Computational Learning Foundations **Fall semester, 2010**

Lecture 4: November 7, 2010

Lecturer: Yishay Mansour *Scribes: Elad Liebman, Yuval Rochman & Allon Wagner*[1]

4.1 Introduction - Online Learning Model

Imagine a *robot* that needs to classify oranges (objects) as *"export"* (high-quality) or *"local-market"*. We want to do this by letting it work and classify oranges as they arrive (online). After making each of its decisions, an *expert* (i.e., an experienced worker) provides it with the "correct" classification so that if it makes mistakes it will be able to modify its prediction method. One hopes that the robot will converge to a "good" classification method.

In this lecture we study the *online learning* protocol.

Our model:

1. The algorithm receives an unlabeled example x.
2. The algorithm predicts a classification b for this example.
3. The algorithm is then told the correct answer, $c_t(x)$.

In our current setting, this scenario is repeated indefinitely.

Note that this model can be adversarial, i.e., the provided input may be selected in the worst possible way for the learning algorithm.

We will call whatever is used to perform step (2), the algorithm's "current hypothesis" (sometimes also referred to as "concept").

A **mistake** is an incorrect prediction, namely $c_t(x) \neq b$.

The **goal** is to make a (small) bound number of mistakes, which is independent of the size of the input. If we achieve our goal, it is assured that from this point onwards, our hypothesis will classify all observations correctly (otherwise, the adversary can always make it fail again). It is also assured that the algorithm will not switch from one hypothesis to another cyclicly (again, in such a case an adversary would make us fail repeatedly).

[1]Based on notes by Maria-Florina Balcan (Januray 14-21, 2010), as well as previous scribes written by Yishay Mansour (March 10, 1996), Gadi Bareli (January 21, 1993), Guy Gaylord & Nir Pedhazur (April 27, 1994), Noam Neer, Assaf Natanzon & Oded Regev (march 8, 1995).

4.2 Learning Linear Separators

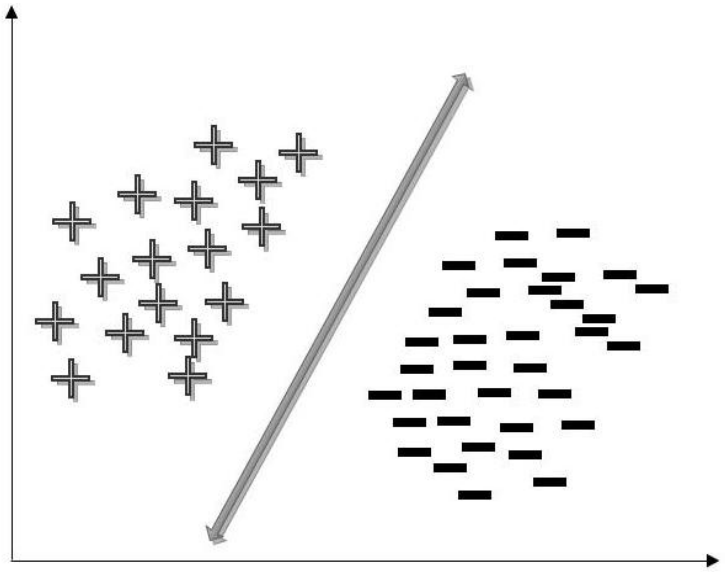

Figure 4.1: A linear separator example

Here we can think of examples as being from $\{0,1\}^n$ or from R^n (the algorithm will not be sensitive to the difference). In the consistency model, the goal is to find w_0 (a threshold) and $\vec{w}$ (a weights vector) defining a hyperplane $\vec{w} \cdot \vec{x} = w_0$ such that all positive examples are on one side and all negative examples are on the other. I.e., $\vec{w} \cdot \vec{x} > w_0$ for positive $\vec{x}$'s and $\vec{w} \cdot \vec{x} < w_0$ for negative $\vec{x}$'s. We can solve this using linear programming.

For simplicity, we'll use a threshold $w_0 = 0$, so we're looking at learning functions like: $\sum_{i=1}^{n} w_i x_i > 0$. 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.

We will begin by discussing the Perceptron algorithm, an online algorithm for learning linear separators, one of the oldest algorithms used in machine learning (from the early 60s).

4.2.1 The Perceptron Algorithm

The main idea of this algorithm is that as long as we do not make a mistake, we remain with the same separator. When we do make a mistake - we move the separator towards it.

We shall automatically scale all examples $\mathbf{x}$ to have Euclidean length 1 (i.e. $\|x\|_2 = 1$), since this doesn't affect which side of the plane they are on.

The Perceptron Algorithm:

1. Start with the all-zeroes weight vector $\mathbf{w}_1 = \mathbf{0}$, and initialize t to 1.
2. Given example $\mathbf{x}_t$, predict positive iff $\mathbf{w}_t \cdot \mathbf{x}_t > 0$.
3. On a mistake, update as follows:
   - Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \mathbf{x}_t$.
   - Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \mathbf{x}_t$.

The intuition: suppose we encounter $\mathbf{x}$, a positive example (we denote $\mathbf{x}_t$ as simply $\mathbf{x}$ for simplicity). If we make a mistake classifying $\mathbf{x}$, then after the update it follows that

$$\mathbf{w}_{t+1} \cdot \mathbf{x} = (\mathbf{w}_t + \mathbf{x}) \cdot \mathbf{x} = \mathbf{w}_t \cdot \mathbf{x} + \mathbf{x} \cdot \mathbf{x} = \mathbf{w}_t \cdot \mathbf{x} + 1$$

The example was positive but (since we've made a mistake) the dot product was negative. Therefore, we have increased it in the right direction. 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.

Another way of perceiving this is that the dot product is the cosine of the angle between $\mathbf{x}$ and $\mathbf{w}$ - when we make a mistake, we change the angle (we will refer to this later).

Theorem 4.1. *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). γ represents the minimal distance of each example $\mathbf{x}$ from $\mathbf{w}^*$, after normalizing both $\mathbf{w}^*$ and the examples. Another way to view the quantity $\frac{\mathbf{w}^* \cdot \mathbf{x}}{||\mathbf{x}||}$ 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.

Proof of Theorem 4.1. We are going to look at the following two quantities $\mathbf{w}_t \cdot \mathbf{w}^*$ and $||\mathbf{w}_t||$.

We may assume WLOG that we only make mistakes, for the simple reason that if we do not make a mistake no update follows.

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, then a vector maximizing the dot product $\mathbf{w}_t \cdot \mathbf{w}^*$ would be $\frac{\mathbf{w}_t}{||\mathbf{w}_T||}$, which entails that

$$\mathbf{w}_t \cdot \mathbf{w}^* \leq \mathbf{w}_t \cdot \frac{\mathbf{w}_t}{||\mathbf{w}_t||} = \frac{||\mathbf{w}_t||^2}{||w_t||} = ||w_t||$$

All in all, we get

$$\gamma M \leq w_{M+1} \cdot w^* \leq ||\mathbf{w}_t|| \leq \sqrt{M}$$

and thus $M \leq 1/\gamma^2$. □

What if there is no perfect separator? What if only *most* of the data is separable by a large margin (as seen on Figure 4.2), 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^*. The hinge loss can also be defined as $max(0, 1 - y)$, where $y = \frac{l(x) \cdot \mathbf{x} \cdot \mathbf{w}^*}{\gamma}$ and $l(x)$ is the classification of $\mathbf{x}$. The hinge loss is a loss function that begins paying linearly as it approaches the hyperplane (see Figure 4.3).

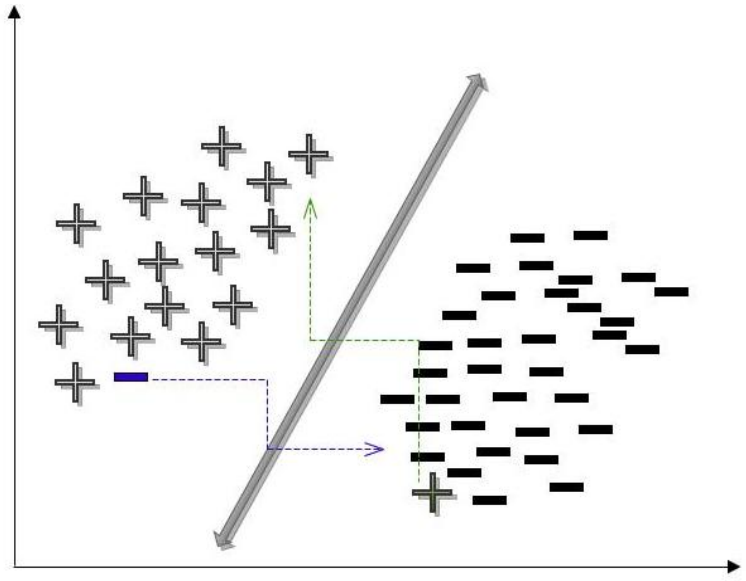

Figure 4.2: A a case in which no perfect separator exists

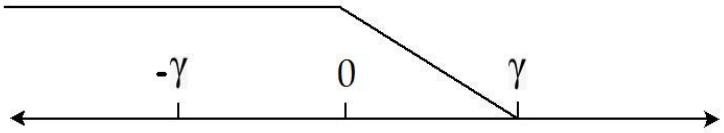

Figure 4.3: Illustrating hinge loss

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.

4.2.2 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 $\mathbf{x}$).

There can be many possible separating hyperplanes, however - some are much better than

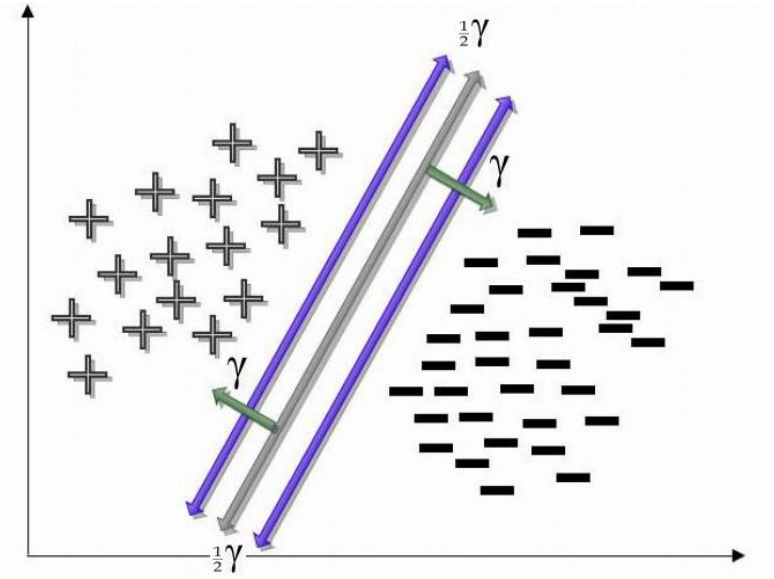

Figure 4.4: Attempting to obtain a good separator by defining a "penalty margin"

others. 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 $\gamma/2$ (see illustration in Figure 4.4). 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 4.2. *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, 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 $12/\gamma^2$.

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

As before, we can show that 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 $||\mathbf{w}_t||$. For the original algorithm, we had: $||\mathbf{w}_{t+1}||^2 \leq ||\mathbf{w}_t||^2 + 1$. Using a similar method to the one we use below, we could have actually shown $||\mathbf{w}_{t+1}|| \leq ||\mathbf{w}_t|| + \frac{1}{2||\mathbf{w}_t||}$.

For the new algorithm, we can show instead:

$$||\mathbf{w}_{t+1}|| \leq ||\mathbf{w}_t|| + \frac{1}{2||\mathbf{w}_t||} + \frac{\gamma}{2}. \tag{4.1}$$

To see this note that:

$$||\mathbf{w}_{t+1}||^2 = ||\mathbf{w}_t||^2 + 2l(x)\mathbf{w}_t \cdot \mathbf{x} + ||\mathbf{x}||^2 = ||\mathbf{w}_t||^2 \left(1 + \frac{2l(x)}{||\mathbf{w}_t||} \frac{\mathbf{w}_t \cdot \mathbf{x}}{||\mathbf{w}_t||} + \frac{1}{||\mathbf{w}_t||^2}\right)$$

Using the inequality $\sqrt{1+\alpha} \leq 1 + \frac{\alpha}{2}$ together with the fact $\frac{l(x)\mathbf{w}_t \cdot \mathbf{x}}{||\mathbf{w}_t||} \leq \frac{\gamma}{2}$ (since w_t made a mistake on x) we get the desired upper bound on $||\mathbf{w}_{t+1}||$, namely:

$$||\mathbf{w}_{t+1}|| = ||\mathbf{w}_t|| \sqrt{1 + \frac{2l(x)}{||\mathbf{w}_t||} \frac{\mathbf{w}_t \mathbf{x}}{||\mathbf{w}_t||} + \frac{1}{||\mathbf{w}_t||^2}} \leq ||\mathbf{w}_t|| \sqrt{1 + \frac{2l(x)}{||w_t||} \frac{\gamma}{2} + \frac{1}{||\mathbf{w}_t||^2}}$$

$$\leq ||\mathbf{w}_t|| (1 + \frac{\frac{\gamma}{||\mathbf{w}_t||} + \frac{1}{||\mathbf{w}_t||^2}}{2}) \Rightarrow ||\mathbf{w}_{t+1}|| \leq ||\mathbf{w}_t|| + \frac{1}{2||\mathbf{w}_t||} + \frac{\gamma}{2}$$

(notice we selected $(\frac{\gamma}{||w_t||} + \frac{1}{||w_t||^2})$ as α, and that $l(x) \leq 1$)

Note that (4.1) implies that if $||\mathbf{w}_t|| \geq 2/\gamma$ then $||\mathbf{w}_{t+1}|| \leq ||\mathbf{w}_t|| + 3\gamma/4$. Given that, it is easy to see that after M updates we have:

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

As before, $\gamma M \leq ||w_{M+1}||$. Solving $M\gamma \leq 1 + 2/\gamma + 3M\gamma/4$ we get $M \leq 12/\gamma^2$, as desired. ■

Comment: We will see later on why it is preferable to take an algorithm with a large margin. Here we saw how the Perceptron algorithm can be modified so that its result approaches the best possible margin. We accomplished this using $\gamma/2$, but we might as well have chosen $(1-\epsilon)\gamma$.

4.3 The Mistake Bound model

In the *mistake bound model*:

- x - examples
- c - the target function, $c_t \in C$
- $x_1, x_2 \ldots x_t$ an input series
- *at the t^{th} stage:*
  1. The algorithm receives x_t
  2. The algorithm predicts a classification for x_t, b_t
  3. The algorithm receives the true classification, $c_t(x)$.
- A mistake occurs if $c_t(x_t) \neq b_t$

Definition 1. *A hypotheses class $\mathcal{C}$ has an algorithm A with mistake M if for any concept $c \in \mathcal{C}$, and for any ordering of examples, the total number of mistakes ever made by A is bounded by M.*

We shall assume $\mathcal{C}$ is a finite class.

4.3.1 A simple algorithm - CON

Let $\mathcal{C}$ be a finite concept class. Consider the following algorithm: in stage t the algorithm examines the set C_t of all concepts in $\mathcal{C}$ which are consistent with all the examples it has seen so far, $x_1, \ldots, x_{t-1}$. It chooses arbitrarily $c_t \in C_t$ and uses it in order to predict a classification for x_t. (In fact this is a family of algorithms and not a single one.)

Analysis:

- Claim[No Regression]: $C_{t+1} \subseteq C_t$. Proof: this is trivial, since any concept which is not consistent at stage t is certainly not consistent at stage $t+1$.
- Claim[Progress]: If a mistake is made at stage t then $|C_{t+1}| < |C_t|$ (Hence, each mistake decreases the number of consistent concepts). Proof: If we make a mistake this means that c_t which is in C_t is not consistent with the classification of x_t and hence does not belong to C_{t+1}.
- Claim[Termination]: $|C_t| \geq 1$, since $c_t \in C_t$ at any stage t.

Theorem 4.3. *For any concept class $\mathcal{C}$, the algorithm CON makes at most $|\mathcal{C}|-1$ mistakes.*

Proof: We start with $C = C_1$. After each mistake C_t decreases by at least one item. Considering that for every t, $|C_t| \geq 1$, it follows that there are at most $|C| - 1$ mistakes. ■

The problem: $|\mathcal{C}|$ may be very large! Note that the number of boolean functions on $\{0,1\}^n$ is 2^{2^n}. The following algorithm will guarantee a much better performance.

4.3.2 The Halving Algorithm - HAL

Let $\mathcal{C}$ be any concept class. Consider the following algorithm *HAL*:

In stage t the algorithm considers the set C_t of all concepts consistent with all the $(t-1)$ examples it has seen so far. For each $c \in C_t$ it considers the value of $c(x_t)$. Its prediction $b_t(x_t)$ is the *majority* of these values. I.e. let $zero_t = \{c \in C_t : c(x_t) = 0\}$ and $one_t = \{c \in C_t : c(x_t) = 1\}$, if $zero_t > one_t$ then $b_t(x_t) = 0$, otherwise $b_t(x_t) = 1$.

Analysis:

- Claim[No regression]: As before.
- Claim[Progress]: If a mistake is made at stage t then $|C_{t+1}| \leq \frac{1}{2}|C_t|$. (Hence, each mistake decreases the number of consistent concepts significantly). Proof: If we make a mistake at stage t this means that the *majority* of the concepts in C_t are not consistent with the true classification of x_t.
- Therefore, for any concept c and sequence s,

$$M_{HAL}(c,s) \leq \lfloor \log_2 |\mathcal{C}| \rfloor \Rightarrow M_{HAL}(\mathcal{C}) \leq \lfloor \log_2 |\mathcal{C}| \rfloor.$$

Conclusion 1. *For any concept class $\mathcal{C}$ the algorithm HAL makes at most $\lfloor \log_2 |\mathcal{C}| \rfloor$ mistakes.*

We will now show that if there is an online algorithm with a bound number of mistakes, we can translate it to a PAC algorithm.

4.4 The Relation between the Mistake Bound and PAC models

It seems that the mistake bound model generates strong online algorithms that make mistakes with low probability.

In the last lecture we have seen the PAC model, which is another learning model, whose algorithms have low probability to make a mistake. It seems that the requirements for a mistake bound algorithms are much stronger than for a PAC algorithm. Therefore we ask : If we know that $\mathcal{A}$ learns some concept class C in the mistake bound model, Should $\mathcal{A}$ learn C in the PAC model?

Well, $\mathcal{A}$ itself cannot learn PAC, since every algorithm learning PAC must have also ϵ and the δ parameters. But we will show how to construct such an algorithm $\mathcal{A}_{PAC}$ which will be in PAC.

In order to define the $\mathcal{A}_{PAC}$ algorithm, first, let us assume that after algorithm $\mathcal{A}$ gets x_i, it constructs a hypothesis h_i. Second, let us define the notion of a "Conservative algorithm" (sometimes also referred to as "lazy").

Definition 2. *A mistake bound algorithm $\mathcal{A}$ is **conservative** iff for every sample x_i if $c_t(x_i) = h_{i-1}(x_i)$, then in the i-th step the algorithm will make a choice $h_i = h_{i-1}$*

In other words, a conservative algorithm will change its hypothesis only if $\mathcal{A}$ has made a mistake on an input sample.

Although not all the algorithms in the mistake bound model are conservative, each algorithm $\mathcal{A}$ in the mistake bound model has its conservative dual algorithm $\mathcal{A}'$: it will run like $\mathcal{A}$, but every time it is getting x_i, then it will see if $c_t(x_i) = h_{i-1}(x_i)$. If so, it won't make any changes, and will skip to the next sample. Otherwise, the algorithm acts like $\mathcal{A}$. We claim that:

Theorem 4.4. *$\mathcal{A}'$ learns a concept class C in the mistake bound model, with the same mistake bound of $\mathcal{A}$*

Proof. If there is a sample $\{x_1, x_2, ..., x_n\}$ for which $\mathcal{A}'$ has a mistake bigger than the mistake bound of $\mathcal{A}$, then running $\mathcal{A}$ over the sample $\{x_i | c_t(x_i) \neq h_{i-1}(x_i)\}$ will be equivalent to running $\mathcal{A}'$ on $\{x_1, x_2, ..., x_n\}$,hence it will be bigger than the mistake bound of $\mathcal{A}$- a contradiction □

Now, if $\mathcal{A}$ has a mistake bound M, we will define $k_i = \frac{i}{\epsilon} \ln(\frac{M}{\delta})$ for $0 \leq i \leq M-1$.

The algorithm $\mathcal{A}_{PAC}$ will perform the following procedure:

Run $\mathcal{A}'$ over a sample size of $M\frac{1}{\epsilon}\ln(\frac{M}{\delta})$, divided into M equal (and independent) blocks. Each time we construct a hypothesis h_{k_i}, $0 \leq i \leq M-1$ (The hypothesis after k_i samples), run the hypothesis on the next block of $\frac{1}{\epsilon}\ln(\frac{M}{\delta})$ samples. If there isn't any mistake, give the current hypothesis as output, and terminate the algorithm. Else, continue running $\mathcal{A}'$

We are guaranteed that $\mathcal{A}'$ makes at most M mistakes. Therefore, we if give it M samples on which it errs, it will produce a hypothesis which is a perfect classifier, and trivially PAC demands are satisfied. However, the sample might contain cases for which the $\mathcal{A}'$ does not err. Our main theorem will show that the procedure described above assures PAC learning:

Theorem 4.5. *$\mathcal{A}_{PAC}$ will produce some hypothesis, and $\mathcal{A}_{PAC}$ learns PAC*

Proof. We begin by noting that since the number of mistakes made is bound by M, the algorithm will terminate after the last block, outputing its final hypothesis h_{M-1}. (Further, note that h_{M-1} was produced after making M errors, thus it's a perfect classifier). Thus, the algorithm always produces an output.

We notice that for every h_{k_i}, the probability of a "bad event" - i.e., halting on h_{k_i} whereas h_{k_i} is $\epsilon - bad$, is as follows:

$$\Pr(h_{k_i} \textit{ succeeds on a block while being } \epsilon\textit{-bad}) = (1-\epsilon)^{\frac{1}{\epsilon}\ln(\frac{M}{\delta})} \leq e^{-\ln(\frac{M}{\delta})} \leq \frac{\delta}{M}$$

Therefore,

$$\Pr(\mathcal{A}_{PAC} \text{ outputs an } \epsilon - \text{bad hypothesis } h) \leq \Pr\left(\exists 0 \leq i \leq M-1 \text{ s.t } h_{k_i} \text{ is } \epsilon\text{-bad }\right) \leq$$

$$\sum_{i=0}^{M-1} \Pr(h_{k_i} \text{ is } \epsilon\text{-bad }) \leq \sum_{i=0}^{M-1} \frac{\delta}{M} = \delta,$$

which means $\mathcal{A}_{PAC}$ learns in the PAC model. □

4.5 Disjunction of Conjunctions

4.5.1 Algorithm for Disjunction

Having seen that every algorithm in the mistake bound model can be converted to an algorithm in PAC, lets see a mistake bound algorithm for the disjunction of conjunctions:

1. Initialize hypothesis set L to be the set of $\{x_1, \overline{x_1}, x_2, \overline{x_2}, \ldots x_n, \overline{x_n}\}$.

2. Given a sample $\vec{y} = (y_1, y_2, \ldots y_n)$, where $y_i \in \{0, 1\}$, do:

   (a) Guess $h(\vec{y})$ where $h_t(\bar{x}) = \bigvee_{x_i \in L} x_i$

   (b) If $h(\vec{y}) = c_t(\vec{y})$ then return to step 2,

   (c) Otherwise, define $S_y = \{x |\ x$ is a literal s.t the value it represnts in $\vec{y}$ is positive$\}$ and make $L \leftarrow L \setminus S_y$.

   (d) Return to step 2

The algorithm changes its hypothesis only when we have a mistake (it is conservative). If we were mistaken, then we change our hypothesis.

For example, if we have only 2 variables, (e.g. $L = \{x_1, \overline{x_1}, x_2, \overline{x_2}\}$) and the first sample is $\bar{y} = (1, 0)$ with the target function $c_t(y) = 0$, then after one iteration of step 2, we get $L = \{\overline{x_1}, x_2\}$. But, if $c_t(y) = 1$, We didn't get a mistake. and we didn't change the hypothesis.

Theorem 4.6. *The algorithm only makes a mistake when $c_t(y) = 0$, and the number of mistakes is bound by $n + 1$ where n is the number of variables*

From this theorem we get that the above algorithm is a mistake bound algorithm

Proof. Let R be the set of all literals in $c_t(y)$, and L_i the hypothesis after getting the i^{th} sample, $\overline{y}_i$.

If we prove by induction, that for all i, $R \subseteq L_i$, then after get the $(i+1)^{\text{th}}$ sample, if $c_t(y) = 1$ then it must be $h_i(\overline{y}) = 1$ and therefore, we only make a mistake when $c_t(y) = 0$.

The induction is easy. It is obvious that $R \subseteq L_0$. Then in the i^{th} sample, if $c_t(y) = 1$ we don't make any change, because $c_t(y) = h_i(\overline{y})$. If $c_t(y) = 0$ then of course S_y and R don't intersect. Hence we get $R \subseteq L_{i+1}$.

In the first mistake, when we are eliminating exactly n literals from L. In the other mistakes, we are eliminating from L at least one literal. L_0 contain $2n$ literals, therefore the number of mistakes is $n + 1$.

□

4.5.2 k-DNF

Let k be a constant. Consider the class of k-DNF functions. That is, functions that can be represented by a disjunction of terms, where each of these terms is a conjunction of at most k literals. The number of the conjunctions of size i terms is $\binom{n}{i}2^i$ - choosing first the i variables, and then choosing for every variable whether it or its negative will be in the term. Hence $t = \sum_{i=0}^{k}\binom{n}{i}2^i = \Theta(n^k)$ (i.e., polynomial in n) is the maximum number of terms of a k-DNF.

We can learn this class in a similar way, either by changing the above algorithm to deal with terms instead of variables, or by reducing the space $X = \{0,1\}^n$ to $Y = \{0,1\}^{n^k}$ by computing for each of the terms its value. The original function on X gives a disjunction on Y and hence can be learned by the ELIM algorithm (for a PAC algorithm) or to use the previous algorithm (for the Mistake bound algorithm) as it is with $O(n^k)$ mistakes.

4.6 The Winnow Algorithm

We now turn to an algorithm called the *Winnow algorithm* . Like the Perceptron procedure discussed previously, the Winnow algorithm learns what is the linear separator. Unlike the Perceptron Procedure, Winnow uses *multiplicative* rather than *additive* updates.

The Winnow algorithm is meant for learning the class of monotone disjunctions (of n variables) in the mistake bound model (monotone disjunctions being disjunctions containing only positive literals). This algorithm preforms much better than the greedy algorithm presented in section 2.1, when the number of terms in the target function, r, is $o(\frac{n}{logn})$. Note that Winnow or generalizations of Winnow can handle other specific concept classes (e.g. non-monotone disjunctions, majority functions, linear separators), but the analysis is simplest in the case of monotone disjunctions.

4.6.1 The Algorithm

Both the Winnow and the Perceptron algorithms use the same classification scheme:

- $h(\overrightarrow{x}) \triangleq \overrightarrow{x} \cdot \overrightarrow{w} \geq \theta \Rightarrow$ positive classification
- $h(\overrightarrow{x}) \triangleq \overrightarrow{x} \cdot \overrightarrow{w} < \theta \Rightarrow$ negative classification

For convenience, we assume that $\theta = n$ (the number of variables) and we initialize $\overrightarrow{w_0} = (1, 1, 1 \ldots 1, 1)$ (Note that the Perceptron algorithm used a threshold of 0 but here we use a

threshold of n.)

The Winnow Algorithm differs from the Perceptron Algorithm in its update scheme. When misclassifying a positive training example (e.g, $h(\overrightarrow{x}) = 0$ but $c_t(\overrightarrow{x}) = 1$) then make:

$$\forall x_i = 1 : w_i \leftarrow 2w_i.$$

When misclassifying a negative training example (e.g $h(\overrightarrow{x}) = 1$ but $c_t(\overrightarrow{x}) = 0$) then make:

$$\forall x_i = 1 : w_i \leftarrow w_i/2.$$

In those two cases, if $x_i = 0$ then we don't change w_i.

Similarly to the Precetron algorithm, if the margin is bigger then γ then we can prove that the error rate is $\Theta(\frac{1}{\gamma^2})$.

Notice that because we are updating multiplicatively, all weights remain positive .

4.6.2 The Mistake Bound Analysis

Now, we will prove that Winnow Algorithm is in the mistake bound model.

Theorem 4.7. *The Winnow algorithm learns the class of monotone disjunctions in the mistake bound model, making at most* $2 + 3r(1 + \log n)$ *mistakes when the target concept is an OR of* r *variables.*

Proof. Let $S = \{x_{i_1}, x_{i_2}, \ldots, x_{i_r}\}$ be the r relevant variables in our target concept (e.g $c_t(\overrightarrow{x}) = x_{i_1} \vee x_{i_2} \ldots \vee x_{i_r}$). Let $W_r = \{w_{i_1}, w_{i_2}, \ldots, w_{i_r}\}$ be the weights of those relevant variables (called "relevant variables"). Let $w(t)$ denote the value of weight w at time t and let $TW(t)$ be the Total Weight of the $w(t)$ (including both relevant and irrelevant variables) at time t.

We will first bound the number of mistakes that will be made on positive examples. Note first that any mistake made on a positive example must double at least one of the relevant weights (If all of the relevant weights aren't be doubled then $c_t(\overrightarrow{x}) = 0$). So if at time t we misclassified a positive example we have:

$$\exists w \in W_r \text{ such that } w(t+1) = 2w(t) \tag{4.2}$$

Moreover, a mistake made on a negative example won't change any of the relevant weights. Hence, for all times t, we have:

$$\forall w \in W_r, w(t+1) \geq w(t) \tag{4.3}$$

Each of these weights can be doubled at most $1 + \log(n)$ times, since only weights that are less than n can ever be doubled (If $w_i \geq n$ we will get that $\overrightarrow{x} \cdot \overrightarrow{w} \geq n$ therefore a positive classification). Combining this together with (4.2) and (4.3), we get that Winnow makes at most $M_+ \leq r(1 + \log(n))$ mistakes on positive examples.

We now bound the number of mistakes made on negative examples. For a positive example we must have $h(\overrightarrow{x}) = 0$, therefore:

$$h(\overrightarrow{x}) = w_1(t)x_1 + ... + w_n(t)x_n < n$$

. Since

$$TW(t+1) = TW(t) + (w_1(t)x_1 + ... + w_n(t)x_n),$$

we get

$$TW(t+1) < TW(t) + n. \tag{4.4}$$

Similarly, we can show that each mistake made on a negative example decreases the total weight by at least $n/2$. To see this assume that we made a mistake on the negative example x at time t. We must have:

$$w_1(t)x_1 + ... + w_n(t)x_n \geq n.$$

Since

$$TW(t+1) = TW(t) - (w_1(t)x_1 + ... + w_n(t)x_n)/2,$$

we get

$$TW(t+1) \leq TW(t) - n/2. \tag{4.5}$$

Finally, the total weight in every time does not drop below zero, i.e :

$$TW(t) > 0 \tag{4.6}$$

Combining equations (4.5), (4.4), and (4.6) we get:

$$0 < TW(t) \leq TW(0) + nM_+ - (n/2)M_- \tag{4.7}$$

The total weight summed over all the variables is initially n since $\overrightarrow{w_0} = (1, 1, 1 \ldots 1, 1)$. Solving (4.7) we get:

$$M_- < 2 + 2M_+ \leq 2 + 2r(\log n + 1).$$

Hence,

$$M_- + M_+ \leq 2 + 3r(\log n + 1),$$

as required. □