

6.036 Spring 2018 : Week 2

February 25, 2018

3 On linear classification and generalization

We have so far been operating under the assumption that if we find a linear classifier with few (or no) training errors, then this classifier will also make few mistakes on the test set that we don't yet have access to. In our discussion, we have introduced two different ways to think about why this might be so. The first line of reasoning pertains to the perceptron algorithm itself. The convergence proof relies on the assumption that there exists some (unknown to us) linear classifier that separates the training examples with a large margin. If this is indeed so, then the algorithm convergence quickly to one of the many possible separating solutions. Moreover, if we extend this large-margin assumption to cover both training *and* test examples, then the perceptron algorithm is guaranteed to make a small number of test errors as well. This is so since there's an overall bound $(R/\gamma)^2$ on the number of errors that the algorithm can make over *both* training and test examples. We can then run the algorithm over the training examples, counting the number of errors, until we find a separable solution. The remaining errors, i.e., $(R/\gamma)^2$ less training errors, then bounds the number of errors the algorithm can make on test examples. There's one subtlety about this argument. Each test prediction we make should accompany immediate feedback about whether the prediction was correct prior to proceeding to the next example so that we can apply the mistake update. In summary, it is important to note that this large-margin assumption is *about the problem*, that it is really easy to solve. So, loosely speaking, all that we are saying is that the perceptron algorithm generalizes well on easy problems.

The second rationale is that the set of linear classifiers is already constrained enough such that, regardless of how hard the task is, the fraction of training errors reflects the fraction of test errors. This rationale depends somewhat on the dimension of the feature vectors. Let's see how it breaks down (we will come back to it more formally later). Suppose we are classifying documents using bag-of-words feature vectors. In other words, each document is mapped to a vector where each coordinate corresponds to a count of the number of times a word appears in the document. The dimension could therefore easily be in the tens of thousands! Why is this a problem then? Well, if each document were just a single distinct word, i.e., each document is tied to a single coordinate x_j, then we could just set the sign of the associated parameter θ_j in the linear classifier to correctly classify all such documents. High dimensional feature vectors give linear classifiers a lot of power. Indeed, we will make them shortly infinitely powerful by expanding the feature vectors to be infinite dimensional! Clearly we should work a little harder to ensure that such classifiers still generalize well.

3.1 Objective functions for learning

- Large margin classification as an optimization problem
- Hinge loss, margin boundaries, geometric view
- Support vector machine
- Stochastic gradient descent and the Pegasos algorithm

One way to frame machine learning problems is to write them down as optimization problems. The solution to the optimization problem is then the classifier we want. Thinking about problems in this way is often beneficial since the optimization problem expresses the key trade-offs we want an ideal classifier to balance. For example, we might wish to find a classifier that separates most if not all of the training points with a large margin. The key trade-off in this case is between how well we can classify each point (expressed through a loss function) and the value of the margin we attain (what we call a regularizer). The larger the margin we require the classifier to attain, the fewer points will actually satisfy it. The trade-off between the loss measured on training points and the margin we can achieve is therefore real and needs to be balanced somehow.

We will make these ideas progressively more precise, starting with a higher level view. In general, if we consider classifiers specified by parameters θ, θ_0, e.g., the set of linear classifiers, we first try to turn the learning problem into an optimization problem over these parameters. We say that we want the classifier, i.e., parameters θ, θ_0 that minimize some objective function that may be composed of two (or more) parts that characterize the key trade-offs. For example, we typically try to minimize

$$J(\theta, \theta_0) = \overbrace{\frac{1}{n} \sum_{i=1}^{n} \text{Loss}(x^{(i)}, y^{(i)}, \theta, \theta_0)}^{\text{average loss on training examples}} + \lambda \overbrace{R(\theta, \theta_0)}^{\text{regularizer}} \tag{1}$$

with respect to θ, θ_0 with the idea that it is the minimizing parameter values we are really after rather than the value of the objective. The objective function here has two key parts: 1) how the classifier performs on the training examples or average loss, and 2) the default answer that we wish to bias the classifier towards in the absence of any evidence from the training examples, expressed via the regularizer. $\lambda > 0$ here is known as the regularization parameter and it is used to quantify how we wish to balance these two competing goals. A large value of λ will emphasize the regularizer and thus steers the classifier towards that default answer. A small value of λ, on the other hand, discounts the regularizer in favor of the losses, resulting in θ, θ_0 that attain small training losses. The key motivation for introducing the concept of a loss function is that not all errors should be treated the same. Points near the decision boundary are inherently uncertain and should be penalized less than those that are overtly placed on the wrong side of the boundary. Formally, we need a loss function that simply specifies numerically how badly we classify each example, and balance such loss values against the regularizer.

3.2 Perceptron and optimization

The perceptron algorithm does not fit well within the optimization framework and therefore doesn't quite have a loss function. However, loosely speaking, we can think of the perceptron algorithm minimizing so-called zero-one loss (just the error), i.e.,

$$\text{Loss}_{0,1}(y(\theta \cdot x + \theta_0)x) = [\![\, y(\theta \cdot x + \theta_0) \leq 0 \,]\!] \tag{2}$$

Note that while this loss function is a function of the "agreement" $y(\theta \cdot x + \theta_0)$ it only cares about whether the agreement is positive (correct classification) or not (mistake). It doesn't depend on the magnitude of the agreement (or lack thereof). Perceptron algorithm also doesn't really have any regularizer or default answer. It starts with the all zero parameter values which we can think, again loosely, as the default. But the perceptron algorithm can get very far from this initial setting as it continues to iterate through the training examples. So there's really no well-defined $R(\theta, \theta_0)$ for the perceptron algorithm either.

3.3 Preface: large margin linear classification as optimization

Our goal is here to derive an objective function for learning linear classifiers by starting with the idea that we want the classifier to separate the points with a large margin. Recall that we can evaluate the margin for each training example relative to a linear classifier θ, θ_0 by

$$\gamma_i(\theta, \theta_0) = \frac{y^{(i)}(\theta \cdot x^{(i)} + \theta_0)}{\|\theta\|} \tag{3}$$

The margin is positive when the training example lies on the correct side of the decision boundary, and negative otherwise. The absolute value of the margin is always the distance of the training point from the decision boundary. We want a classifier θ, θ_0 that achieves a large margin for most or all training examples.

To translate the large margin goal into an objective function, we will introduce an additional parameter $\gamma_{ref} > 0$ into the optimization problem. γ_{ref} specifies the margin we would wish to attain for all examples. Of course, we don't know a priori what the reasonable value for γ_{ref} would be but we do want it to be large. The reason for introducing this reference margin is that we have a good idea of how it should be regularized. We wish to maximize γ_{ref} or, equivalently, minimize $1/\gamma_{ref}$, or $1/\gamma_{ref}^2$ (cf. perceptron convergence guarantee in terms of simple problems; we assume by default that the problem is simple). So we can just set our regularizer to be $1/\gamma_{ref}^2$) so that it favors large margin solutions. (of course, there would be other ways of measuring the value of margin and we selected a particular one so that we end up with the support vector machine).

We also need to specify the loss function, i.e., measure how well the reference margin γ_{ref} agrees with the margin actually attained, i.e., $\gamma_i(\theta, \theta_0)$. For example, we can define this loss in terms of the ratio γ/γ_{ref}

$$\text{Loss}_h(\gamma/\gamma_{ref}) = \begin{cases} 1 - \gamma/\gamma_{ref}, & \text{if } \gamma < \gamma_{ref} \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

In other words, if the ratio is higher than one, the margin attained is larger than the reference and we incur no loss at all. If the margin for a particular example drops below the reference, we incur a loss $1 - \gamma/\gamma_{ref}$. We will prematurely call this loss function the Hinge loss.

Now, our optimization problem is over θ, θ_0 as well as $\gamma_{ref} > 0$. Specifically, we wish to set these parameters so as to minimize

$$J(\theta, \theta_0, \gamma_{ref}) = \frac{1}{n} \sum_{i=1}^{n} \text{Loss}_h(\gamma_i(\theta, \theta_0)/\gamma_{ref}) + \lambda(1/\gamma_{ref}^2) \tag{5}$$

Note that θ, θ_0 only appear in the optimization problem through the margins $\gamma_i(\theta, \theta_0)$. If we fix θ, θ_0, there will be an optimal value for γ_{ref}. If we fix γ_{ref}, we can find the optimal θ, θ_0, and these values change as a function of γ_{ref}. Put another way, γ_{ref} specifies how much space we wish to have separating the training examples and θ, θ_0 are optimized to best accommodate this wish. For large values of γ_{ref}, not all examples can be separated with that large margin and we start incurring some losses. The solution is then obtained by finding the minimizing balance between the desire for a large margin (the regularizer) and the losses incurred (average Hinge loss).

It turns out that we can simplify this optimization problem further by harnessing an unused degree of freedom in the linear classifier. Specifically, we can fix $\|\theta\|$ to any positive value and still be able to find any linear decision boundary. For example, let's say we want $\|\theta\| = 1$. Then, the decision boundary of any linear classifier θ', θ'_0 coincides with that of $\theta = \theta'/\|\theta'\|$, $\theta_0 = \theta'_0/\|\theta'\|$ (make sure you understand why). Our definition of the margin $\gamma_i(\theta, \theta_0)$ as the distance from the boundary, is similarly unconstrained by the choice of $\|\theta\|$. So, we can express γ_{ref} as a function of $\|\theta\|$ so that the optimization problem will be only over θ, θ_0. Specifically, we set $\gamma_{ref} = 1/\|\theta\|$, so that

$$\gamma_i(\theta, \theta_0)/\gamma_{ref} = y^{(i)}(\theta \cdot x^{(i)} + \theta_0) \tag{6}$$

i.e., the agreement, and $1/\gamma_{ref}^2$ is just $\|\theta\|^2$. Our resulting objective function for large margin classification is then given by

$$J(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} \text{Loss}_h\big(y^{(i)}(\theta \cdot x^{(i)} + \theta_0)\big) + \lambda\|\theta\|^2 \tag{7}$$

The goal is to find θ, θ_0 that minimize $J(\theta, \theta_0)$. Now the choice of $\|\theta\|$ matters a great deal as it is tied to the margin we are after, i.e., the preferred distance to the boundary. For later beautification of derivatives, we will use $\lambda/2$ as the regularization parameter in place of λ above.

3.4 Linear Classification with Loss Functions

We have introduced a more sensitive loss function for linear classification than mere error by penalizing any prediction for which the agreement drops below one (as opposed to zero), and increasing the penalty for greater violations. Specifically, we use the Hinge loss, now taking the agreement as its argument

$$\text{Loss}_h(y(\theta \cdot x + \theta_0)) = \max\{0, 1 - y(\theta \cdot x + \theta_0)\} = \begin{cases} 1 - y(\theta \cdot x + \theta_0) \text{ if } y(\theta \cdot x + \theta_0) \leq 1 \\ 0, \text{ o.w.} \end{cases} \tag{8}$$

By introducing this loss, we do not exclude any linear classifier from being selected but will strongly bias the selection towards particular kinds of classifiers, thereby effectively restricting our choices (in a good way) and improving generalization.

Let's understand geometrically how different parameter choices θ, θ_0 affect the Hinge loss that we incur on each example. The relevant geometric object(s) are no longer the decision boundaries but parallel boundaries around the decision boundary known as the positive and negative margin boundaries $\{x : \theta \cdot x + \theta_0 = 1\}$ and $\{x : \theta \cdot x + \theta_0 = -1\}$, respectively. See Figure 1 below. In order for a positively labeled point to get zero Hinge loss, it must be on the correct (positive) side of the positive margin boundary $\{x : \theta \cdot x + \theta_0 = 1\}$. If we figuratively moved a training example past the relevant margin boundary, the Hinge loss would increase linearly as a function of the degree of violation. The margin boundaries are at distance $1/\|\theta\|$ away from the decision boundary (recall that $\gamma_{ref} = 1/\|\theta\|$ in our derivation above). To see this in a different way, consider a point (x, y) that is correctly classified and lies exactly on the margin boundary. As a result, $y(\theta \cdot x + \theta_0) = 1$, and the orthogonal distance from the boundary is $y(\theta \cdot x + \theta_0)/\|\theta\| = 1/\|\theta\|$. If we kept the decision boundary intact but increased $\|\theta\|$, then the margin boundaries would draw closer to each other. In contrast, a small $|\theta\|$ implies that margin boundaries are far apart, increasing the likelihood of violations.

So, how do we search for linear classifier parameters θ, θ_0, while measuring the performance with respect to the Hinge loss? The goal is to bias our search towards classifiers with small $\|\theta\|$ so that the margin boundaries remain far apart. This forces us to select classifiers where the decision boundary has lots of "empty space" between positive and negative examples. In other words, in contrast to the perceptron algorithm, we try to explicitly find large margin classifiers. Of course, sometimes there will be examples that violate the margin boundaries but we seek to balance such violations against how large the margin is (how small $\|\theta\|$ can be). With this intuition, we are ready to use the optimization problem for finding the maximum margin classifier. The resulting method is called the Support Vector Machine (SVM) and its on-line variant Pegasos.

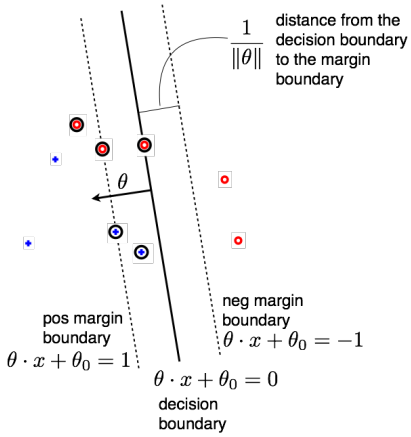

Figure 1: Decision and margin boundaries for a linear classifier. The blue '+'s are positively labeled points while red 'o's are negative. Points that are circled incur a non-zero Hinge loss as they violate the margin boundaries.

3.5 Support Vector Machine

The support vector machine objective function tries to 1) minimize the average Hinge loss on the training examples while 2) pushing margin boundaries apart by reducing $\|\theta\|$. These two goals oppose each other, as discussed above. The more we emphasize the losses, the more the resulting classifier is determined by the few examples right near the decision boundary. In contrast, if we extend the margin boundaries, we incur losses on examples that are further away and the solution will be defined by where the bulk of the positive and negative examples are. We will discuss a bit later how to find the right balance. For now, we will focus on formulating the problem with a prescribed balance. We find θ, θ_0 so as to minimize

$$\frac{1}{n}\sum_{i=1}^{n}\text{Loss}_h\big(y^{(i)}(\theta\cdot x^{(i)}+\theta_0)\big)+\frac{\lambda}{2}\|\theta\|^2 \tag{9}$$

where the *regularization parameter* λ balances these two goals. The second term $\|\theta\|^2/2$ is a *regularization term* that pulls the parameters towards a default answer (here zero vector). By including the regularization term we have made the problem well-posed even when data is conflicting or absent.[1] Figure 2 shows how the solution changes if we change λ when the data are not linearly separable.

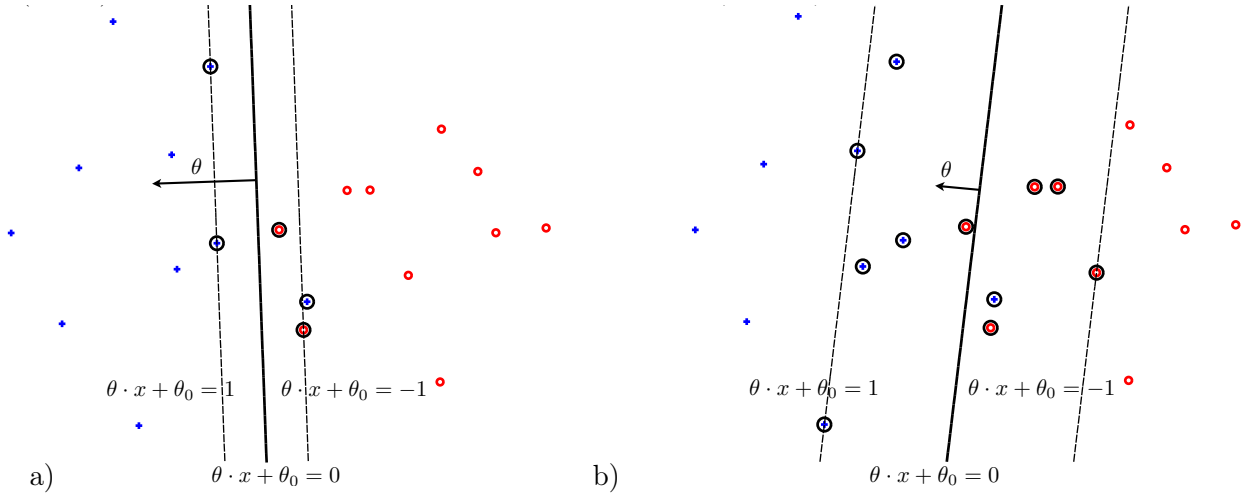

Figure 2: SVM solution when a) λ is small so as to minimize margin violations b) λ is large and we accept margin violations in favor of increasing the size of the margin. Note that $\|\theta\|$ is inversely related to the margin. We have circled all the points that either violate the margin boundaries or lie exactly on them.

[1] While $\theta = 0$ (zero vector) in the absence of data, the offset parameter cannot be determined in this case.

Quadratic program

How do we actually find the minimizing θ and θ_0? The SVM objective function can be reformulated as a *quadratic program* and solved with a variety of available optimization packages. Specifically, we can minimize

$$\frac{1}{n}\sum_{i=1}^{n}\xi_i + \frac{\lambda}{2}\|\theta\|^2 \quad \text{subject to} \quad y^{(i)}(\theta\cdot x^{(i)}+\theta_0) \geq 1-\xi_i, \;\; \xi_i \geq 0, \;\; i=1,\ldots,n \tag{10}$$

with respect to θ, θ_0, and $\xi_i \geq 0$, $i = 1,\ldots,n$. The *slack variables* ξ_i are introduced to represent the Hinge loss in terms of linear constraints so that the overall problem has a quadratic objective with linear constraints. Can you figure out why optimizing ξ_i (keeping other things fixed) will make it exactly the loss on the i^{th} example, i.e., reconstituting $\text{Loss}_h\big(y^{(i)}(\theta\cdot x^{(i)}+\theta_0)\big)$ in the objective?

While it would be good to to find such a solution, this quadratic program doesn't scale well with the number of training examples. Indeed, it takes about $O(dn^3)$ computation to solve it reasonably well in practice, limiting it to smaller problems (on the order of 10,000 training examples without additional tricks).

3.5.1 Stochastic Gradient Descent and the Pegasos Algorithm

We can always try to solve the same SVM minimization problem approximately using much simpler (and scalable) *stochastic gradient methods*. These methods, as on-line algorithms, consider each training example in turn, move the parameters slightly in the negative gradient direction, and move on. Let's again drop the offset parameter θ_0 and rewrite the SVM objective as an average of terms

$$\frac{1}{n}\sum_{i=1}^{n}\text{Loss}_h\big(y^{(i)}(\theta\cdot x^{(i)})\big) + \frac{\lambda}{2}\|\theta\|^2 = \frac{1}{n}\sum_{i=1}^{n}\left[\text{Loss}_h\big(y^{(i)}(\theta\cdot x^{(i)})\big) + \frac{\lambda}{2}\|\theta\|^2\right] \tag{11}$$

When the objective function to be minimized has this form we can always select a term at random and apply a stochastic gradient descent update. Specifically, in response to each example, we update[2]:

$$\begin{aligned}
\theta &= \theta - \eta\nabla_\theta\left[\text{Loss}_h\big(y^{(i)}(\theta\cdot x^{(i)})\big) + \frac{\lambda}{2}\|\theta\|^2\right] &(12)\\
&= \theta - \eta\nabla_\theta\left[\text{Loss}_h\big(y^{(i)}(\theta\cdot x^{(i)})\big)\right] - \eta\nabla_\theta\left[\frac{\lambda}{2}\|\theta\|^2\right] &(13)\\
&= \theta - \eta\nabla_\theta\left[\text{Loss}_h\big(y^{(i)}(\theta\cdot x^{(i)})\big)\right] - \eta\lambda\theta &(14)\\
&= (1-\lambda\eta)\theta - \eta\nabla_\theta\left[\text{Loss}_h\big(y^{(i)}(\theta\cdot x^{(i)})\big)\right] &(15)\\
&= (1-\lambda\eta)\theta + \eta\begin{cases} y^{(i)}x^{(i)} & \text{if } y^{(i)}(\theta\cdot x^{(i)}) \leq 1\\ 0 & \text{o.w.}\end{cases} &(16)
\end{aligned}$$

[2] Technically, this is again a sub-gradient update

where η is the learning rate. If we decrease the learning rate appropriately, these gradient descent updates are indeed guaranteed to converge to the same SVM solution. For example, we can change η_k after each update k such that $\sum_{k=1}^{\infty} \eta_k = \infty$ (not summable) and $\sum_{k=1}^{\infty} \eta_k^2 < \infty$ (square summable). These two conditions are necessary to make sure that, regardless of where we start (or end up during optimization), we can move to the optimum, and that the "noise" inherent in the gradient updates (stochastic choice of terms, finite step-size) vanishes in the end. For example, $\eta_k = 1/(k+1)$ is a valid (albeit somewhat slow) choice.

Algorithm 1 Pegasos Algorithm (without offset)

```
procedure PEGASOS({(x^(i), y^(i)), i = 1, ..., n}, λ, T)
    θ = 0 (vector)
    for t = 1, ..., T do
        Select i ∈ {1, ..., n} at random
        η = 1/t
        if y^(i) θ · x^(i) ≤ 1 then
            θ = (1 − ηλ)θ + η y^(i) x^(i)
        else
            θ = (1 − ηλ)θ
    return θ
```

3.5.2 Maximum Margin Hyperplane – Realizable Case

You may see the SVM optimization problem written slightly differently. First, let's multiply the objective by $1/\lambda$. Since λ is constant with respect to θ and θ_0, this is fine and doesn't change the minimizing solution. We get

$$\overbrace{\left(\frac{1}{\lambda n}\right)}^{=C} \sum_{i=1}^{n} \text{Loss}_h\big(y^{(i)}(\theta \cdot x^{(i)} + \theta_0)\big) + \frac{1}{2}\|\theta\|^2 \tag{17}$$

where the parameter C tells us how much to emphasize the training losses. Whether you prefer to use C or λ as the regularization parameter is immaterial. Just remember that they are inversely related.

What will happen if $C \to \infty$ (or C is very large)? In other words, what do we get as the solution if we adamantly do not want any margin violations? Clearly, the question is relevant only when the data are linearly separable. In this extreme case, we can write the problem as

$$\text{minimize } \frac{1}{2}\|\theta\|^2 \text{ subject to } \text{Loss}_h\big(y^{(i)}(\theta \cdot x^{(i)} + \theta_0)\big) = 0,\ i = 1, \ldots, n \tag{18}$$

or, equivalently,

$$\text{minimize } \frac{1}{2}\|\theta\|^2 \text{ subject to } y^{(i)}(\theta \cdot x^{(i)} + \theta_0) \geq 1,\ i = 1, \ldots, n \tag{19}$$

This is again a *quadratic program* (quadratic objective to be minimized subject to linear constraints). Geometrically, by minimizing $\|\theta\|$ we push the margin boundaries farther and farther apart until we can no longer do this without violating some of the constraints. This gives us the linear separator whose decision boundary is furthest away from all the training examples as shown in Figure 3 below. If the training examples are separable and contain at least one positive and one negative example, this maximum margin solution is unique. However, the maximum margin linear separator can be easily affected by a single misclassified example (do you see why?). For this reason, and to be able to use it even in case of non-separable data, it is better to be able to "give up" on trying to satisfy all the classification constraints.

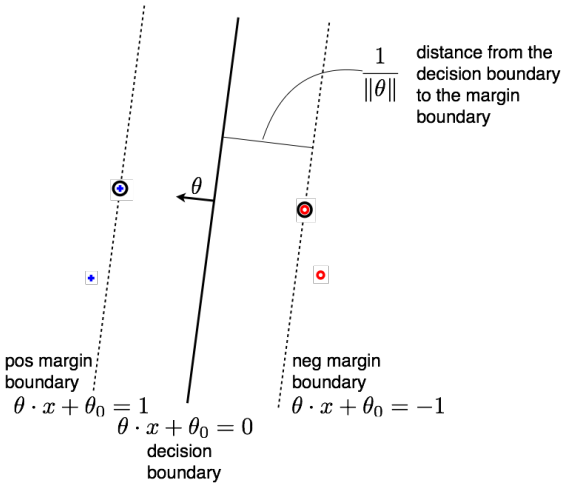

Figure 3: Maximum margin linear separator