathbf{w}_k||}$$

Solution

$$||\mathbf{w}_{k+1}||^2 \leq \left(||\mathbf{w}_k|| + \frac{1}{2||\mathbf{w}_k||}\right)^2$$

$$||\mathbf{w}_{k+1}|| \leq ||\mathbf{w}_k|| + \frac{1}{2||\mathbf{w}_k||}$$

(ii) Now let's modify the bound above to account for a modified perceptron learning rule specified in part c of this problem. **Show that** for this modified perceptron learning algorithm, the following bound on the *length* of the weight vector holds:

$$||\mathbf{w}_{k+1}|| \leq ||\mathbf{w}_k|| + \frac{1}{2||\mathbf{w}_k||} + \frac{\gamma}{2}$$

Hint 1 It may be helpful to recall that $||\mathbf{w}_{k+1}||^2 = ||\mathbf{w}_k + y\mathbf{x}||^2$ and that a squared norm of any vector $||\mathbf{v}||^2$ can be written as $\mathbf{v} \cdot \mathbf{v}$. Additionally, the following inequality may be useful: $\sqrt{1+x} \leq 1 + \frac{x}{2}$. In contrast to the perceptron update rule you saw in class, you need to consider how the new update rule (that incorporates the margin) will affect the upper bound.

Hint 2 You may also use geometric intuition instead, and show that the growth in $||w_{k+1}||$ is also affected by a nonzero projection of $y\mathbf{x}$ on $\mathbf{w}$ under the new update rule, and use that to help you derive the upper bound.

Solution

Update rule:

$$\mathbf{w}_{k+1} = \mathbf{w}_k + y_i\mathbf{x}_i$$

$$\mathbf{w}_{k+1} \cdot \mathbf{w}_{k+1} = (\mathbf{w}_k + y_i\mathbf{x}_i) \cdot (\mathbf{w}_k + y_i\mathbf{x}_i)$$

$$||\mathbf{w}_{k+1}||^2 = ||\mathbf{w}_k||^2 + 2y_i\mathbf{w}_k \cdot \mathbf{x}_i + y_i^2||\mathbf{x}_i||^2$$

Factor out $||\mathbf{w}_k||$, and since $||\mathbf{x}_i|| \leq 1$

$$||\mathbf{w}_{k+1}|| = ||\mathbf{w}_k||\sqrt{1 + \frac{2y_i\mathbf{w}_k \cdot \mathbf{x}_i}{||\mathbf{w}_k||^2} + \frac{1}{||\mathbf{w}_k||^2}}$$

Using the given hint $\sqrt{1+x} \leq 1 + x/2$

$$||\mathbf{w}_{k+1}|| = ||\mathbf{w}_k|| \left(1 + \frac{y_i\mathbf{w}_k \cdot \mathbf{x}_i}{||\mathbf{w}_k||^2} + \frac{1}{2||\mathbf{w}_k||^2}\right)$$

Multiplying through with $||\mathbf{w}_k||$

$$||\mathbf{w}_{k+1}|| = ||\mathbf{w}_k|| + \frac{y_i\mathbf{w}_k \cdot \mathbf{x}_i}{||\mathbf{w}_k||} + \frac{1}{2||\mathbf{w}_k||}$$

The middle term is the geometric distance of an instance from the hyperplane. By our update condition, we only cause an update if the instance is within the specified margin $\beta\gamma$. For this problem, $\beta\gamma = \gamma/2$, so we get

$$||\mathbf{w}_{k+1}|| = ||\mathbf{w}_k|| + \frac{1}{2||\mathbf{w}_k||} + \frac{\gamma}{2}$$

We can also get the same result by imagining that a single instance $\mathbf{x}_i$ is replaced by a linear combination of two instances, one parallel and another perpendicular to the hyperplane. The instance parallel to the hp will be consumed as part of the original perceptron bound, while the instance perpendicular to the hp will contribute at most $\gamma/2$ to the original bound, as desired.

(iii) **Show that** the following simpler bound holds if $||\mathbf{w}_k|| \geq \frac{2}{\gamma}$:

$$||\mathbf{w}_{k+1}|| \leq ||\mathbf{w}_k|| + \frac{3\gamma}{4}$$

Solution

Notice that $\frac{1}{2||\mathbf{w}_k||} + \gamma/2$ is a monotonically decreasing function of $||\mathbf{w}_k||$, thus it's sufficient to find one value $\mathbf{w}_{min}$ for which the function is equal to some constant c, to conclude that it's bounded by c for all $\mathbf{w}_k > \mathbf{w}_{min}$.

It can be easily checked that the given value for $\mathbf{w}_{min}$ results in $c = \frac{3\gamma}{4}$ and thus generates a valid upper bound for the original expression in the previous part.

(iv) **Finally show that** the total number of updates (mistakes) K for this modified perceptron algorithm can be bounded by $K \leq 8/\gamma^2$. Use the lower bound on $||w_k||$ together with the upper bound above to arrive to the desired upper bound on K. Notice that the lower bound on $||w_k||$ is the same as for the perceptron algorithm you analyzed in class.

Solution

By induction (noting our initial condition for $||\mathbf{w}_{k-1}||$ from the previous part), $||\mathbf{w}||$ as a function of K is given by:

$$||\mathbf{w}_K|| \leq \frac{2}{\gamma} + K\frac{3\gamma}{4}$$

Combining with the provided lower bound on $||\mathbf{w}_k|| \geq K\gamma$, we solve for K to obtain

$$K \leq \frac{8}{\gamma^2}$$

Solution

In the bound in part ii, replace $\gamma/2$ with $\beta\gamma$ to obtain

$$||\mathbf{w}_{k+1}|| \leq ||\mathbf{w}_k|| + \frac{1}{2||\mathbf{w}_k||} + \beta\gamma$$

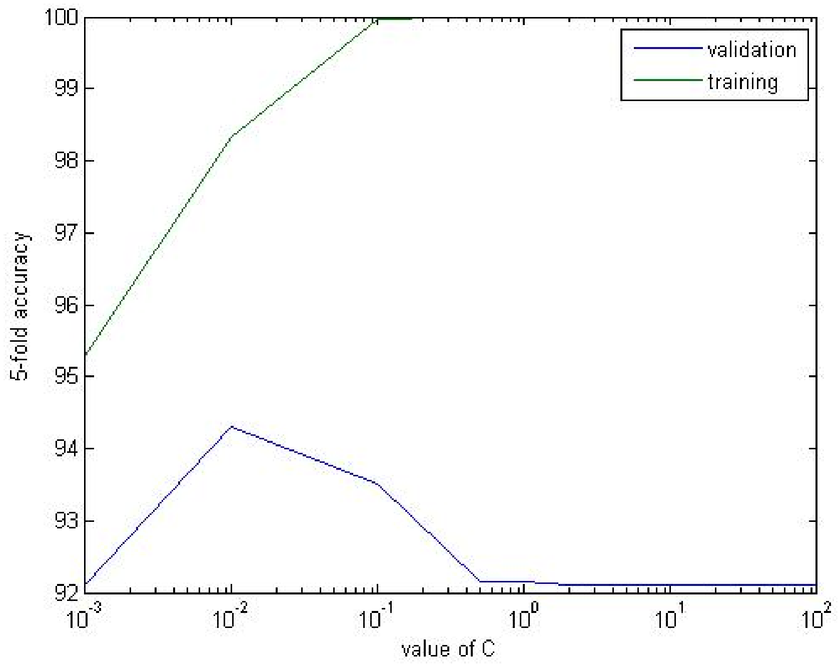
Maintaining that $||\mathbf{w}_k|| \geq 2/\gamma$, get the updated simpler bound on $||\mathbf{w}_{k+1}||$, we get

$$K \leq \frac{8}{\gamma^2(3-4\beta)}$$

Note that there are multiple bounds possible depending on your assumption on $||\mathbf{w}_k||$. The general trend that you observe is that with increasing β, the bound on the number of mistakes increases. This is consistent with your experiments.

2 Multi-class classification with SVMs

a) Randomly partition the training set into 5 equal-sized folds. Then, do 5-fold cross-validation for $C \in \{0.001, 0.01, 0.1, 0.5, 1.0, 2.0, 5.0, 10.0, 100.0\}$. Use the same C-value and folds for each binary SVM. Make sure that the only parameter you use with `svm_learn` is `-c`.
If the prediction $h(\mathbf{x})$ is different from the ground-truth class, we count it as an error. Plot a graph showing both training and validation accuracies (averaged across the 5 folds) together with C in log-scale (i.e., x-axis: $\log_2 C$, y-axis: accuracy). Provide also a table showing these numbers. What value of C gives the best validation accuracy?

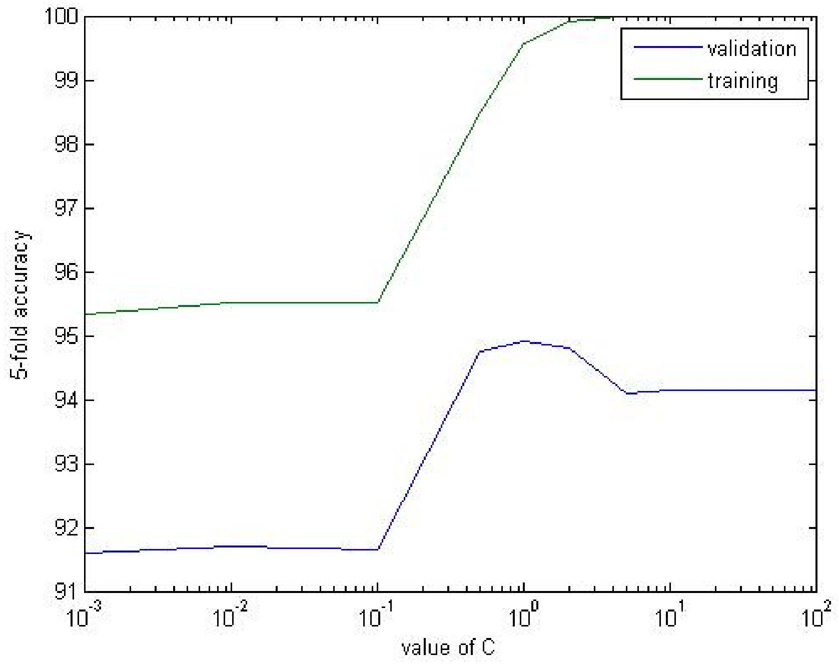


C	0.001	0.01	0.1	0.5	1	2	5	10	100
train acc.	95.28	98.34	99.96	100.00	100.00	100.00	100.00	100.00	100.00
val. acc.	92.10	94.30	93.50	92.15	92.15	92.10	92.10	92.10	92.10

Best cross-validation performance for $C = 0.01$.

b) Now, re-train your four binary SVMs on the complete training set using the optimal C from a). Report the resulting training and test accuracies.
Final train accuracy: 98.15
Final test accuracy: 94.90

c) A standard preprocessing technique often used in practice is length normalization. There, each feature vector $\mathbf{x}$ gets rescaled so that $\|\mathbf{x}\|_2 = 1$. Now, normalize all training and test instances and repeat part a) and b) for the new data. Provide the same plots and tables.



C	0.001	0.01	0.1	0.5	1	2	5	10	100
train acc.	95.33	95.51	95.53	98.48	99.56	99.91	100.00	100.00	100.00
val. acc.	91.60	91.70	91.65	94.75	94.90	94.80	94.10	94.15	94.15

Best cross-validation performance for $C = 1$.
Final train accuracy: 99.50
Final test accuracy: 95.80

d) Compare the test error from c) to previous test error from b) without normalization, and explain why normalization makes a difference (*Hint:* What happens to the training error of a single example?). Why did we have to reestimate the optimal value for C? Relate your answers to the SVM objective.
We obtain better accuracy by length-normalizing feature vectors first. The weight vector now has larger values because we rescaled features. This means that we need to also choose a bigger value for C to balance the $\frac{1}{2}\mathbf{w} \cdot \mathbf{w}$ term out. Normalization ensures that the slack errors of longer documents will not dominate the overall slack error.

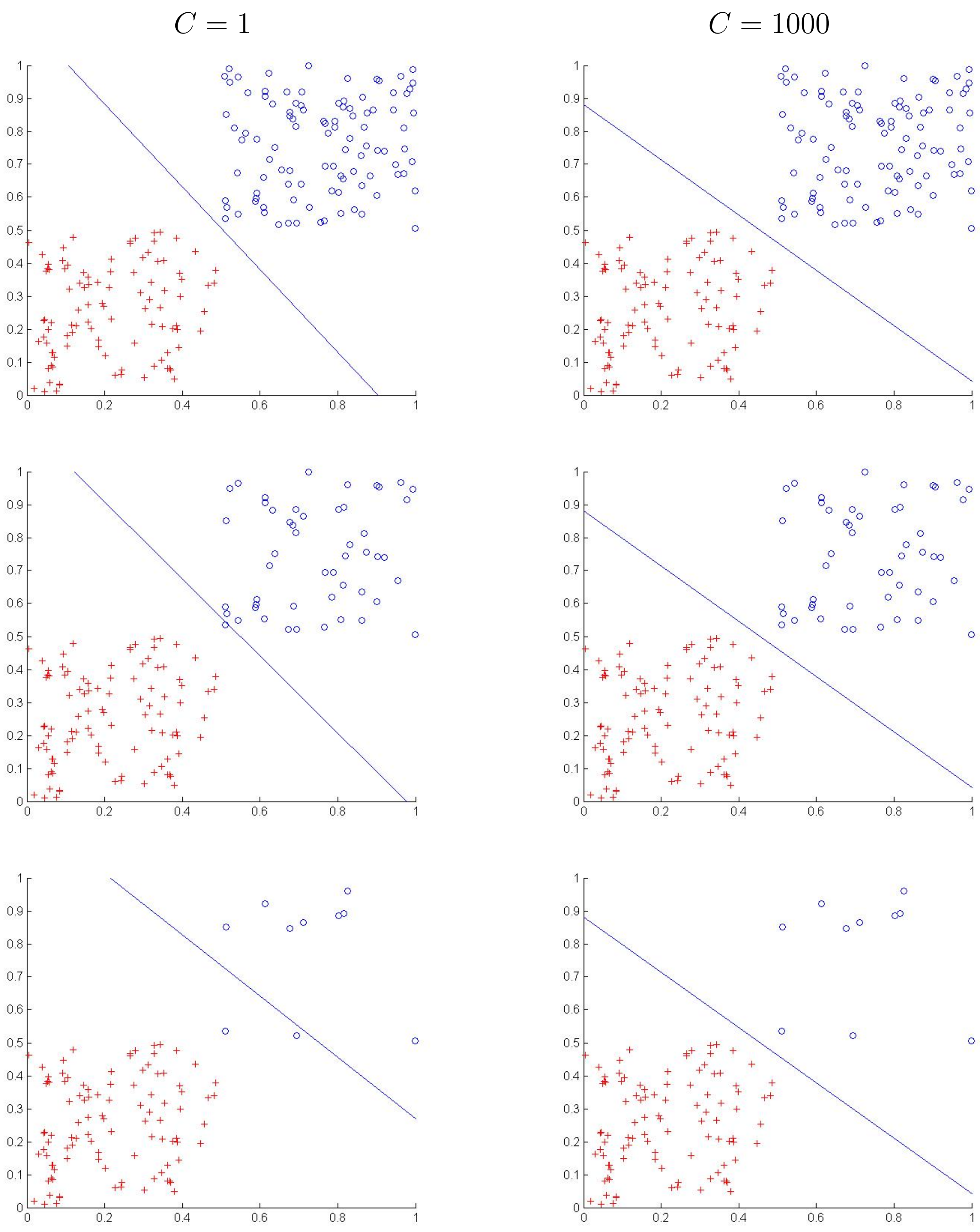
3 Skewed datasets

In practice, you will often have to deal with datasets that do not contain an equal number of positive and negative training examples. These *skewed* (or *unbalanced*) datasets can, for example, occur in clinical studies where patients with a rare form of cancer are the positive examples and patients with no abnormalities are the negative examples. In such situations, it is virtually impossible to obtain more positive examples. Another example is spam classification where users usually receive more ham than spam.

In this problem, you will investigate various problems with skewed datasets are and learn a way of mitigating them.

a) Suppose you know for a binary classification problem that $P(Y = 1) = 0.07$ in both the test and training set. Describe a simple rule that would give you at least 90% classification accuracy on the test set. What is the major problem with that rule?
By simply classifying all examples as negative, we achieve an overall accuracy of 93%. The obvious problem is, however, that all positive instances get misclassified.

b) You are given two synthetic data files *boxes.train* and *boxes.test*, for training and testing respectively. The training dataset contains 200 examples, 100 positive and 100 negative ones, the test dataset ten times as many samples. The positive examples stem from a uniform distribution $f(x, y)$ with $0 \leq x, y \leq 0.5$ and the negative ones from a uniform distribution $g(x, y)$ with $0.5 < x, y \leq 1.0$.
Let S_{nneg} denote the dataset containing all positive training examples but only the first *nneg* negative training examples (in the order they appear in the file). For example, S_{30} would contain 130 examples in total, 100 positive examples and the first 30 negative ones.
Train a soft-margin linear classifier with SVM^{light} on S_{nneg} for all combinations of $nneg \in \{100, 50, 10\}$ and $C \in \{1, 1000\}$. Make sure that the only parameter you set for `svm_learn` is C (option `-c`). For each combination, plot the resulting dataset and the solution of the SVM, i.e. the optimal hyperplane. Use plus signs and circles as markers for positive and negative instances respectively.



c) Look at the plots of part b). How does the location of the optimal hyperplane change for $C = 1$ as we decrease the number of negative instances? How does it change for $C = 1000$? Provide an explanation for both observations. Relate your answers to the primal optimization objective of soft-margin SVMs.
Hyperplane moves for C = 1 into direction of negative examples because margin errors from negative instances are outweighed by positive examples.
However, for C = 1000, the resulting hyperplane is almost equal to the one of a hard margin SVM (misclassified training examples cause extremely high penalties). The

location is thus mainly determined by the left-most datapoint.

d) One way of overcoming this issue is using custom weights in the optimization objective:

$$\min_{\mathbf{w},\mathbf{b},\xi} \quad \frac{1}{2}\mathbf{w}\cdot\mathbf{w} + C\cdot j\sum_{i:y_i=1}^{M}\xi_i + C\sum_{i:y_i=-1}^{M}\xi_i \tag{2}$$

$$\text{subject to} \quad \forall k : y_k(\mathbf{w}\cdot\mathbf{x}_k+\mathbf{b}) \geq 1-\xi_k \quad , \xi_k \geq 0 \tag{3}$$

Suppose we set

$$j = \frac{\#\text{ negative training examples}}{\#\text{ positive training examples}}.$$

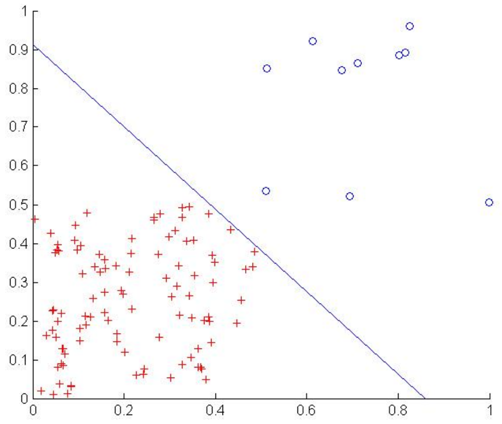
Assume that we always have at least one positive training example so that j is well-defined. Give a quantitative argument for why this choice is reasonable given no other information about the classes.

It weighs the margin errors of negative and positive instances equally.

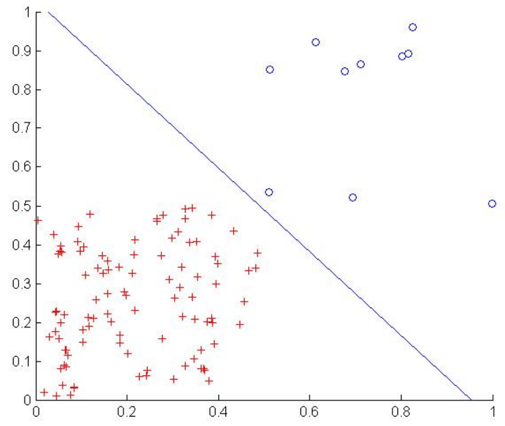
e) Run SVM^{light} on S_{10} using $C = 1$ and $j \in \{0.5, 0.1, 0.05\}$ and give plots (same as in b)) for each value of j.

How does the position of the hyperplane change when j gets decreased? Why does that happen? Relate your answer to the optimization objective in d).

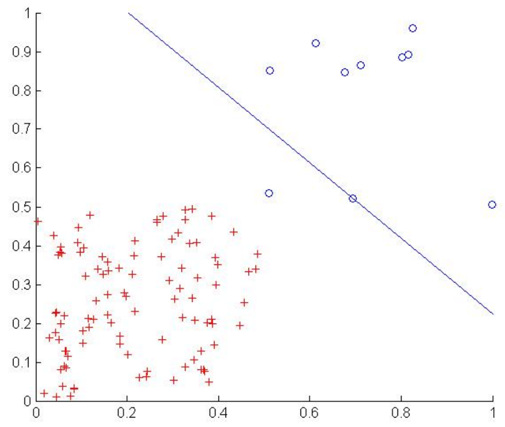
$j = 0.05$



$j = 0.1$



$j = 0.5$



As j decreases, errors on the negative class get weighted more strongly. This is why the hyperplane moves closer to the positive examples.

f) Now run each of the three models of part e) on *boxes.test* and report overall accuracy, as well as the number of false positives and false negatives for each model.
How does the number of false negatives / false positives behave when j gets decreased? Relate your findings to the plots in e).

	j = 0.05	**j = 0.1**	**j = 0.5**
false positives	0	0	87
false negatives	29	0	0
accuracy	98.55%	100.00%	95.65%

As j decreases, we make fewer errors on negative examples but start making errors on positive examples. This is very plausible given the positions of the hyperplanes in the plots.