15-859(B) Machine Learning Theory

Avrim Blum Lecture 4: January 26, 2009

```
Online Learning contd
* The Perceptron Algorithm
* Perceptron for Approximately Maximizing the Margins
* Kernel Functions
```

Plan for today: Last time we looked at the Winnow algorithm, which has a very nice mistake-bound for learning an OR-function, which we then generalized for learning a linear separator. Today will look at a more classic algorithm for learning linear separators, with a different kind of guarantee.

1 The Perceptron Algorithm

One of the oldest algorithms used in machine learning (from early 60s) is an online algorithm for learning a linear threshold function called the Perceptron Algorithm.

For simplicity, we'll use a threshold of 0, so we're looking at learning functions like:

$$w_1x_1 + w_2x_2 + ... + w_nx_n > 0.$$

We can simulate a nonzero threshold with a "dummy" input x_0 that is always 1, so this can be done without loss of generality. The guarantee we'll show for the Perceptron Algorithm is the following:

Theorem 1 *Let $\mathcal{S}$ be a sequence of labeled examples consistent with a linear threshold function $\mathbf{w}^* \cdot \mathbf{x} > 0$, where $\mathbf{w}^*$ is a unit-length vector. Then the number of mistakes M on $\mathcal{S}$ made by the online Perceptron algorithm is at most $(1/\gamma)^2$, where*

$$\gamma = \min_{\mathbf{x} \in \mathcal{S}} \frac{|\mathbf{w}^* \cdot \mathbf{x}|}{||\mathbf{x}||}.$$

(I.e., if we scale examples to have Euclidean length 1, then γ is the minimum distance of any example to the plane $\mathbf{w}^* \cdot \mathbf{x} = 0$.)

The parameter "γ" is often called the "margin" of $\mathbf{w}^*$ (or more formally, the L_2 margin because we are scaling by the L_2 lengths of the target and examples). Another way to view the quantity $\mathbf{w}^* \cdot \mathbf{x}/||\mathbf{x}||$ is that it is the cosine of the angle between $\mathbf{x}$ and $\mathbf{w}^*$, so we will also use $\cos(\mathbf{w}^*, \mathbf{x})$ for it.

The Perceptron Algorithm:

1. Start with the all-zeroes weight vector $\mathbf{w}_1 = \mathbf{0}$, and initialize t to 1. Also let's automatically scale all examples $\mathbf{x}$ to have (Euclidean) length 1, since this doesn't affect which side of the plane they are on.
2. Given example $\mathbf{x}$, predict positive iff $\mathbf{w}_t \cdot \mathbf{x} > 0$.
3. On a mistake, update as follows:
   - Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \mathbf{x}$.
   - Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \mathbf{x}$.

   $t \leftarrow t + 1$.

So, this seems reasonable. If we make a mistake on a positive $\mathbf{x}$ we get $\mathbf{w}_{t+1} \cdot \mathbf{x} = (\mathbf{w}_t + \mathbf{x}) \cdot \mathbf{x} = \mathbf{w}_t \cdot \mathbf{x} + 1$, and similarly if we make a mistake on a negative $\mathbf{x}$ we have $\mathbf{w}_{t+1} \cdot \mathbf{x} = (\mathbf{w}_t - \mathbf{x}) \cdot \mathbf{x} = \mathbf{w}_t \cdot \mathbf{x} - 1$. So, in both cases we move closer (by 1) to the value we wanted.

Proof of Theorem 1. We're going to look at the magic quantities $\mathbf{w}_t \cdot \mathbf{w}^*$ and $||\mathbf{w}_t||$.

Claim 1: $\mathbf{w}_{t+1} \cdot \mathbf{w}^* \geq \mathbf{w}_t \cdot \mathbf{w}^* + \gamma$. That is, every time we make a mistake, the dot-product of our weight vector with the target increases by at least γ.

> Proof: if $\mathbf{x}$ was a positive example, then we get $\mathbf{w}_{t+1} \cdot \mathbf{w}^* = (\mathbf{w}_t + \mathbf{x}) \cdot \mathbf{w}^* = \mathbf{w}_t \cdot \mathbf{w}^* + \mathbf{x} \cdot \mathbf{w}^* \geq \mathbf{w}_t \cdot \mathbf{w}^* + \gamma$ (by definition of γ). Similarly, if $\mathbf{x}$ was a negative example, we get $(\mathbf{w}_t - \mathbf{x}) \cdot \mathbf{w}^* = \mathbf{w}_t \cdot \mathbf{w}^* - \mathbf{x} \cdot \mathbf{w}^* \geq \mathbf{w}_t \cdot \mathbf{w}^* + \gamma$.

Claim 2: $||\mathbf{w}_{t+1}||^2 \leq ||\mathbf{w}_t||^2 + 1$. That is, every time we make a mistake, the length squared of our weight vector increases by at most 1.

> Proof: if $\mathbf{x}$ was a positive example, we get $||\mathbf{w}_t + \mathbf{x}||^2 = ||\mathbf{w}_t||^2 + 2\mathbf{w}_t \cdot \mathbf{x} + ||\mathbf{x}||^2$. This is less than $||\mathbf{w}_t||^2 + 1$ because $\mathbf{w}_t \cdot \mathbf{x}$ is negative (remember, we made a mistake on $\mathbf{x}$). Same thing (flipping signs) if $\mathbf{x}$ was negative but we predicted positive.

Claim 1 implies that after M mistakes, $\mathbf{w}_{M+1} \cdot \mathbf{w}^* \geq \gamma M$. On the other hand, Claim 2 implies that after M mistakes, $||\mathbf{w}_{M+1}|| \leq \sqrt{M}$. Now, all we need to do is use the fact that $\mathbf{w}_t \cdot \mathbf{w}^* \leq ||\mathbf{w}_t||$, since $\mathbf{w}^*$ is a unit vector. So, this means we must have $\gamma M \leq \sqrt{M}$, and thus $M \leq 1/\gamma^2$. ■

Discussion: In the worst case, γ can be exponentially small in n. On the other hand, if we're lucky and the data is well-separated, γ might even be large compared to $1/n$. This is called the "large margin" case. (In fact, the latter is the more modern spin on things: namely, that in many natural cases, we would hope that there exists a large-margin separator.) In fact, one nice thing here is that the mistake-bound depends on just a purely geometric quantity:

the amount of "wiggle-room" available for a solution and doesn't depend in any direct way on the number of features in the space.

So, if data is separable by a large margin, then Perceptron is a good algorithm to use.

What if there is no perfect separator? What if only *most* of the data is separable by a large margin, or what if $\mathbf{w}^*$ is not perfect? We can see that the thing we need to look at is Claim 1. Claim 1 said that we make "γ amount of progress" on every mistake. Now it's possible there will be mistakes where we make very little progress, or even negative progress. One thing we can do is bound the total number of mistakes we make in terms of the total distance we would have to move the points to make them actually separable by margin γ. Let's call that TD_γ. Then, we get that after M mistakes, $\mathbf{w}_{M+1} \cdot \mathbf{w}^* \geq \gamma M - \text{TD}_\gamma$. So, combining with Claim 2, we get that $\sqrt{M} \geq \gamma M - \text{TD}_\gamma$. We could solve the quadratic, but this implies, for instance, that $M \leq 1/\gamma^2 + (2/\gamma)\text{TD}_\gamma$. The quantity $\frac{1}{\gamma}\text{TD}_\gamma$ is called the total *hinge-loss* of w^*.

So, this is not too bad: we can't necessarily say that we're making only a small multiple of the number of mistakes that $\mathbf{w}^*$ is (in fact, the problem of finding an approximately-optimal separator is NP-hard), but we can say we're doing well in terms of the "total distance" parameter.

Perceptron for approximately maximizing margins. We saw that the perceptron algorithm makes at most $1/\gamma^2$ mistakes on any sequence of examples that is linearly-separable by margin γ (i.e., any sequence for which there exists a unit-length vector $\mathbf{w}^*$ such that all examples $\mathbf{x}$ satisfy $\ell(\mathbf{x})(\mathbf{w}^* \cdot \mathbf{x})/||\mathbf{x}|| \geq \gamma$, where $\ell(\mathbf{x}) \in \{-1, 1\}$ is the label of $\mathbf{x}$).

Suppose we are handed a set of examples $\mathcal{S}$ and we want to actually find a *large-margin* separator for them. One approach is to directly solve for the maximum-margin separator using convex programming (which is what is done in the SVM algorithm). However, if we only need to *approximately* maximize the margin, then another approach is to use Perceptron. In particular, suppose we cycle through the data using the Perceptron algorithm, updating not only on mistakes, but also on examples $\mathbf{x}$ that our current hypothesis gets correct by margin less than $\gamma/2$. Assuming our data is separable by margin γ, then we can show that this is guaranteed to halt in a number of rounds that is polynomial in $1/\gamma$. (In fact, we can replace $\gamma/2$ with $(1-\epsilon)\gamma$ and have bounds that are polynomial in $1/(\epsilon\gamma)$.)

The Margin Perceptron Algorithm(γ):

1. Assume again that all examples are normalized to have Euclidean length 1. Initialize $\mathbf{w}_1 = \ell(\mathbf{x})\mathbf{x}$, where $\mathbf{x}$ is the first example seen and initialize t to 1.

2. Predict positive if $\frac{\mathbf{w}_t \cdot \mathbf{x}}{||\mathbf{w}_t||} \geq \gamma/2$, predict negative if $\frac{\mathbf{w}_t \cdot \mathbf{x}}{||\mathbf{w}_t||} \leq -\gamma/2$, and consider an example to be a margin mistake when $\frac{\mathbf{w}_t \cdot x}{||\mathbf{w}_t||} \in (-\gamma/2, \gamma/2)$.

3. On a mistake (incorrect prediction or margin mistake), update as in the standard Perceptron algorithm: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \ell(\mathbf{x})\mathbf{x}$; $t \leftarrow t + 1$.

Theorem 2 *Let $\mathcal{S}$ be a sequence of labeled examples consistent with a linear threshold func-*

tion $\mathbf{w}^* \cdot \mathbf{x} > 0$*, where* $\mathbf{w}^*$ *is a unit-length vector, and let*

$$\gamma = \min_{\mathbf{x} \in \mathcal{S}} \frac{|\mathbf{w}^* \cdot \mathbf{x}|}{||\mathbf{x}||}.$$

Then the number of mistakes (including margin mistakes) made by Margin Perceptron(γ) *on* $\mathcal{S}$ *is at most* $8/\gamma^2$*.*

Proof: The argument for this new algorithm follows the same lines as the argument for the original Perceptron algorithm.

As before, each update increases $\mathbf{w}_t \cdot \mathbf{w}^*$ by at least γ. What is now a little more complicated is to bound the increase in $||\mathbf{w}_t||$, since we are updating on some examples where the angle is more than 90^o. For the original algorithm, we had: $||\mathbf{w}_{t+1}||^2 \leq ||\mathbf{w}_t||^2 + 1$, which implies $||\mathbf{w}_{t+1}|| \leq ||\mathbf{w}_t|| + \frac{1}{2||\mathbf{w}_t||}$.

For the new algorithm, we instead get

$$||\mathbf{w}_{t+1}|| \leq ||\mathbf{w}_t|| + \frac{1}{2||\mathbf{w}_t||} + \frac{\gamma}{2},$$

which we can see by breaking each $\mathbf{x}$ into its orthogonal part (for which the original statement holds) and its parallel part (which adds at most $\gamma/2$ to the length of $\mathbf{w}_t$).

We can now solve this directly, but just to get a simple upper bound, just notice that if $||\mathbf{w}_t|| \geq 2/\gamma$ then $||\mathbf{w}_{t+1}|| \leq ||\mathbf{w}_t|| + 3\gamma/4$. So, after M updates we have:

$$||\mathbf{w}_{M+1}|| \leq 2/\gamma + 3M\gamma/4.$$

Solving $M\gamma \leq 2/\gamma + 3M\gamma/4$ we get $M \leq 8/\gamma^2$, as desired. ■

For more information on perceptron and the analyses given here, see [Blo62, Nov62, MP69, FS99, SSS05, TST05, BB06].

2 Kernel functions

What if our data doesn't have a good linear separator? Here's a neat idea, called the *kernel trick*.

One thing we might like to do is map our data to a higher dimensional space, e.g., look at all products of pairs of features, in the hope that data will be linearly separable there. If we're lucky, data will be separable by a large margin so we don't have to pay a lot in terms of mistakes. But this is going to a pain computationally. However, it turns out that many learning algorithms only access data through performing dot-products (will get back to how to interpret algorithms like Perceptron in this way in a minute). So, maybe we can perform our mapping in such a way that we have an efficient way of computing dot-products. This leads to idea of a kernel.

A Kernel is a function $K(\mathbf{x}, \mathbf{y})$ such that for some mapping ϕ, $K(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{y})$.

Some examples:

- $K(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x} \cdot \mathbf{y})^d$.
- $K(\mathbf{x}, \mathbf{y}) = (1 + x_1 y_1)(1 + x_2 y_2)...(1 + x_n y_n)$
  [corresponds to mapping $\mathbf{x}, \mathbf{y}$ to list of all products of subsets]
- String kernels [count how many substrings of length p two strings have in common]

More generally, this is nice for the case where examples aren't so easy to map directly into R^n, but we have a reasonable notion of similarity we can encode in a kernel K.

Neat fact: many of the learning algorithms for learning linear separators can be run using kernels. E.g., for the Perceptron algorithm, $\mathbf{w}_t$ is a weighted sum of examples, for all t. I.e.,

$$\mathbf{w}_t = \ell(\mathbf{x}_{i_1})\mathbf{x}_{i_1} + \cdots + \ell(\mathbf{x}_{i_k})\mathbf{x}_{i_{t-1}},$$

where $\mathbf{x}_{i_1}$, ...,$\mathbf{x}_{i_{t-1}}$ are the examples where we've made mistakes so far. So to compute $\phi(\mathbf{w}_t) \cdot \phi(\mathbf{x})$, just do:

$$\ell(\mathbf{x}_{i_1})K(\mathbf{x}_{i_1}, \mathbf{x}) + \cdots + \ell(\mathbf{x}_{i_{t-1}})K(\mathbf{x}_{i_{t-1}}, \mathbf{x}).$$

The examples that the hypothesis is written in terms of are called *support vectors*. If we find the maximum margin separator for a given dataset, that is also something that can be written in terms of support vectors (not hard to see). That's the reason for the name "support vector machines" for the algorithm that takes a set of data and finds the maximum-margin separator.

References

[BB06] M.F. Balcan and A. Blum. On a theory of learning with similarity functions. In *Proc. International Conference on Machine Learning (ICML)*, pages 73–80, 2006.

[Blo62] H.D. Block. The perceptron: A model for brain functioning. *Reviews of Modern Physics*, 34:123–135, 1962. Reprinted in *Neurocomputing*, Anderson and Rosenfeld.

[FS99] Y. Freund and R.E. Schapire. Large margin classification using the Perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

[MP69] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry.* The MIT Press, 1969.

[Nov62] A.B.J. Novikoff. On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata, Vol. XII*, pages 615–622, 1962.

[SSS05] S. Shalev-Shwartz and Y. Singer. A new perspective on an old perceptron algorithm. In *Proc. 16thh Annual Conference on Learning Theory*, 2005.

[TST05] P. Tsampouka and J. Shawe-Taylor. Analysis of generic perceptron-like large margin classifiers. In *ECML*, 2005.