

Perceptron, convergence, and generalization

Recall that we are dealing with linear classifiers through origin, i.e.,

$$f(\underline{x}; \underline{\theta}) = \text{sign}(\underline{\theta}^T \underline{x}) \quad (1)$$

where $\underline{\theta} \in \mathcal{R}^d$ specifies the parameters that we have to estimate on the basis of training examples (images) $\underline{x}_1, \dots, \underline{x}_n$ and labels y_1, \dots, y_n .

We will use the perceptron algorithm to solve the estimation task. Let k denote the number of parameter updates we have performed and $\underline{\theta}^{(k)}$ the parameter vector after k updates. Initially $k = 0$ and $\underline{\theta}^{(k)} = \underline{0}$. The algorithm then cycles through all the training instances (\underline{x}_t, y_t) and updates the parameters only in response to mistakes, i.e., when the label is predicted incorrectly. More precisely, we set $\underline{\theta}^{(k+1)} = \underline{\theta}^{(k)} + y_t \underline{x}_t$ when $y_t (\underline{\theta}^{(k)})^T \underline{x}_t \leq 0$ (mistake), and otherwise leave the parameters unchanged.

Convergence in a finite number of updates

Let's now show that the perceptron algorithm indeed converges in a finite number of updates. The same analysis will also help us understand how the linear classifier generalizes to unseen images. To this end, we will assume that all the (training) images have bounded Euclidean norms, i.e., $\|\underline{x}_t\| \leq R$ for all t and some finite R . This is clearly the case for any pixel images with bounded intensity values. We also make a much stronger assumption that there exists a linear classifier in our class with finite parameter values that correctly classifies all the (training) images. More precisely, we assume that there is some $\gamma > 0$ such that $y_t (\underline{\theta}^*)^T \underline{x}_t \geq \gamma$ for all $t = 1, \dots, n$. The additional number $\gamma > 0$ is used to ensure that each example is classified correctly with a *finite margin*.

The convergence proof is based on combining two results: 1) we will show that the inner product $(\underline{\theta}^*)^T \underline{\theta}^{(k)}$ increases at least linearly with each update, and 2) the squared norm $\|\underline{\theta}^{(k)}\|^2$ increases at most linearly in the number of updates k . By combining the two we can show that the cosine of the angle between $\underline{\theta}^{(k)}$ and $\underline{\theta}^*$ has to increase by a finite increment due to each update. Since cosine is bounded by one, it follows that we can only make a finite number of updates.

Part 1: we simply take the inner product $(\underline{\theta}^*)^T \underline{\theta}^{(k)}$ before and after each update. When making the k^{th} update, say due to a mistake on image \underline{x}_t , we get

$$(\underline{\theta}^*)^T \underline{\theta}^{(k)} = (\underline{\theta}^*)^T \underline{\theta}^{(k-1)} + y_t (\underline{\theta}^*)^T \underline{x}_t \geq (\underline{\theta}^*)^T \underline{\theta}^{(k-1)} + \gamma \quad (2)$$

since, by assumption, $y_t(\underline{\theta}^*)^T \underline{x}_t \geq \gamma$ for all t ($\underline{\theta}^*$ is always correct). Thus, after k updates,

$$(\underline{\theta}^*)^T \underline{\theta}^{(k)} \geq k\gamma \quad (3)$$

Part 2: Our second claim follows simply from the fact that updates are made only on mistakes:

$$\|\underline{\theta}^{(k)}\|^2 = \|\underline{\theta}^{(k-1)} + y_t \underline{x}_t\|^2 \quad (4)$$

$$= \|\underline{\theta}^{(k-1)}\|^2 + 2y_t(\underline{\theta}^{(k-1)})^T \underline{x}_t + \|\underline{x}_t\|^2 \quad (5)$$

$$\leq \|\underline{\theta}^{(k-1)}\|^2 + \|\underline{x}_t\|^2 \quad (6)$$

$$\leq \|\underline{\theta}^{(k-1)}\|^2 + R^2 \quad (7)$$

since $y_t(\underline{\theta}^{(k-1)})^T \underline{x}_t \leq 0$ whenever an update is made and, by assumption, $\|\underline{x}_t\| \leq R$. Thus,

$$\|\underline{\theta}^{(k)}\|^2 \leq kR^2 \quad (8)$$

We can now combine parts 1) and 2) to bound the cosine of the angle between $\underline{\theta}^*$ and $\underline{\theta}^{(k)}$:

$$\cos(\underline{\theta}^*, \underline{\theta}^{(k)}) = \frac{(\underline{\theta}^*)^T \underline{\theta}^{(k)}}{\|\underline{\theta}^{(k)}\| \|\underline{\theta}^*\|} \stackrel{1)}{\geq} \frac{k\gamma}{\|\underline{\theta}^{(k)}\| \|\underline{\theta}^*\|} \stackrel{2)}{\geq} \frac{k\gamma}{\sqrt{kR^2} \|\underline{\theta}^*\|} \quad (9)$$

Since cosine is bounded by one, we get

$$1 \geq \frac{k\gamma}{\sqrt{kR^2} \|\underline{\theta}^*\|} \quad \text{or} \quad k \leq \frac{R^2 \|\underline{\theta}^*\|^2}{\gamma^2} \quad (10)$$

Margin and geometry

It is worthwhile to understand this result a bit further. For example, does $\|\underline{\theta}^*\|^2/\gamma^2$ relate to how difficult the classification problem is? Indeed, it does. We claim that its inverse, i.e., $\gamma/\|\underline{\theta}^*\|$ is the smallest distance in the image space from any example (image) to the decision boundary specified by $\underline{\theta}^*$. In other words, it serves as a measure of how well the two classes of images are separated (by a linear boundary). We will call this the geometric margin or γ_{geom} (see figure 1). γ_{geom}^{-1} is then a fair measure of how difficult the problem is: the smaller the geometric margin that separates the training images, the more difficult the problem.

To calculate γ_{geom} we measure the distance from the decision boundary $\underline{\theta}^{*T} \underline{x} = 0$ to one of the images \underline{x}_t for which $y_t \underline{\theta}^{*T} \underline{x}_t = \gamma$. Since $\underline{\theta}^*$ specifies the normal to the decision boundary,

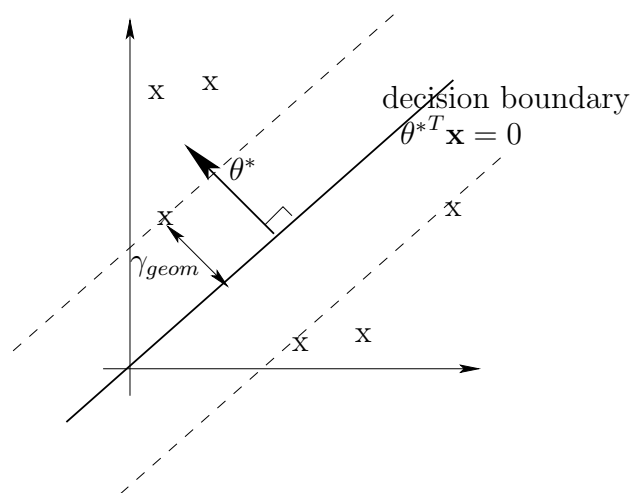


Figure 1: Geometric margin

the shortest path from the boundary to the image \underline{x}_t will be parallel to the normal. The image for which $y_t \underline{\theta}^{*T} \underline{x}_t = \gamma$ is therefore among those closest to the boundary. Now, let's define a line segment from $\underline{x}(0) = \underline{x}_t$, parallel to $\underline{\theta}^*$, towards the boundary. This is given by

$$\underline{x}(\xi) = \underline{x}(0) - \xi \frac{y_t \underline{\theta}^*}{\|\underline{\theta}^*\|} \quad (11)$$

where ξ defines the length of the line segment since it multiplies a unit length vector. It remains to find the value of ξ such that $\underline{\theta}^{*T} \underline{x}(\xi) = 0$, or, equivalently, $y_t \underline{\theta}^{*T} \underline{x}(\xi) = 0$. This is the point where the segment hits the decision boundary. Thus

$$y_t \underline{\theta}^{*T} \underline{x}(\xi) = y_t \underline{\theta}^{*T} \left[\underline{x}(0) - \xi \frac{y_t \underline{\theta}^*}{\|\underline{\theta}^*\|} \right] \quad (12)$$

$$= y_t \underline{\theta}^{*T} \left[\underline{x}_t - \xi \frac{y_t \underline{\theta}^*}{\|\underline{\theta}^*\|} \right] \quad (13)$$

$$= y_t \underline{\theta}^{*T} \underline{x}_t - \xi \frac{\|\underline{\theta}^*\|^2}{\|\underline{\theta}^*\|} \quad (14)$$

$$= \gamma - \xi \|\underline{\theta}^*\| = 0 \quad (15)$$

implying that the distance is exactly $\xi = \gamma / \|\underline{\theta}^*\|$ as claimed. As a result, the bound on the number of perceptron updates can be written more succinctly in terms of the geometric

margin γ_{geom} (distance to the boundary):

$$k \leq \left(\frac{R}{\gamma_{geom}} \right)^2 \quad (16)$$

with the understanding that γ_{geom} is the largest geometric margin that could be achieved by a linear classifier for this problem. Note that the result does not depend (directly) on the dimension d of the examples, nor the number of training examples n . It is nevertheless tempting to interpret $\left(\frac{R}{\gamma_{geom}} \right)^2$ as a measure of difficulty (or complexity) of the problem of learning linear classifiers in this setting. You will see later in the course that this is exactly the case, cast in terms of a measure known as *VC-dimension*.

Generalization guarantees

We have so far discussed the perceptron algorithm only in relation to the training set but we are more interested in how well the perceptron classifies images we have not yet seen, i.e., how well it generalizes to new images. Our simple analysis above actually provides some information about generalization. Let's assume then that all the images and labels we could possibly encounter satisfy the same two assumptions. In other words, 1) $\|\underline{x}_t\| \leq R$ and 2) $y_t \underline{\theta}^{*T} \underline{x}_t \geq \gamma$ for all t and some finite $\underline{\theta}^*$. So, in essence, we assume that there is a linear classifier that works for all images and labels in this problem, we just don't know what this linear classifier is to start with. Let's now imagine getting the images and labels one by one and performing only a single update per image, if misclassified, and move on. The previous situation concerning the training set corresponds to encountering the same set of images repeatedly. How many mistakes are we now going to make in this infinite arbitrary sequence of images and labels, subject only to the two assumptions? The same number $k \leq (R/\gamma_{geom})^2$. Once we have made this many mistakes we would classify all the new images correctly. So, provided that the two assumptions hold, especially the second one, we obtain a nice guarantee of generalization. One caveat here is that the perceptron algorithm does need to know when it has made a mistake. The bound is after all cast in terms of the number of updates based on mistakes.

Maximum margin classifier – support vector machine

We have so far used a simple on-line algorithm, the perceptron algorithm, to estimate a linear classifier. Our reference assumption has been, however, that there exists a linear classifier that has a large geometric margin, i.e., whose decision boundary is well separated

from all the training images (examples). Can't we find such a large margin classifier directly? Yes, we can. The classifier is known as the Support Vector Machine or SVM for short.

You could imagine finding the maximum margin linear classifier by first identifying any classifier that correctly classifies all the examples (Figure 3a) and then increasing the geometric margin until the classifier "locks in place" at the point when we cannot increase the margin any further (Figure 3b).

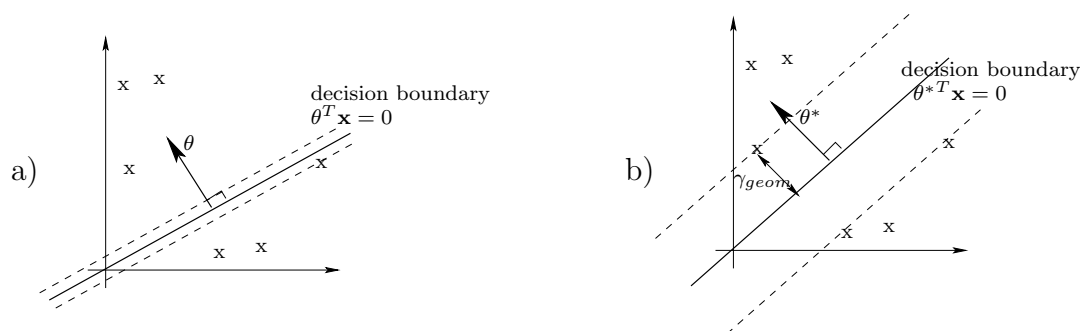


Figure 2: a) A linear classifier (through origin) with a small geometric margin, b) maximum margin linear classifier.

More formally, we can set up an optimization problem for directly maximizing the geometric margin. We will need the classifier to be correct on all the training examples or $y_t \underline{\theta}^T \underline{x}_t \geq \gamma$ for all $t = 1, \dots, n$. Subject to these constraints, we would like to maximize $\gamma / \|\underline{\theta}\|$, i.e., the geometric margin. We can alternatively minimize the inverse $\|\underline{\theta}\| / \gamma$ or the inverse squared $(\|\underline{\theta}\| / \gamma)^2$ subject to the same constraints. We then have the following optimization problem for finding $\underline{\theta}^*$:

$$\text{minimize } \frac{1}{2} \|\underline{\theta}\|^2 / \gamma^2 \quad \text{subject to } y_t \underline{\theta}^T \underline{x}_t \geq \gamma \quad \text{for all } t = 1, \dots, n \quad (17)$$

where the additional factor of 1/2 is inconsequential but convenient later on. We can simplify this objective a bit further by getting rid of γ . By rewriting the objective slightly, combining $\underline{\theta}$ and γ , we see that the training examples only constrain the ratio $\underline{\theta} / \gamma$:

$$\text{minimize } \frac{1}{2} \|\underline{\theta} / \gamma\|^2 \quad \text{subject to } y_t (\underline{\theta} / \gamma)^T \underline{x}_t \geq 1 \quad \text{for all } t = 1, \dots, n \quad (18)$$

If we set $\gamma = 0.5$ or $\gamma = 1$, we would obtain $\underline{\theta}$ that is scaled correspondingly so as to maintain the ratio $\underline{\theta} / \gamma$. For simplicity, and without any loss of generality, we can set $\gamma = 1$.

The resulting optimization problem for finding the maximum margin decision boundary through origin is

$$\text{minimize } \frac{1}{2} \|\underline{\theta}\|^2 \quad \text{subject to } y_t \underline{\theta}^T \underline{x}_t \geq 1 \quad \text{for all } t = 1, \dots, n \quad (19)$$

This optimization problem is a *quadratic programming* problem (objective is quadratic in the parameters with linear constraints).

General formulation, offset parameter

We will modify the linear classifier here slightly by adding an offset term so that the decision boundary does not have to go through the origin. In other words, the classifier that we consider has the form

$$f(\underline{x}; \underline{\theta}, \theta_0) = \text{sign}(\underline{\theta}^T \underline{x} + \theta_0) \quad (20)$$

with parameters $\underline{\theta}$ (normal to the separating hyper-plane) and the offset parameter θ_0 , a real number. As before, the equation for the separating hyper-plane is obtained by setting the argument to the sign function to zero or $\underline{\theta}^T \underline{x} + \theta_0 = 0$. This is a general equation for a hyper-plane (line in 2-dim). The additional offset parameter can lead to a classifier with a larger geometric margin. This is illustrated in Figures 3a and 3b. Note that the vector $\hat{\theta}$ corresponding to the maximum margin solution is different in the two figures.

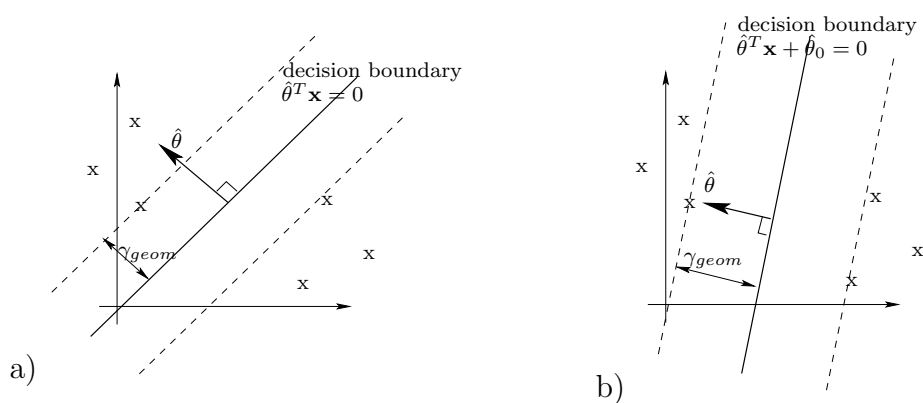


Figure 3: a) Maximum margin linear classifier through origin; b) Maximum margin linear classifier with an offset parameter

The offset parameter changes the optimization problem only slightly:

$$\text{minimize } \frac{1}{2} \|\underline{\theta}\|^2 \quad \text{subject to } y_t(\underline{\theta}^T \underline{x}_t + \theta_0) \geq 1 \quad \text{for all } t = 1, \dots, n \quad (21)$$

Note that the offset parameter only appears in the constraints. This is different from simply modifying the linear classifier through origin by feeding it with examples that have an additional constant component, i.e., $\underline{x}' = [\underline{x}; 1]$. In the above formulation we do not bias in any way where the separating hyper-plane should appear, only that it should maximize the geometric margin.

Properties of the maximum margin linear classifier

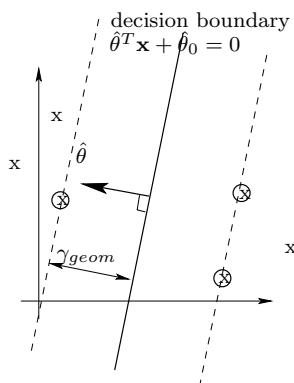


Figure 4: Support vectors (circled) associated the maximum margin linear classifier

The maximum margin classifier has several very nice properties, and some not so advantageous features.

Benefits. On the positive side, we have already motivated these classifiers based on the perceptron algorithm as the “best reference classifiers”. The solution is also unique based on any linearly separable training set. Moreover, drawing the separating boundary as far from the training examples as possible makes it robust to noisy examples (though not noisy labels). The maximum margin linear boundary also has the curious property that the solution depends only on a subset of the examples, those that appear exactly on the margin (the dashed lines parallel to the boundary in the figures). The examples that lie exactly on the margin are called *support vectors* (see Figure 4). The rest of the examples could lie anywhere outside the margin without affecting the solution. We would therefore

get the same classifier if we had only received the support vectors as training examples. Is this a good thing? To answer this question we need a bit more formal (and fair) way of measuring how good a classifier is.

One possible “fair” performance measure evaluated only on the basis of the training examples is *cross-validation*. This is simply a method of retraining the classifier with subsets of training examples and testing it on the remaining held-out (and therefore fair) examples, pretending we had not seen them before. A particular version of this type of procedure is called *leave-one-out cross-validation*. As the name suggests, the procedure is defined as follows: select each training example in turn as the single example to be held-out, train the classifier on the basis of all the remaining training examples, test the resulting classifier on the held-out example, and count the errors. More precisely, let the superscript ‘ $-i$ ’ denote the parameters we would obtain by finding the maximum margin linear separator without the i^{th} training example. Then

$$\text{leave-one-out CV error} = \frac{1}{n} \sum_{i=1}^n \text{Loss} \left(y_i, f(\underline{x}_i; \hat{\underline{\theta}}^{-i}, \hat{\theta}_0^{-i}) \right) \quad (22)$$

where $\text{Loss}(y, y')$ is the zero-one loss. We are effectively trying to gauge how well the classifier would generalize to each training example if it had not been part of the training set. A classifier that has a low leave-one-out cross-validation error is likely to generalize well though it is not guaranteed to do so.

Now, what is the leave-one-out CV error of the maximum margin linear classifier? Well, examples that lie outside the margin would be classified correctly regardless of whether they are part of the training set. Not so for support vectors. They are key to defining the linear separator and thus, if removed from the training set, may be misclassified as a result. We can therefore derive a simple upper bound on the leave-one-out CV error:

$$\text{leave-one-out CV error} \leq \frac{\# \text{ of support vectors}}{n} \quad (23)$$

A small number of support vectors – a sparse solution – is therefore advantageous. This is another argument in favor of the maximum margin linear separator.

Problems. There are problems as well, however. Even a single training example, if mislabeled, can radically change the maximum margin linear classifier. Consider, for example, what would happen if we switched the label of the top right support vector in Figure 4.

Allowing misclassified examples, relaxation

Labeling errors are common in many practical problems and we should try to mitigate their effect. We typically do not know whether examples are difficult to classify because of labeling errors or because they simply are not linearly separable (there isn't a linear classifier that can classify them correctly). In either case we have to articulate a trade-off between misclassifying a training example and the potential benefit for other examples.

Perhaps the simplest way to permit errors in the maximum margin linear classifier is to introduce "slack" variables for the classification/margin constraints in the optimization problem. In other words, we measure the degree to which each margin constraint is violated and associate a cost for the violation. The costs of violating constraints are minimized together with the norm of the parameter vector. This gives rise to a simple relaxed optimization problem:

$$\text{minimize } \frac{1}{2} \|\underline{\theta}\|^2 + C \sum_{t=1}^n \xi_t \quad (24)$$

$$\text{subject to } y_t(\underline{\theta}^T \underline{x}_t + \theta_0) \geq 1 - \xi_t \text{ and } \xi_t \geq 0 \text{ for all } t = 1, \dots, n \quad (25)$$

where ξ_t are the slack variables. The margin constraint is violated when we have to set $\xi_t > 0$ for some example. The penalty for this violation is $C\xi_t$ and it is traded-off with the possible gain in minimizing the squared norm of the parameter vector or $\|\underline{\theta}\|^2$. If we increase the penalty C for margin violations then at some point all $\xi_t = 0$ and we get back the maximum margin linear separator (if possible). On the other hand, for small C many margin constraints can be violated. Note that the relaxed optimization problem specifies a particular *quantitative* trade-off between the norm of the parameter vector and margin violations. It is reasonable to ask whether this is indeed the trade-off we want.

Let's try to understand the setup a little further. For example, what is the resulting margin when some of the constraints are violated? We can still take $1/\|\hat{\underline{\theta}}\|$ as the geometric margin. This is indeed the geometric margin based on examples for which $\hat{\xi}_t = 0$. So, is it the case that we get the maximum margin linear classifier for the subset of examples for which $\xi_t^* = 0$? No, we don't. The examples that violate the margin constraints, including those training examples that are actually misclassified (larger violation), do affect the solution. In other words, the parameter vector $\hat{\underline{\theta}}$ is defined on the basis of examples right at the margin ($\hat{\xi}_t = 0$ but the constraint is "active"), those that violate the constraint but not enough to be misclassified (so that $\hat{\xi}_t \in (0, 1)$), as well as those that are misclassified ($\hat{\xi}_t \geq 1$). All of these are support vectors in this sense.