SNA Assignment #03

Mujtaba Asim
bscs22012@itu.edu.pk

Q1: After how many iterations of the perceptron algorithm are we guaranteed to find that a data point has been correctly classified?

Lines, Planes, Hyperplanes, x and w

To begin with, I will explain some fundamental, underlying principles I had to read up on/refresh my memory of in order to attempt this assignment.

The equation of a line is $ax + by + c$ and the equation of a plane is $ax + by + cz + d$.

One could imagine this pattern extending onwards to infinity could trivially, but it would quickly get irksome to write out, so instead we generalise the aforementioned expressions like so: $w^T x = w_1 x_1 + w_2 x_2 + ...w_d x_d$ where d is the number of dimensions that x is spread out across.

Each 'feature component' i.e 'part' of x from an arbitrarily selected dimension corresponds to the x, y, or z i.e axes from the original equation (of course, these are unwieldy to continue with once d surpasses 3) and each part of w, i.e w_i is the coefficient of x_i in that dimension.

All the weights together are referred to as the weight vector w. To take the dot product of them with all the feature components of x, we must first trans pose this vector so they form a row rather than a column vector.

Now, let us attempt to understand why we are even modifying w in the first place.

Orthogonality b/w w and $w^T x$

To begin with, we must acknowledge that w is perpendicular to the hyperplane, also referred to as the 'decision boundary' generated by $w^T x$. This is trivially provable by momentarily setting d to 2.

$$w_1 x_1 + w_2 x_2 = 0$$

$$w^T x = 0$$

Pick any point p and q on the line.

$$w^T p = 0, w^T q = 0$$

One vector along the line is $q - p$. Thus:

$$w^T (q - p) = 0$$

And we know that the only time a dot product results in 0 is when the two input vectors are orthogonal i.e perpendicular to each other. This same concept can be extrapolated to higher dimensions.

As for why this orthogonality is important? Tweaking w effectively tweaks the hyperplane because the only other input in the equation of the hyperplane, x is costant at that point in time.

How is w used for classification?

In the hyperplane itself, all the vectors x such that $w^T x = 0$ lie exactly on the plane itself. The act of classification is trivially 'segregating' a point based on which side of the hyperplane it's fallen on.

If $w^T x > 0$, then the point x is the on the same side of the hyperplane as the weight vector w. If the dot product is a negative value, then the vector x must lie on the opposite side as that of w. The reason for this being the $\cos\theta$ component in the dot product formula – if x is on the same side as w, $\theta < 90$, i.e $\cos\theta > 0$.

How do we correct w?

Well, the first step is to understand *how far off* from the hyperplane the vector x in question is.

To this end, the magnitude of the unsigned hyperplane expression, $|w^T x|$, tells us the distance of the vector x is from the hyperplane, perpendicularly (after dividing it by the magnitude of w i.e $\|w\|$).

And why must we divide by $\|w\|$?

The shortest distance from a point to a plane is always perpendicular to it. w is exactly this. Therefore, the distance from x to the hyperplane boils down to the question "how much of x lies in the direction of w?" Pedantically, this is the *projection length* of x onto w.

This can also be derived via some trigonometry rules, but for brevity's sake, it's $\|x\|\cos\theta$. Observing closely, we find it to be a subset of the dot product expression generated for $w^T x$, and as such

$$\frac{\|w\|\ \|x\|\cos\theta}{\|w\|} = \|x\|\cos\theta$$

Updation Procedure

Once the aforementioned is clear, the actual updation is quite straightforward.

$$w = w + yx$$

Where y is the true label and it's value is equal to either +1 or −1. In the case of the former, w moves towards x, thereby increasing the prior-mentioned desirable 'projection of x along w.' Conversely, in the case of the latter, the expression becomes $w = w - x$, thereby taking w **away** from x.

With all this background out of the way, we can now *finally* start determining how many w updates are needed to correctly classify an arbitrary vector x

Maximum Number of w iterations

To begin with, I will provide an intuitive explanation before proceeding with a formalism. The latter necessitates the definition of a new term i.e the 'margin' (γ). Succinctly put, this refers to the distance between the hyperplane and the point in the dataset nearest to it: $\mathbf{min}_{\boldsymbol{x},\boldsymbol{y}\in\boldsymbol{D}}\ \boldsymbol{yw}\cdot\boldsymbol{x}$.

Intuition

For simplicity's sake, let us assume our dataset contains only two points. If they are at a great distance from each other, we are quite likely to find an arbitrary hyperplane (generated by an arbitrary w) that succesfully segregates them much quicker.

Conversely, as the distance between them shrinks infinitesimally smaller, finding a valid decision boundary becomes a far more daunting task, as a far more precise configuration of elements in the weight vector would be required.

Formalism

I would like to preface this proof with two assumptions:

1. The norm of all points x in the dataset is 1 – we will later look at a generalisation
2. The dataset is linearly separable – recall that if this condition is not met, the perceptron algorithm loops infinitely

To begin with, let us assume that a 'perfect' weight vector w^*, that generates the perfect decision boundary, exists, the dataset has a margin γ, and w_k is the weight vector after the k-th iteration.

The goal of the perceptron algorithm is to have w_k, (or w_{new} at the latest iteration) approach w^*. Pedantically, we want the projection of the former on the latter (i.e dot product) to increase.

However, the dot product can also increase by virtue of an increasing magnitude, so we show in Q2 that the magnitude is, in fact, not increasing that significantly. Therefore, the only way the dot product can increase is if the angle between the two weight vectors decreases.

$$w^* \cdot w_{\text{new}} = w^* \cdot (w_{\text{old}} + yx)$$

$$= w^* \cdot w_{\text{old}} + yw^* \cdot x$$

$$\geq w^* \cdot w_{\text{old}} + \gamma$$

The above can be interpeted to state that everytime w_k is updated, its projection onto w^* increases by at least γ. Thus: $\boldsymbol{w^* \cdot w_k \geq k\gamma}$.

By Q2, we know that the squared norm of w_k increases by *at most* one with every update, which means $\|w_k\|^2 \leq k$ or $\boldsymbol{\|w_k\| \leq \sqrt{k}}$.

Additionally, by virtue of w^* being a unit vector, we know that $\boldsymbol{\|w_k\| > w^* \cdot w_k}$ – as the length of a vector's projection onto another vector can not be longer than the vector itself.

Combining the three aforementioned points…

$$\sqrt{k} \geq \|w_k\| > w^* \cdot w_k \geq k\gamma$$

Taking the extreme terms, we observe $\sqrt{k} \geq k\gamma$. Solving for k, we see that it is **guaranteed to be less than or equal to $\frac{1}{\gamma^2}$**.

Q2 Bound the amount by which the norm of the weights vector may increase in a single iteration.

This expression is easier to solve by first squaring.

$$\|w_{\text{new}}\|^2 = \|w_{\text{old}} + yx\|^2$$

$$= \|w_{\text{old}}\|^2 + y^2 \ \|x\|^2 + 2yw_{\text{old}} \cdot x$$

$$\leq \|w_{\text{old}}\|^2 + 1 + 0$$

With regards to the final term ($yw_{\text{old}} \cdot x$), we know that updation only happens when the sign of $yw \cdot x$ is negative in the first place. Hence, in the absolute best-case scenario, when the point gets misclassified by an infinitesimally small deviation from the decision boundary, the value of $yw \cdot x$ will approach 0.

Subtracting $\|w_{\text{new}}\|^2$ from $\|w_{\text{old}}\|^2$, we find the increment to be trivially 1. Square-rooting, we again get 1; note that we are still operating under the assumption that the maximum norm of any point x is 1.

References

- A Course in Machine Learning (Chapter 4.5) ~ Hal Daumé III