

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.867 MACHINE LEARNING, FALL 2009

Problem Set 1

Due Date: Thursday, Sep 24 (at 5pm)

NOTE: the problem sets are not graded in the usual way. We have adopted a new grading scheme for problem sets, emphasizing the process rather than a grade. We will help you understand and work through the problems. You are therefore strongly encouraged to clear up any misunderstandings prior to the deadline. By submitting your solutions, you indicate that you do understand the problem and have worked through the solution yourself. Your submission will be recorded and you will receive no other grade from the submission.

Electronic submission is required! A submission form will be made available at the course website.

Please see the course website for help on finding and using Matlab. Note that you will need the Optimization Toolbox, which is included in the MIT distribution.

Problem 1 Perceptron

We can think of running the Perceptron algorithm along an infinite sequence of examples $\underline{x}_1, \underline{x}_2, \dots$. In response to each input example \underline{x}_t , we must predict a ± 1 label. After making the prediction, we receive the corresponding label y_t and will be able to adjust the classifier parameters in case we made a mistake. Clearly, if we make no assumptions about the sequence of examples and the associated labels, the Perceptron algorithm can behave arbitrarily badly. In fact, it could make a mistake on each example. However, if there exists a good classifier for the sequence, one that we could determine with hindsight, shouldn't the perceptron algorithm also perform well? Indeed, if we assume $\|\underline{x}_t\| \leq R$ and that there exists a linear reference classifier that correctly classifies all the examples in the sequence with a geometric margin γ_g , then the perceptron algorithm is guaranteed to make only a few mistakes. The algorithm will make at most R^2/γ_g^2 mistakes along the infinite sequence!

(a) Perhaps the mistake guarantee holds also for different variations of the perceptron algorithm. Show that we obtain the same guarantee if the updates in response to mistakes (here at \underline{x}_t) are replaced by

$$\underline{\theta} \leftarrow \underline{\theta} + y_t \frac{\underline{x}_t}{\|\underline{x}_t\|} \quad (1)$$

(b) Consider linear classifiers through origin. If we are given only a single example (\underline{x}, y) , show that the perceptron update, starting with $\underline{\theta} = 0$, gives the maximum margin separator for that example.

(c) Separation with a geometric margin γ_g is a strong assumption about the sequence of examples and labels. For example, suppose $\|\underline{x}_t\| \leq R$ for all t , and let $\gamma_g \rightarrow R$. What are the possible sequences in this case? Using part b), can you say that the perceptron would indeed make only $R^2/\gamma_g^2 = 1$ mistake for such sequences?

(d) One of the remarkable properties of the perceptron guarantee is that it does not depend on the dimension of the examples $\dim(\underline{x}) = d$ or, equivalently, the number of parameters in the linear classifier. Perhaps the effect of the dimension is nevertheless lurking there somewhere? Consider the following sequence of coordinate vectors

$$\underline{x}_1 = \begin{bmatrix} 1 \\ 0 \\ \dots \end{bmatrix}, \quad \underline{x}_2 = \begin{bmatrix} 0 \\ 1 \\ \dots \end{bmatrix}, \dots \quad (2)$$

such that the first n examples lie in n dimensional subspace and $\|\underline{x}_t\| = 1$ for all t . If we call $y_t(\underline{\theta} \cdot \underline{x}_t) \leq 0$ a mistake, how many mistakes will the perceptron algorithm make on the first n examples with any labels? What can you say about the geometric margin γ_g in this case?

(e) Let's consider the Perceptron algorithm in a bit more practical setting. You can download the MATLAB scripts and the data for this problem from the course website. Unpack `hw1.zip`. `data` are labeled examples (use the MATLAB command “load data”). `data.X` is a matrix where each row corresponds to an input vector and `data.y` is a column vector of the corresponding labels.

Implement the Perceptron algorithm that finds a hyperplane through the origin (no offset). We provided a function prototype in the file “`perceptron_build.m`” which should be called as `perc = perceptron_build(data)`. It should go through the points *once* in the order they are given in the dataset and record the number of times we make a mistake on each data point. These are to be returned in a column vector `perc.mistakes`.

Randomize the order of the points in the dataset. This can be done, e.g., as follows `[tmp,ind] = sort(rand(size(data.X,1),1)); data.X = data.X(ind,:); data.y = data.y(ind);` How does the number of mistakes depend on the order? Why?

Can you use a few runs of the perceptron algorithm to estimate the value of the geometric margin γ_g (if it exists)? How?

Problem 2 Support vector machines

In many cases the data points to be classified are not linearly separable. Even if they are, however, the maximum margin separator may be unduly influenced by a single or only a few data points. It would be great if we could just “give up” on a few points and construct the classifier on the basis of the remaining points. This would be a hard problem to solve but we can do something close to it by introducing slack variables for the margin constraints in support vector machines. Here we will try to understand the effect of the slack variables on the SVM solution and how the penalty constant C should be chosen. The SVM optimization problem is given by

$$\text{minimize } \frac{1}{2}\|\underline{\theta}\|^2 + C \sum_{i=1}^n \xi_i \quad \text{subject to} \quad (3)$$

$$y_i(\underline{\theta} \cdot \underline{x}_i + \theta_0) \geq 1 - \xi_i, \quad i = 1, \dots, n \quad (4)$$

$$\xi_i \geq 0, \quad i = 1, \dots, n \quad (5)$$

$$(6)$$

We have provided you with MATLAB code to experiment with different slack penalties. You can get the SVM solution by running `svm = svm_build(data,@K_linear,C)` where C is the slack penalty and `@K_linear` passes a function reference to a “linear kernel” (see file “`k_linear.m`”). The data is the same as before. You can plot the points and the SVM solution by running `svm_plot(data,svm)` which will highlight the support vectors as well as the margin around the linear separator.

After solving the SVM, the variable “`svm.NS`” contains the number of support vectors in the resulting solution. “`svm.w0`” contains the offset (bias) of the hyperplane, “`svm.XS`” contains the support vectors (1 per row), and “`svm.beta`” for row i gives the dual parameters for the quadratic programming problem (equal to $\alpha_i y_i$). See “`svm_discrim_func.m`” for an example of how they are used.

(a) Try $C = 100$ and $C = 1000$. Why doesn’t the solution change? Explain this in terms of the optimization problem.

(b) Recall that, for the hard-margin SVM, the geometric margin is defined as the minimum distance from any point to the separating hyperplane. This distance can be shown to be $\frac{1}{\|\underline{\theta}\|}$, where the 1 in the numerator comes from the classification constraints $y_t(\underline{\theta} \cdot \underline{x}_t + \theta_0) \geq 1$ in the optimization problem.

When we use slack variables (regardless of whether the data is separable), some of the points may be misclassified, so we can no longer measure the margin directly in terms of the minimum distance to the separating hyperplane. However, we can still use the expression $\frac{1}{\|\underline{\theta}\|}$ as the definition of the margin, where $\underline{\theta}$ comes from solving the SVM optimization problem with slack variables.

Try $C = 0.01, 0.1, 1, 10, 100$. What happens to the margin $1/\|\underline{\theta}\|$ as C increases? Will this always happen as we increase C ? What happens to the number of support vectors as C increases?

(c) The value of C will typically change the resulting classifier and therefore also affects the accuracy on test examples. The best choice of C would be to optimize the test performance. With access only to the few training examples, we cannot hope to do that. However, we can simulate test performance via leave-one-out cross-validation. In other words, we can hold out each training example-label pair in turn, training the classifier on the remaining examples, and testing the classifier on the held out “test” example. Implement the cross-validation procedure. Do choices $C = 0.01, 0.1, 1, 10, 100$ have any significant effect on the cross-validation performance? Could we instead use a much simpler procedure by training the classifier only once for each value of C and relying on the bound given by the number of support vectors? Explain why or why not.

(d) If we fix C and obtain $\hat{\underline{\theta}}$, $\hat{\theta}_0$, and $\hat{\xi}_i$ as the solution to the quadratic programming problem. The slack is relevant (non-zero) only for support vectors. How does the value of slack $\hat{\xi}_i$ relate to the distance of a support vector \underline{x}_i from the decision boundary?

(e) The quadratic programming problem involves both $\underline{\theta}, \theta_0$ and the slack variables ξ_i . We can rewrite the optimization problem in terms of $\underline{\theta}, \theta_0$ alone. By doing so we explicate what the loss-function is on the training points corresponding to the relaxed max-margin formulation. You can do this as follows. First, fix $\underline{\theta}, \theta_0$ and find the optimal values of the slack variables $\xi_i = \xi_i(\underline{\theta}, \theta_0)$ as functions of $\underline{\theta}, \theta_0$. What are these values? Are all the margin constraints satisfied with these values for the slack variables?

The resulting minimizing problem over $\underline{\theta}, \theta_0$ can be formally written as

$$\frac{1}{2}\|\underline{\theta}\|^2 + C \sum_{i=1}^n \xi_i(\underline{\theta}, \theta_0) \tag{7}$$

where the first (regularization) term biases our solution towards zero in the absence of any data and the remaining terms give rise to the loss functions. What are the loss functions? Do we need any additional constraints? Many learning criteria can be understood and compared in the above regularization + loss form.