CS-621 Theory Gems September 26, 2012

Lecture 3

Lecturer: Aleksander Mądry *Scribes: Yann Barbotin and Vlad Ureche*

1 Introduction

The main topic of this lecture is another classical algorithm for finding linear classifiers – the *Winnow Algorithm*. We also briefly discuss *Support Vector Machines* (SVMs).

2 The Winnow Algorithm

Recall the problem of linear classification we discussed in the last lecture. In this problem, we are given a collection of m labeled points $\{(x^j, l^j)\}_j$ with each $x^j \in \mathbb{R}^n$. Our goal is to find a vector (linear classifier) $(w, \theta) \in \mathbb{R}^{n+1}$ such that, for all j,

$$\text{sign}(w \cdot x^j - \theta) = l^j.$$

That is, we want the hyperplane corresponding to (w, θ) to separate the positive examples from the negative ones. As we already argued previously, wlog we can constrain ourselves to the case when $\theta = 0$ (i.e., the hyperplane passes through the origin) and there are only positive examples (i.e., $l^j = 1$, for all j).

Last time we presented a simple algorithm for this problem called *Perceptron* algorithm. Today, we will see a different (and also simple) algorithm for the same task – the *Winnow Algorithm*.

2.1 The Algorithm

The Winnow algorithm can be seen as a direct application of the multiplicative weights update paradigm that we discussed in Lecture 1. To make this connection precise, instead of providing an explicit description of the algorithm, we will cast it as an application of the Multiplicative Weights Update (MWU) algorithm to an appropriately tailored execution of the (general) learning-from-expert framework – cf. Section 5 in the notes from Lecture 1. (As we will see later in the course, there are many, quite diverse, algorithms that can be cast in this manner - this is one reason why the multiplicative weights update paradigm has such a wide array of applications.)

Similarly to the case of the analysis of the Perceptron algorithm, we will wlog assume that our data is normalized. However, this time we will normalize it in l_∞ norm instead of l_2. So, from now on, $\|x^j\|_\infty = \max_i |x_i^j| = 1$, for all j.

Winnow algorithm:
Consider the following execution of the learning-from-expert-advice framework that is interacting with the MWU algorithm (as described in Section 5 in notes from Lecture 1) for $\rho = 1$ and some $0 < \varepsilon \leq \frac{1}{2}$. We have n experts - one for each coordinate (feature). In each round t:

- Let $(p_1^t, \ldots, p_n^t)$ be the convex combination supplied by the MWU algorithm;
- Set w^t to be the linear classifier given by p_i^ts, i.e., $w^t \leftarrow (p_1^t, \ \ldots, \ p_n^t)$;
- Check if there exists x^{j_t} such that $w^t \cdot x^{j_t} \leq 0$ (i.e., x^{j_t} is misclassified);
- If no such j_t exists then we just stop the algorithm and output the classifier w^t (as it classifies all the data correctly);
- Otherwise, we set the loss vector to be $l_i^t = -x_i^{j_t}$, for each i, and proceed to the next round;

Clearly, given that our stopping condition ensures correctness of the output classifier, our only task is to bound the total number of executed rounds (provided an appropriate linear classifier exists).

Theorem 1 *Let w^* be a correct linear classifier such that $|w^*|_1 = 1$, $w_i^* \geq 0$ for all i, and let $\gamma_W = \min_j x^j \cdot w^*$. If $\frac{\gamma_W}{4} \leq \varepsilon \leq \frac{\gamma_W}{2}$ then the number T of the iterations of the Winnow algorithm is at most*

$$T \leq \frac{8 \log n}{\gamma_W^2}.$$

Before we prove this theorem, let us discuss the assumptions that it makes, as well as, provide some intuition underlying the algorithm.

As we already discussed in the last lecture, we can always assume that a correct linear classifier is normalized. On the other hand, to make sure that all coordinates of w^* are non-negative, we can just apply to each x^j a transformation $x^j \to (x^j, -x^j)$. This will double the number of our features, but now, as it is not hard to see, if there existed a correct linear classifier for the original data then there is one for the transformed one that has all its coordinates non-negative and achieves the same margin. So, the assumption made on w^* are not really constraining the applicability of the theorem.

To understand the idea behind the algorithm and its analysis, note that we setup the loss vectors in such a way that in each round t the MWU algorithms suffers a loss proportional to the extent it misclassifies the point x^{j_t}. So, in particular, this loss is always non-negative.

On the other hand, the existence of the correct classifier w^* is a certificate that there exists a convex combination of experts that never suffers a loss that is larger than $-\gamma_W$ (and this quantity is strictly negative). Therefore, by the "asymptotically optimal" loss guarantee of the MWU algorithm, we know that such a poor performance of the MWU algorithm cannot last for too long. (In particular, the convex combination $(p_1^t, \ldots, p_n^t)$ that it maintains will eventually shift its mass towards the features/experts that are most important for correct classification.)

Finally, note that the theorem requires that the MWU algorithm is used with the value of ε being within a factor of two of the value of $\frac{\gamma_W}{2}$. This might be problematic as we usually do not know how large γ_W is. There is, however, a simple technique that allows us to circumvent this problem.

In this technique, we start with $\varepsilon = \frac{1}{2}$ (which is an obvious upper bound on the value of $\frac{\gamma_W}{2}$) and run the Winnow algorithm for $\frac{8 \log n}{\varepsilon^2}$ rounds. If the algorithm terminates by that time, we get a correct classifier and are done. Otherwise - by Theorem 1 - we know that it must have been the case that our value of ε was too large (i.e., $\frac{\gamma_W}{2} < \varepsilon$). Thus we half the value of ε and repeat the procedure (cf. Figure 1).

Clearly, this procedure will finish once ε becomes less that $\frac{\gamma_W}{2}$ (or even earlier), and it is not hard to see that the overall number of iterations performed is at most twice as large as the one promised by Theorem 1 in case of ε being set correctly.

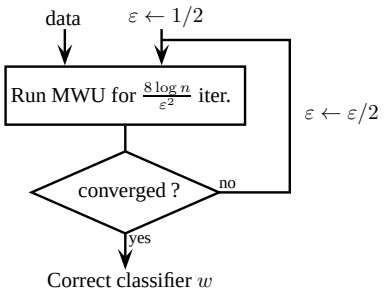

Figure 1: Iterative margin estimation using Theorem 1

We now turn to the proof of the theorem.

Proof

We have that the total loss l_{MWU} of the MWU algorithm is

$$l_{MWU} = \sum_t \sum_i p_i^t l_i^t = -\sum_t \sum_i w_i^t x_i^{j_t} = -\sum_t w^t \cdot x^{j_t} \geq 0,$$

as each x^{j_t} was chosen so as the classifier w^t misclassifies it.

Therefore, by the loss guarantee of MWU algorithm (cf. Theorem 6 in the notes from Lecture 1), we have for any feature/expert i that

$$0 \leq l_{MWU} \leq \sum_t l_i^t + \varepsilon \sum_t |l_i^t| + \frac{\ln n}{\varepsilon} \leq -\sum_t x_i^{j_t} + \varepsilon T + \frac{\ln n}{\varepsilon}, \tag{1}$$

as $\rho = 1$ in our setting (due to normalization of all x^js) and thus $|l_i^t| \leq 1$ for all i and t.

Let us multiply both sides of each equation (1) corresponding to some i, by w_i^* and add them all together. As all w_i^* are non-negative and they sum up to one, we obtain

$$0 \leq \sum_i w_i^* \left(-\sum_t x_i^{j_t} + \varepsilon T + \frac{\ln n}{\varepsilon} \right) = -\sum_t w^* \cdot x^{j_t} + \varepsilon T + \frac{\ln n}{\varepsilon}.$$

Now, by definition of γ_W, we have that $w^* \cdot x^j \geq \gamma_W$ for each j. Therefore, it must be the case that

$$0 \leq -\sum_t w^* \cdot x^{j_t} + \varepsilon T + \frac{\ln n}{\varepsilon} \leq -\gamma_W T + \frac{\gamma_W}{2} T + 4\frac{\ln n}{\gamma_W},$$

as $\frac{\gamma_W}{4} \leq \varepsilon \leq \frac{\gamma_W}{2}$.

Rearranging the terms and dividing both sides by $\frac{\gamma_W}{2}$, we get that

$$T \leq \frac{8 \log n}{\gamma_W^2},$$

as desired. ■

Finally, it is worth pointing out that the MWU algorithm is treated here in a purely black-box fashion. So, it can be replaced by any other algorithm for the learning-from-expert-advice framework, as long as, this algorithm offers a similar loss guarantee to the one described by Theorem 6 from Lecture 1.

2.2 Comparison of the Perceptron and Winnow Algorithms

At this point, we have seen two different algorithms (Perceptron and Winnow) for exactly the same task. It is thus natural to wonder which one of them one should use. As it is often the case, there is no clear answer to that question. The performance guarantees of both these algorithms – although look similar – are to large extent incomparable.

Recall that if we do not normalize the data then the performance guarantees for these algorithms that we have established are:

Algorithm	Number of iterations
Perceptron	$\frac{1}{\gamma_P^2}$ with $\gamma_P = \max_{w^*} \min_j \frac{w^* x^j}{\|\|w^*\|\|_2 \|\|x^j\|\|_2}$
Winnow	$\frac{8 \log n}{\gamma_W^2}$ with $\gamma_W = \max_{w^*} \min_j \frac{w^* x^j}{\|\|w^*\|\|_1 \|\|x^j\|\|_\infty}$

So, if both γ_P and γ_W are similar, using Perceptron should be better. However, in general, γ_P – often called l_2/l_2-margin – and γ_W – often referred to as l_∞/l_1-margin – can be quite different, as the corresponding norms have different characteristics - cf. Figure 2.

In particular, one advantage of Winnow algorithm is that the l_∞/l_1-margin is much less sensitive to presence of irrelevant features in our data set. More precisely, if one imagines that features take values from some fixed interval, say $[-1, 1]$, then adding new features that are not really needed for correct classification (i.e., the optimal classifier does not put any weight on them) does not affect the l_∞/l_1-margin, while still might significantly change the l_2/l_2-margin.

The general consensus is that Winnow algorithm is more preferable when one expects the target classifier to be rather sparse (i.e., a lot of features is irrelevant). On the other hand, if one suspects that most of the features are relevant for correct classification (i.e., the optimal classifier is non-zero on most features) then the Perceptron algorithm should perform better, as the l_2 norm of the target should be much smaller than its l_1 norm.

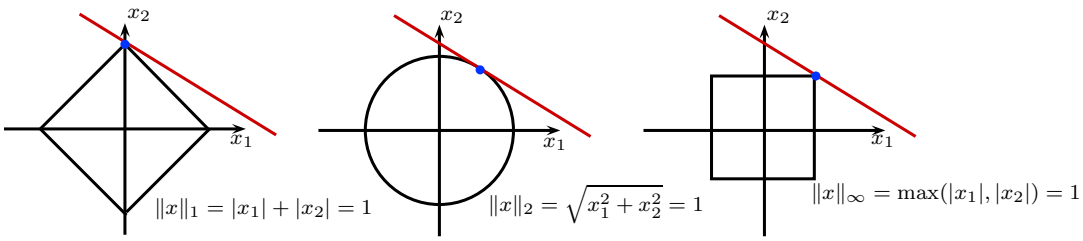

Figure 2: Unit balls in, respectively, ℓ_1, ℓ_2 and ℓ_∞ norm. The intersection with the red line corresponds to the point in these balls that maximizes linear objective $2x_1 + x_2$. The ℓ_1 solution has only one of its component non-zero (sparse solution), while the ℓ_2 norm solution is full and minimizes the energy. The ℓ_∞ solution favors homogeneity.

3 Support Vector Machines (SVMs)

Our general strategy so far was to relate the running time of our algorithms to the quality of the margin achieved by some optimal classifier. In this way we were able to prove that if the data can be separated with large margin then our algorithms return a correct classifier. However, while doing so, we provided no guarantee on the quality of the margin of this returned classifier. This is rather unfortunate, as one would expect (and, in fact, it can be shown) that classifiers with larger margins have better generalization properties. Therefore, ability to return a classifier with good margin (if such exists) would be a very useful property.

To some extent, both Perceptron and Winnow algorithm can be adjusted to provide this kind of guarantees (e.g., by treating all classifications that are correct but have small margin as misclassifications). However, it is sometimes better to take a more principled approach here and just state the search for maximum margin classifier as optimization problem. Such optimization problems are called *Support Vector Machines (SVM)*.

Formally, in the case of l_2/l_2-margin, we have our data normalized in l_2 norm and we want to solve the following problem

$$\begin{aligned} &\min ||w||_2^2 \\ &\text{s.t. } w \cdot x^j \geq 1 \quad \forall_j \end{aligned} \tag{2}$$

(Note that the constraint is ≥ 1 instead of ≥ 0 here.)

As the above program is convex, we can solve it efficiently. (In fact, there are very efficient methods specialized in solving such SVMs.) One can show – see below – that indeed the optimum solution to this program gives a maximum l_2/l_2-margin classifier. Also, one can use duality arguments to show that - similarly to the case of Perceptron and Winnow algorithm - this optimal classifier is a combination of some of the data points x^j. These points are often called *support vector* (as they "support" the separating hyperplane), which explains the name of SVMs.

Lemma 2 *Let w^* be the optimum solution to the SVM (2). The l_2/l_2-margin achieved by w^* is equal to γ_P and $\|w^*\|_2^2 = 1/\gamma_P^2$.*

Proof

First, we show that w^* is a correct classifiers with l_2/l_2-margin being at least $\frac{1}{\|w^*\|_2}$. This follows easily by noting that, for any j,

$$\frac{w^* \cdot x^j}{\|w^*\|_2 \|x^j\|_2} \geq \frac{1}{\|w^*\|_2|},$$

where we used the feasibility of w^* and the fact that all x^js are normalized.

On the other hand, if w is a correct classifier with l_2/l_2-margin γ then $\frac{w}{\gamma\|w\|_2}$ is a feasible solution to the SVM (2). To see this, note that for any j

$$\frac{w \cdot x^j}{\gamma\|w\|_2} = \frac{w \cdot x^j \|w\|_2}{\min_{j'} w \cdot x^{j'} \|w\|_2} \geq 1.$$

Also, the objective value corresponding to $\frac{w}{\gamma\|w\|_2}$ is $1/\gamma^2$. The lemma now follows by combining the two above facts. ■

Although at first glance very appealing, in practice, looking for the maximum margin classifier that correctly classifies all the data might not be the best idea. This is so as the real-world data is usually noisy and just a few outliers might either render the data not linearly separable, or at least significantly reduce the margin of the best linear classifier. Therefore, instead of treating the correct classification of all data points as a hard constraint, one might prefer to be able to allow oneself to misclassify some samples if this enables separation of the remaining data with large margin.

This type of soft trade-off between the correctness and quality of the margin can be very conveniently expressed in SVMs framework, by introducing so-called *slack variables* ε_j for each data point. Formally, one consider the following augmented optimization problem:

$$\min ||w||_2^2 + C \sum_j \varepsilon_j$$

$$\text{s.t. } w \cdot x^j \geq 1 - \varepsilon_j \quad \forall_j$$

$$\varepsilon_j \geq 0 \quad \forall_j$$

Here, the parameter C is used to weigh the size of the margin of the obtained classifier against its classification error measured by the total hinge loss. Clearly, if we set C to be very large, we essentially recover the setting of SVM in (2).