Today we discuss an algorithm for solving linear programming feasibility problems of the form $A\mathbf{w} \geq \mathbf{1}$, where $\mathbf{w}$ is our vector of variables. It will in fact do more than this, as we will see.[1]

1 Online linear classification

Consider the following problem. You have a sequence of email messages arriving one at a time $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \ldots$, and for each one you (the algorithm) have to decide whether to mark it "important" or "not important". After marking it, you are then told (by your user) if you were correct or if you made a mistake.

Let's assume that email messages are represented as vectors in some space $\mathbb{R}^n$, such as indicating how many times each word in the dictionary appears in the email message (in this case, n is the number of words in the dictionary, and this would be called a "bag of words" model). Let's furthermore suppose that there exists some unknown weight vector $\mathbf{w}^*$ such that $\mathbf{a}_i \cdot \mathbf{w}^* \geq 1$ for the important emails (the *positive* examples) and $\mathbf{a}_i \cdot \mathbf{w}^* \leq -1$ for the non-important emails (the *negative* examples). Our goal is to give an algorithm for performing this task that makes as few mistakes as possible.

2 The Perceptron Algorithm

The Perceptron Algorithm is one such algorithm for this problem. It maintains a weight vector $\mathbf{w}$ that it uses for prediction (predicting positive on $\mathbf{a}_i$ if $\mathbf{a}_i \cdot \mathbf{w} > 0$ and predicting negative if $\mathbf{a}_i \cdot \mathbf{w} < 0$) and then updates it when it makes a mistake (let's say it says "I don't know" if $\mathbf{a}_i \cdot \mathbf{w} = 0$ and so makes a mistake either way). What we will prove about the algorithm is the following theorem.

Theorem 1 *[Block ('62), Novikoff ('62), Minsky-Papert ('69)] On any sequence of examples $\mathbf{a}_1, \mathbf{a}_2, \ldots$, if there exists a consistent $\mathbf{w}^*$, i.e., $\mathbf{a}_i \cdot \mathbf{w}^* \geq 1$ for the positive examples and $\mathbf{a}_i \cdot w^* \leq -1$ for the negative examples, then the Perceptron algorithm makes at most $R^2||\mathbf{w}^*||^2$ mistakes, where $R = \max_i ||\mathbf{a}_i||$.*

To get a feel for this statement, notice that if we multiply all entries in all the $\mathbf{a}_i$ by 100, we can divide all entries in $\mathbf{w}^*$ by 100 and it will still be consistent. So the bound is invariant to this kind of scaling (i.e., what our "units" are).

Secondly, if we rewrite $\mathbf{a}_i \cdot \mathbf{w}^* \geq 1$ as $\mathbf{a}_i \cdot \mathbf{w}^*/||\mathbf{w}^*|| \geq 1/||\mathbf{w}^*||$, and we think of the $\mathbf{a}_i$ as points in $\mathbb{R}^n$, then we can think of the LHS as the distance of $\mathbf{a}_i$ to the hyperplane $\mathbf{a} \cdot \mathbf{w}^* = 0$. So, we can think of w^* as defining a *linear separator* that separates the positive examples from the negative examples, and what the theorem is saying is that if there exists a hyperplane that correctly separates the positive examples from the negative examples by a large *margin*, then the total number of mistakes will be small.

2.1 Perceptron for Solving Linear Programs

Before giving the algorithm and analysis, note that by the above theorem, we can use it to solve "feasibility" linear programs of the form $A\mathbf{w} \geq \mathbf{1}$. I.e., these are LPs where we just want to find

[1] Note the change in notation, using $\mathbf{w}$ instead of $\mathbf{x}$; this will avoid confusion: traditionally in our machine learning motivation, the data is called $\mathbf{x}_1, \mathbf{x}_2, \ldots$, whereas in LPs $\mathbf{x}$ is the solution. So we'll just not use $\mathbf{x}$ at all today.

$\mathbf{w} \in \mathbb{R}^n$ such that $\mathbf{a}_i \cdot \mathbf{w} \geq 1$ for all $i \in [m]$. (Since A may contain both positive and negative values, finding such a $\mathbf{w}$ is not trivial. Such LPs are called "cone" LPs.) In fact, given any LP of the form $A\mathbf{w} \geq \mathbf{b}$, we could multiply each entry on the i^{th} row $\mathbf{a}_i$ of A by $1/b_i$, and transform it into $A'\mathbf{w} \geq \mathbf{1}$ where $a'_{ij} = a_{ij}/b_i$.

How to solve this feasibility LP? Just cycle through the constraints (viewing each one as a positive example) until the algorithm stops making any more mistakes. At this point we have $A\mathbf{w} > \mathbf{0}$, and we can just scale up $\mathbf{w}$ to be large enough so that $A\mathbf{w} \geq \mathbf{1}$.

2.2 The algorithm

The Perceptron algorithm (due to Frank Rosenblatt) is simply the following:

1. Begin with $\mathbf{w} = \mathbf{0}$.
2. Given $\mathbf{a}_i$, predict positive if $\mathbf{a}_i \cdot \mathbf{w} > 0$ and predict negative if $\mathbf{a}_i \cdot \mathbf{w} < 0$, else say "I don't know".
3. If a mistake was made (or the prediction was "I don't know"):
   - If the correct answer was "positive", update: $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{a}_i$.
   - If the correct answer was "negative", update: $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{a}_i$.

2.3 The analysis

Proof (Theorem 1): Fix some consistent $\mathbf{w}^*$. We will keep track of two quantities, $\mathbf{w} \cdot \mathbf{w}^*$ and $||\mathbf{w}||^2$.

- First of all, each time we make a mistake, $\mathbf{w} \cdot \mathbf{w}^*$ increases by at least 1. That is because if $\mathbf{a}_i$ is a positive example, then

  $$(\mathbf{w} + \mathbf{a}_i) \cdot \mathbf{w}^* = \mathbf{w} \cdot \mathbf{w}^* + \mathbf{a}_i \cdot \mathbf{w}^* \geq \mathbf{w} \cdot \mathbf{w}^* + 1,$$

  by definition of $\mathbf{w}^*$. Similarly, if $\mathbf{a}_i$ is a negative example, then

  $$(\mathbf{w} - \mathbf{a}_i) \cdot \mathbf{w}^* = \mathbf{w} \cdot \mathbf{w}^* - \mathbf{a}_i \cdot \mathbf{w}^* \geq \mathbf{w} \cdot \mathbf{w}^* + 1.$$

- Next, on each mistake, we claim that $||\mathbf{w}||^2$ increases by at most R^2. Let us first consider mistakes on positive examples. If we make a mistake on a positive example $\mathbf{a}_i$ then we have

  $$(\mathbf{w} + \mathbf{a}_i) \cdot (\mathbf{w} + \mathbf{a}_i) = ||\mathbf{w}||^2 + 2\mathbf{w} \cdot \mathbf{a}_i + ||\mathbf{a}_i||^2 \leq ||\mathbf{w}||^2 + ||\mathbf{a}_i||^2 \leq ||\mathbf{w}||^2 + R^2,$$

  where the middle inequality comes from the fact that we made a mistake, which means that $\mathbf{w} \cdot \mathbf{a}_i \leq 0$. Similarly, if we make a mistake on a negative example $\mathbf{a}_i$ then we have

  $$(\mathbf{w} - \mathbf{a}_i) \cdot (\mathbf{w} - \mathbf{a}_i) = ||\mathbf{w}||^2 - 2\mathbf{w} \cdot \mathbf{a}_i + ||\mathbf{a}_i||^2 \leq ||\mathbf{w}||^2 + ||\mathbf{a}_i||^2 \leq ||\mathbf{w}||^2 + R^2.$$

  Note that it is important here that we only update on a mistake.

So, if we make M mistakes, then $\mathbf{w} \cdot \mathbf{w}^* \geq M$ (by the first bullet point). Also, $||\mathbf{w}||^2 \leq MR^2$, or equivalently, $||\mathbf{w}|| \leq R\sqrt{M}$ (by the second).

Finally, we use the fact that $\mathbf{w} \cdot \mathbf{w}^*/||\mathbf{w}^*|| \leq ||\mathbf{w}||$ which is just saying that the projection of $\mathbf{w}$ in the direction of $\mathbf{w}^*$ cannot be larger than the length of $\mathbf{w}$. This gives us:

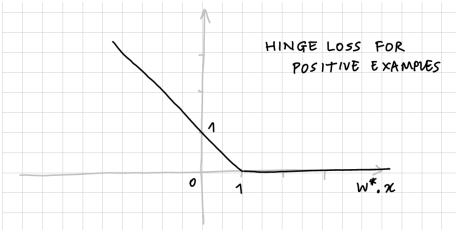
$$\begin{aligned} M/||\mathbf{w}^*|| &\leq R\sqrt{M} \\ \sqrt{M} &\leq R||\mathbf{w}^*|| \\ M &\leq R^2||\mathbf{w}^*||^2 \end{aligned}$$

as desired. ■

2.4 Extensions and hinge-loss

We assumed above that there existed a perfect $\mathbf{w}^*$ that correctly classified all the examples, i.e., correctly classified all the emails into important versus non-important. This is rarely the case in real-life data. What if $\mathbf{w}^*$ isn't quite perfect? We can see what this does to the above proof: if there is an example that $\mathbf{w}^*$ doesn't correctly classify, then while the second part of the proof still holds, the first part (the dot product of $\mathbf{w}$ with $\mathbf{w}^*$ increasing) breaks down. However, if this doesn't happen too often, and also $\mathbf{a}_i \cdot \mathbf{w}^*$ is just a "little bit wrong" then this just means we will make a few more mistakes.

To make this formal, define the *hinge-loss* of $\mathbf{w}^*$ on a positive example $\mathbf{a}_i$ as $\max(0, 1 - \mathbf{a}_i \cdot \mathbf{w}^*)$. In other words, if $\mathbf{a}_i \cdot \mathbf{w}^* \geq 1$ as desired then the hinge-loss is zero; else, the hinge-loss is the amount the LHS is less than the RHS.[2]



Similarly, the hinge-loss of $\mathbf{w}^*$ on a negative example $\mathbf{a}_i$ is $\max(0, 1 + \mathbf{a}_i \cdot \mathbf{w}^*)$. Given a sequence of labeled examples A, define the total hinge-loss $L_{hinge}(\mathbf{w}^*, A)$ as the sum of hinge-losses of $\mathbf{w}^*$ on all examples in A. We now get the following extended theorem.

Theorem 2 *On any sequence of examples $A = \mathbf{a}_1, \mathbf{a}_2, \ldots$, the Perceptron algorithm makes at most*

$$\min_{\mathbf{w}^*} \left(R^2||\mathbf{w}^*||^2 + 2L_{hinge}(\mathbf{w}^*, A) \right)$$

mistakes, where $R = \max_i ||\mathbf{a}_i||$.

Proof: As before, each update of the Perceptron algorithm increases $||\mathbf{w}||^2$ by at most R^2, so if the algorithm makes M mistakes, we have $||\mathbf{w}||^2 \leq MR^2$.

What we can no longer say is that each update of the algorithm increases $\mathbf{w} \cdot \mathbf{w}^*$ by at least 1. Instead, on a positive example we are "increasing" $\mathbf{w} \cdot \mathbf{w}^*$ by $\mathbf{a}_i \cdot \mathbf{w}^*$ (it could be negative), which is at least $1 - L_{hinge}(\mathbf{w}^*, \mathbf{a}_i)$. Similarly, on a negative example we "increase" $\mathbf{w} \cdot \mathbf{w}^*$ by $-\mathbf{a}_i \cdot \mathbf{w}^*$, which is also at least $1 - L_{hinge}(\mathbf{w}^*, \mathbf{a}_i)$. If we sum this up over all mistakes, we get that at the end we have $\mathbf{w} \cdot \mathbf{w}^* \geq M - L_{hinge}(\mathbf{w}^*, A)$, where we are using here the fact that hinge-loss is never negative so summing over all of A is only larger than summing over the mistakes that $\mathbf{w}$ made.

[2]This is called "hinge-loss" because as a function of $\mathbf{a}_i \cdot \mathbf{w}^*$ it looks like a hinge.

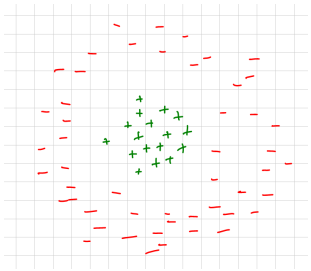
Finally, we just do some algebra. Let $L = L_{hinge}(\mathbf{w}^*, A)$. If $M \leq L$, then the theorem is clearly true. Else we can assume that $M > L$, and hence $0 < M - L \leq \mathbf{w} \cdot \mathbf{w}^*$. Now we have:

$$\begin{aligned}
\mathbf{w} \cdot \mathbf{w}^* / ||\mathbf{w}^*|| &\leq ||\mathbf{w}|| \\
\Rightarrow (\mathbf{w} \cdot \mathbf{w}^*)^2 &\leq ||\mathbf{w}||^2 ||\mathbf{w}^*||^2 \\
\Rightarrow (M - L)^2 &\leq M R^2 ||\mathbf{w}^*||^2 \\
\Rightarrow M^2 - 2ML + L^2 &\leq M R^2 ||\mathbf{w}^*||^2 \\
\Rightarrow M - 2L + L^2/M &\leq R^2 ||\mathbf{w}^*||^2 \\
\Rightarrow M &\leq R^2 ||\mathbf{w}^*||^2 + 2L - L^2/M \;\leq\; R^2 ||\mathbf{w}^*||^2 + 2L
\end{aligned}$$

as desired. ■

2.5 Kernel functions

What if even the best $\mathbf{w}^*$ has high hinge-loss? E.g., maybe instead of a linear separator decision boundary, the boundary between important emails and unimportant emails looks more like a circle?



A neat idea for addressing situations like this is to use what are called *kernel functions*, or sometimes the "kernel trick". Here is the idea. Suppose you have a function K (called a "kernel") over pairs of data points such that for some (doesn't even have to be known) function $\phi : \mathbb{R}^n \to \mathbb{R}^N$ (where perhaps $N \gg n$) we have $K(\mathbf{a}_i, \mathbf{a}_j) = \phi(\mathbf{a}_i) \cdot \phi(\mathbf{a}_j)$.

In that case, if we can write the Perceptron algorithm so that it only interacts with the data via dot products, and then replace every dot-product with an invocation of K, then we can act as if we had performed the function ϕ explicitly without having to actually compute ϕ. For example, consider $K(\mathbf{a}_i, \mathbf{a}_j) = (1 + \mathbf{a}_i \cdot \mathbf{a}_j)^d$. It turns out this corresponds to a mapping ϕ into a space of dimension $N \approx n^d$ (try doing it with $d = 2$), and perhaps in this higher-dimensional space there is a $\mathbf{w}^*$ such that the bound of Theorem 2 is small. But the nice thing is we didn't have to computationally perform the mapping ϕ!

So, how can we view the Perceptron algorithm as only interacting with data via dot-products? Notice that $\mathbf{w}$ is always a linear combination of data points, e.g., we might have $\mathbf{w} = \mathbf{a}_1 + \mathbf{a}_2 - \mathbf{a}_5$. So if we keep track of it this way, and need to predict on a new example $\mathbf{a}_6$, we can write $\mathbf{w} \cdot \mathbf{a}_6 = \mathbf{a}_1 \cdot \mathbf{a}_6 + \mathbf{a}_2 \cdot \mathbf{a}_6 - \mathbf{a}_5 \cdot \mathbf{a}_6$. So if we just replace each of these dot-products with "K", we are running the algorithm as if we had explicitly performed the ϕ mapping. This is called "kernelizing" the algorithm.

3 Other Notes

3.1 Perceptron's Worst-Case Runtime

The runtime of Perceptron may be exponential in the number of bits required to write down the input. Here's an example:

(Take the powerpoint example from today's lecture. The positive points are at $(1, 1)$, $(1, 0)$, and a negative point at $(0, 1)$.) Now add in another positive point at $(1/K, 1)$ for some large integer $K > 0$. Note that the length of each vector $\mathbf{a}_i$ is no more than $\sqrt{2}$ so the R^2 term is only 2.

Moreover, the total length of the input is $O(\log K)$ bits. This is because the number of bits to write down a fraction $1/K$ is only $O(\log K)$. And numbers 0 and 1 take $O(1)$ bits.

But if you run the Perceptron algorithm, you'll see that you will run for at least $\Omega(K)$ steps, maybe even more, before you find a linear separator for this data. And $\Omega(K)$ is exponential in the length of the input.

3.2 Affine Linear Separators

Suppose we want a linear separator that did not necessarily pass through the origin. I.e., we want a vector $\mathbf{w}^*$ and real b, such that $\mathbf{w}^* \cdot \mathbf{a}_i > b$ for the positive examples and $\mathbf{w}^* \cdot \mathbf{a}_i < b$ for the negative ones. Call the pair $(\mathbf{w}^*, b)$ an affine separator. Here's an easy way to do this using Perceptron.

Add a new coordinate to each data point $\mathbf{a}_i \in \mathbb{R}^n$ and put value 1 in it—call the new "lifted" points $\mathbf{a}'_i \in \mathbb{R}^{n+1}$. Then for any $\mathbf{w}^*, b$ as above, you can define vector $\mathbf{v}^* \in \mathbb{R}^{n+1}$ to be the same as $\mathbf{w}^*$ in the first n coordinates, and with value $-b$ in the $n + 1^{st}$ coordinate. Observe $\mathbf{v}^* \cdot \mathbf{a}'_i = \mathbf{w}^* \cdot \mathbf{a}_i - b$ which is > 0 for the positive examples and < 0 for the negative ones. So there exists a linear separator $\mathbf{v}^*$ for the lifted points if and only if there exists an affine separator for the original points.

Now, you could just run Perceptron on the lifted points, find the vector $\mathbf{v} = (v_1, v_2, \ldots, v_{n+1}) \in \mathbb{R}^{n+1}$, and be able to infer that $\mathbf{w} = (v_1, \ldots, v_n), b = -v_{n+1}$ is a good affine separator.