Introduction to Machine Learning **Fall Semester 2016/7**

Lecture 4, November 20

Lecturer: Yishay Mansour *Scribe: Yishay Mansour*

4.1 Online Learning Model

Imagine a *robot* that needs to classify oranges (objects) as *"export"* (high-quality) or *"local-market"* (low-quality). 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 which abstracts this setting.
Online model: We have discrete time steps. At time t the following occur:

1. The algorithm receives an unlabeled example $\mathbf{x}_t$.

2. The algorithm predicts a classification $\hat{y}_t$ for $\mathbf{x}_t$. The prediction function h_t in stage t is called the "current hypothesis". I.e., $\hat{y}_t = h_t(\mathbf{x}_t)$.

3. The algorithm is then told the correct answer, $c^*(\mathbf{x}_t) \in \{-1, +1\}$. (In this lecture we will switch the target function from c_t to c^* since we will use t to denote time.)

In the online model, the number of time steps is usually unbounded. Note that the online model is adversarial, i.e., we assume that the provided input is selected in the worst possible way for the learning algorithm. Namely, we do not assume any stochastic assumption regarding how the inputs are generated.

We will call h_t (which is used to perform step (2) to generate the prediction), the algorithm's "current hypothesis" (sometimes also referred to as "concept"). A **mistake** is an incorrect prediction, namely $c^*(\mathbf{x}_t) \neq h_t(\mathbf{x}_t)$. The **goal** is to make a (small) bounded number of mistakes, which is independent of the number of time steps. If we achieve our goal, and we have already made the maximum number of mistakes, then from this time onwards our hypothesis will classify all observations correctly (otherwise, the adversary can cause the algorithm more mistakes than the bound). Similarly, the online algorithm will not cycle through hypotheses (otherwise the adversary would be able to force an infinite number of errors).

4.1.1 A simple algorithm - CON

Let $\mathcal{C}$ be a finite concept class. Consider the following algorithm. CON. At time t let the set C_t be all the concepts in $\mathcal{C}$ which are consistent with all the examples seen so far, i.e., $\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}$. Algorithm CON chooses arbitrarily $h_t \in C_t$ and uses it in order to predict a classification for $\mathbf{x}_t$. (In fact this is a family of algorithms and not a single one.) For the analysis we use:

- Claim[No Regression]: $C_{t+1} \subseteq C_t$. This holds, 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 in time t then $|C_{t+1}| \leq |C_t| - 1$. A mistake at time t decreases the number of consistent concepts, since $h_t \in C_t$ and h_t not consistent with $\mathbf{x}_t$.
- Claim[Termination]: $|C_t| \geq 1$, since $c^* \in C_t$ at any stage t.

We can now show:

Theorem 4.1 *For any concept class $\mathcal{C}$, the algorithm CON makes at most $|\mathcal{C}| - 1$ mistakes.*

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

The problem with the theorem is that $|\mathcal{C}|$ may be exponential in the natural parameters.

4.1.2 The Halving Algorithm - HAL

Let $\mathcal{C}$ be a finite concept class. Consider the following algorithm HAL. In time t the HAL considers the set C_t of all concepts consistent with all the $t-1$ examples seen so far. For each $c \in C_t$ it considers the value of $c(\mathbf{x}_t)$. Its prediction $\hat{y}_t$ 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 $\hat{y}_t = 0$, otherwise $\hat{y}_t = 1$. For the analysis we make the following claims:

- Claim[No regression]: As before.
- Claim[Progress]: If a mistake is made at time t then $|C_{t+1}| \leq \frac{1}{2}|C_t|$. Each mistake means that the *majority* of the concepts in C_t are not consistent with the true classification of $\mathbf{x}_t$, and the number of consistent concepts is cut in half.
- Claim[Termination]: $|C_t| \geq 1$, since $c^* \in C_t$ at any stage t.

Since every mistake makes cuts the number of consistent concepts by half, we have:

Theorem 4.2 *For any concept class $\mathcal{C}$ algorithm HAL makes at most $\lfloor \log_2 |\mathcal{C}| \rfloor$ mistakes.*

Note that the algorithm HAL is not efficient, each prediction requires $O(|\mathcal{C}|)$ evaluations and computation time.

4.1.3 Example - Learning Boolean Disjunctions

Consider the example of learning Boolean disjunction functions. Recall that the problem is defined as follows: Given a set of boolean variables $T = \{z_1, ..., z_n\}$ and a set of literals $L = \{z_1, \bar{z}_1, \ldots, z_n, \bar{z}_n\}$. we need to learn an OR function over the literals, for example: $z_1 \vee \bar{z}_3 \vee z_5$. Let $\mathcal{C}_{or}$ will be the set of all possible disjunctions. We will assume the realizable case, i.e., $\mathcal{H} = \mathcal{C}$.

Using HAL we get a mistake bound of $\lfloor \log_2 3^n \rfloor = O(n)$. We will show that ELIM is both efficient and has a sightly lower mistake bound.

ELIM algorithm for online learning Boolean disjunctions

Recall that we used the algorithm ELIM to PAC learn the class $\mathcal{C}_{or}$. We will show that we can use it to learn in the online model with a bounded number of mistakes.

Recall that algorithm ELIM, is initialized with the set $L_1 = L$. At time t, ELIM has a set L_t and predicts 1 iff one of the literals in L_t is true, i.e., it predicts using the disjunction $\vee_{z \in L_t} z$. We will show that every time ELIM predicts 0 then the target function is also 0 (and ELIM does not make an error). This implies that ELIM makes errors only on negative examples (when ELIM predicts positive). Every time we are given a negative example ELIM eliminates all literals that evaluate to 1.

Assume the target disjunction has $\mathcal{L}$ as its set of literals. We claim that $\mathcal{L} \subset L_t$. This clearly holds initially. Each time we delete a literal from L_t it is clearly not in $\mathcal{L}$ (since it is positive when the target function is negative). Therefore, each time ELIM predicts negative we are correct.

We start with a set of $2n$ literals. In the first mistake, which is the first negative example, algorithm ELIM eliminates n literals. Each following mistake it eliminates at least one literal. Therefore the number of mistakes is at most $n + 1$.

Theorem 4.3 *Algorithm ELIM makes at most $n + 1$ mistakes when $c^* \in \mathcal{C}_{or}$.*

4.1.4 Regret Minimization

What happens if we are in the non-realizable setting? In this case, there is a literature which is referred to as*regret minimization*. The highlight of the result, is that we can guarantee,

$$Loss(online) \leq \min_{c \in \mathcal{C}} Loss(c) + \sqrt{\frac{\log |C|}{T}}.$$

The regret is $REGRET = Loss(online) - \min_{c \in \mathcal{C}} Loss(c)$. Notice that the average regret vanishes as T increases.

The topic of regret minimization material is (unfortunately) not part on this introductory class.

4.1.5 Online Mistake Bound and PAC models

We have two models, Mistake Bound and PAC, are they equivalent? Namely, is any class learnable in one is also learnable in the other?

Unfortunately the answer is no. Consider the class $\mathcal{C}_{pre}$ of prefix of an interval, namely includes $c_\theta(x) = I(x \geq \theta)$ for $x, \theta \in [0,1]$. We saw that this class is PAC learnable (and also has VC dimension 1). We can show that this class does not have a finite mistake bound algorithm. (Every time t we select a point x_t in the uncertainty region and label it randomly. The expected number of errors after T steps is $T/2$ and goes to infinity when T goes to infinity.)

What about the other direction. Namely, if a class is finite Mistake Bound learnable it is PAC learnable. We will show that this is true. Our task is the following. We are given an online $\mathcal{A}$ that learns a concept class $\mathcal{C}$ and makes at most M mistakes. We like to design an algorithm that PAC learns $\mathcal{C}$. One natural strategy is to take a large sample and run $\mathcal{A}$ on it, and return its last hypothesis. The problem is that we are not guaranteed that $\mathcal{A}$ has a low error at the end of the sequence. We will define a new algorithm $\mathcal{A}_{PAC}$, based on $\mathcal{A}$, which PAC learns $\mathcal{C}$.

Before we define the $\mathcal{A}_{PAC}$ algorithm, we define the notion of a conservative algorithm, an algorithm that that does not change its hypothesis unless it makes an error. (This is sometimes also referred to as "lazy".) Recall that h_t is the hypothesis of $\mathcal{A}$ at time t.

Definition 4.4 *A mistake bound algorithm $\mathcal{A}$ is **conservative** if for every time t such that $c^*(\mathbf{x}_t) = h_t(\mathbf{x}_t)$, then $h_{t+1} = h_t$.*

Although not all online algorithms in the mistake bound model are conservative, each algorithm $\mathcal{A}$ in the mistake bound model has its conservative dual algorithm $\mathcal{A}'$. Algorithm $\mathcal{A}'$ will runs as follows. It keeps a sequence of examples ER which is initially empty. At time t it receives $\mathbf{x}_t$, and predicts $h_t(\mathbf{x}_t)$. Then it observes the true label and tests whether $c^*(\mathbf{x}_t) = h_t(\mathbf{x}_t)$. If so, it sets $h_{t+1} = h_t$ (keeps the same hypothesis). Otherwise, it appends $\mathbf{x}_t$ to ER and runs $\mathcal{A}$ or ER to get h_{t+1}. By design algorithm $\mathcal{A}'$ is conservative.

Let $M_A(C)$ be the maximum number of mistakes that an algorithm A makes when $c^* \in C$. We show that in the worse case the two algorithms have the same mistake bound.

Theorem 4.5 *Algorithm $\mathcal{A}$ learns a concept class $\mathcal{C}$ in the mistake bound model with the minimal number of mistakes, and $\mathcal{A}'$ be its dual conservative algorithm. Then,*

$$M_{\mathcal{A}}(\mathcal{C}) = M_{\mathcal{A}'}(\mathcal{C})$$

Proof: For contradiction assume that $M_{\mathcal{A}}(\mathcal{C}) < M_{\mathcal{A}'}(\mathcal{C})$. Then there is a target function $c^* \in \mathcal{C}$ and an example sequence $e = \langle \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T \rangle$ for which $\mathcal{A}'$ makes $M > M_{\mathcal{A}}(\mathcal{C})$ mistakes. Consider running $\mathcal{A}$ on ER which is generated in the run of $\mathcal{A}'$ on e. By design $\mathcal{A}$ makes a mistake on each example in ER and therefore makes M mistakes. This implies that $M \leq M_{\mathcal{A}}(\mathcal{C})$. A contradiction to the assumption that $M > M_{\mathcal{A}}(\mathcal{C})$. $\square$

We now design the algorithm $\mathcal{A}_{PAC}$ based on $\mathcal{A}$ to PAC learn $\mathcal{C}$. We will assume that $\mathcal{A}$ is conservative. The algorithm $\mathcal{A}_{PAC}$ will be composed from stages. In stage i we will take a sample of size $m_i = \frac{1}{\epsilon}\ln(\frac{M}{\delta_i})$, where $\delta_i = \delta/2^i$. Let h_i be the hypothesis of $\mathcal{A}$ after the first $i-1$ mistakes. In stage i we use the m_i examples. If h_i does not make any mistake we output h_i. If it makes a mistake on some $\mathbf{x}$, we continue to stage $i+1$.

Recall that we assume $\mathcal{A}$ makes at most M mistakes on any sequence. This implies that algorithm $\mathcal{A}_{PAC}$ utilizes at most $M+1$ stages before it terminates. This follows since we end stage i either since we made an error (which can happen at most M times) or since we did not make an error, and then terminate (can happen at most once). Once $\mathcal{A}_{PAC}$ terminates, in the last stage it did not make any error. Let i^* be the last stage. In stage i^* we have a sample of size m_{i^*} on which h_{i^*} does not make mistakes. From the standard Chernoff bound, this implies that with probability $1-\delta_{i^*}$ the error of h_{i^*} is at most ϵ.

We now need to compute the confidence of algorithm $\mathcal{A}_{PAC}$. The algorithm can fail if in some stage it outputs a hypothesis which has error more than ϵ. For stage i the probability that this will occur is at most δ_i. We now do a union bound over those failure events, and the probability that at some stage $\mathcal{A}_{PAC}$ will output a bad hypothesis is at most $\sum_i \delta_i \leq \delta$. This establishes that with probability at most δ algorithm $\mathcal{A}_{PAC}$ will output a hypothesis which has error more than ϵ. The sample size of $\mathcal{A}_{PAC}$ is $\sum_{i=1}^{M+1}(1/\epsilon)\ln(\frac{M}{\delta_i})$ which is $O(\frac{M^2\ln(1/\delta)}{\epsilon})$. Therefore,

Theorem 4.6 *Algorithm $\mathcal{A}_{PAC}$ will PAC learn the class $\mathcal{C}$.*

Note that algorithm $\mathcal{A}_{PAC}$ does not need to get M, and the value of M is used only in the analysis. (Both for guaranteeing termination and bounding the sample complexity.)

4.2 Learning Linear Separators

We will consider the examples as being from $\{0,1\}^n$ or from $\mathbb{R}^n$ (the algorithm will not be sensitive to the difference). In the realizable case, the goal is to find θ (a threshold) and $\vec{w}$ (a weights vector) defining a hyperplane $\vec{w}\cdot\vec{x} = \theta$ (the notation '$\cdot$' refers to the dot product of two vectors, i.e., $\vec{w}\cdot\vec{x} = \sum_{i=1}^n w_i x_i$) such that all positive examples are on one side and all negative examples are on the other. I.e., $\vec{w}\cdot\vec{x} \geq \theta$ for positive $\vec{x}$'s and $\vec{w}\cdot\vec{x} < \theta$ for negative $\vec{x}$'s.

For simplicity, we use a threshold $\theta = 0$, so we are looking at learning functions like: $\sum_{i=1}^n w_i x_i \geq 0$. (We can simulate a nonzero threshold by adding a "dummy" attribute x_0

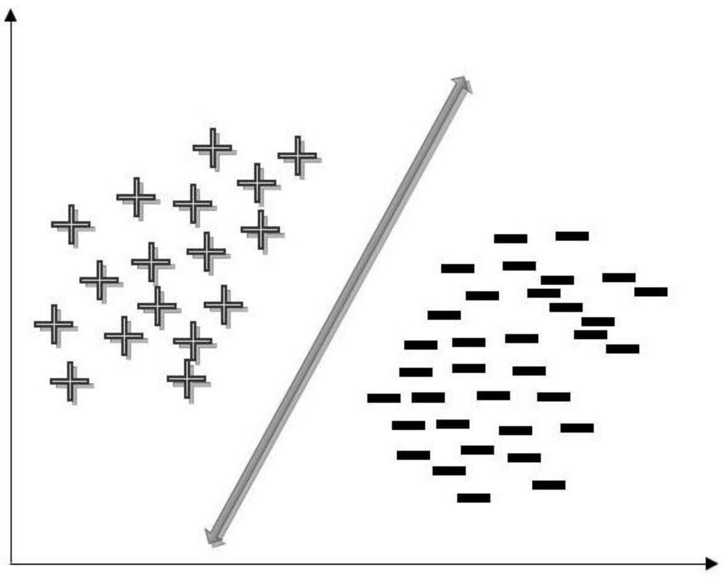

Figure 4.1: A linear separator example

that is always -1, and the weight assigned to it would be θ.) We will also assume that the wights $\vec{w}$ are unit vector, i.e., $\|\vec{w}\|_2 = 1$.

We begin by discussing the Perceptron algorithm, an online algorithm for learning linear separators, and one of the oldest algorithms used in machine learning (by Rosenblatt from 1957).

4.3 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 hyperplane. When we do make a mistake - we move the hyperplane towards it.

We scale all examples $\mathbf{x}$ to have Euclidean length 1 (i.e., $\|\mathbf{x}\|_2 = 1$), since this doesn't affect which side of the plane they are on (given our assumption that $\theta = 0$).

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 $\mathbf{w}_t \cdot \mathbf{x}_t \geq 0$.

3. On a mistake, update as follows: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + c^*(\mathbf{x}_t)\mathbf{x}_t$, namely,

- Mistake on positive (i.e., $c^*(\mathbf{x}_t) = 1$): $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \mathbf{x}_t$.
- Mistake on negative (i.e., $c^*(\mathbf{x}_t) = -1$): $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \mathbf{x}_t$.

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

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

where the last identity uses $\|\mathbf{x}\| = 1$. This implies that if the example was positive $c^*(\mathbf{x}) = +1$ we increase the dot product $(\mathbf{w}_t \cdot \mathbf{x})$ and if the example was negative $c^*(\mathbf{x}) = -1$ we decrease the dot product $(\mathbf{w}_t \cdot \mathbf{x})$. So the update is always in the "right" direction.

Theorem 4.7 *Let $\mathcal{S}$ be a sequence of labeled examples consistent with a linear threshold function $\mathbf{w}^* \cdot \mathbf{x} \geq 0$, 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 hyperplane $\mathbf{w}^* \cdot \mathbf{x} = 0$.

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). The margin γ represents the minimal distance of each example $\mathbf{x}$ from the plane perpendicular to $\mathbf{w}^*$, 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 4.7. We are going to consider two quantities: $\mathbf{w}_t \cdot \mathbf{w}^*$ and $\|\mathbf{w}_t\|$. We will show a claim regarding the change in each of the two quantities. We assume, without loss of generality, that we always make mistakes, since if we do not make a mistake we do not change the weights.
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. (Note that since $\mathbf{w}_1 \cdot \mathbf{w}^* = 0$, this also implies that $\mathbf{w}_{t+1} \cdot \mathbf{w}^* \geq 0$.)

> **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 and since $\|\mathbf{x}\| = 1$). Similarly, if $\mathbf{x}$ was a negative example, we get $(\mathbf{w}_t - \mathbf{x}) \cdot \mathbf{w}^* = \mathbf{w}_t \cdot \mathbf{w}^* - \mathbf{x} \cdot \mathbf{w}^* \geq \mathbf{w}_t \cdot \mathbf{w}^* + \gamma$. (Recall that $\mathbf{x} \cdot \mathbf{w}^* < 0$ when $\mathbf{x}$ is a negative example.)

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, $\|\mathbf{w}_{M+1}\| \leq \sqrt{M}$. Since $\mathbf{w}^*$ is a unit vector, we have $\mathbf{w}_t \cdot \mathbf{w}^* \leq \|\mathbf{w}_t\|$, since the vector maximizing the dot product $\mathbf{w}_t \cdot \mathbf{w}^*$ is $\frac{\mathbf{w}_t}{\|\mathbf{w}_t\|}$, which entails that

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

Therefore, we get

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

and thus $M \leq \frac{1}{\gamma^2}$. □

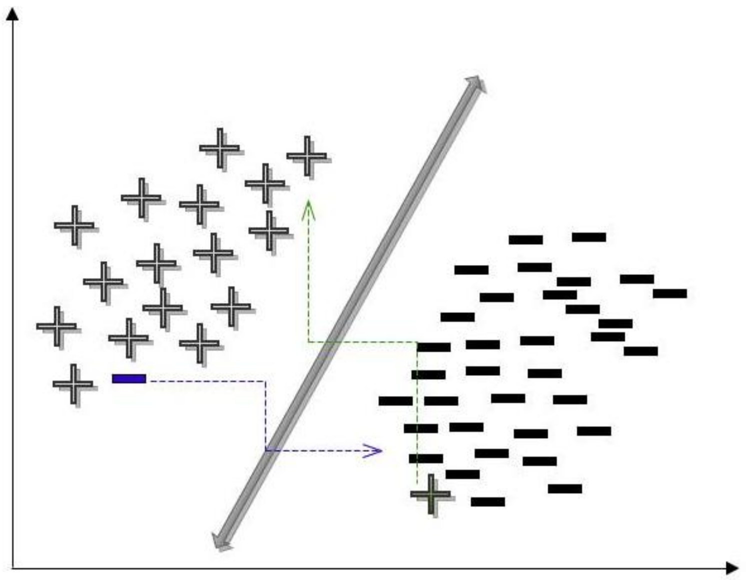

Figure 4.2: A case in which no perfect separator exists

4.3.1 Perceptron - non-realizable case

What if there is no perfect separator (w^*)? 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 need in this case to reconsider Claim 1. Claim 1 said that we make "γ amount of progress" on every mistake.

Now it is 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 us call that TD_γ be the distance we have to move the points, i.e., $TD_\gamma = \sum_{t=1}^{T} \max\{0, \gamma - c^*(\mathbf{x}_t)(\mathbf{x}_t \cdot \mathbf{w}^*)\}$. Similar to Claim 1, we get that after M mistakes, $\mathbf{w}_{M+1} \cdot \mathbf{w}^* \geq \gamma M - \mathrm{TD}_\gamma$.[1] So, combining with Claim 2, we get that $\sqrt{M} \geq \gamma M - \mathrm{TD}_\gamma$, which gives an upper bound on M.

To compute the upper bound on M we can solve the quadratic equation, but a simple upper bound is $M \leq \frac{1}{\gamma^2} + (\frac{2}{\gamma})\mathrm{TD}_\gamma$. The quantity $\frac{1}{\gamma}\mathrm{TD}_\gamma$ is called the total *hinge-loss* of $\mathbf{w}^*$. The hinge loss of a point $\mathbf{x}$ with respect to $\mathbf{w}^*$ can be defined as $\max\{0, 1 - y\}$, where $y = \frac{c^*(x) \cdot \mathbf{x} \cdot \mathbf{w}^*}{\gamma}$ and $c^*(x)$ is the classification of $\mathbf{x}$. The expression for the hinge loss is equivalent to $\frac{1}{\gamma} \max\{0, \gamma - c^*(\mathbf{x})\mathbf{x} \cdot \mathbf{w}^*\}$. The hinge loss is a loss function that begins paying linearly as it approaches the hyperplane after the margin parameter γ (see Figure 4.3). Note that the maximum contribution of an error is $1 + \gamma$, since the examples are normalized to a unit length.

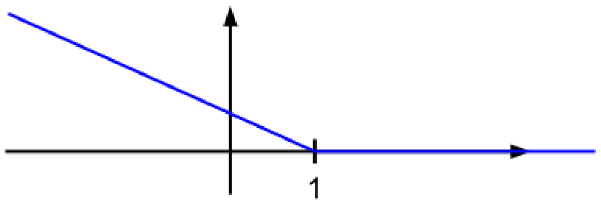

Figure 4.3: Illustrating hinge loss

This is partially good news: we cannot necessarily say that we are making only a number of mistakes proportional to that which $\mathbf{w}^*$ does (in fact, the problem of finding an approximately-optimal separator is NP-hard), but we can say we are doing well in terms of the "total distance" parameter or the hinge loss of $\mathbf{w}^*$.

4.4 WINNOW2

We now describe an alternative algorithm for learning hyperplanes. Here it will be more convenient to assume that $\mathbf{x} \in \{0,1\}^n$ rather than $\mathbf{x} \in [0,1]^n$, but the algorithm extends to the latter case as well. Also we will discuss the case where the coefficients are only positive.

[1]Note that $c^*(x_t)(\mathbf{x}_t \cdot \mathbf{w}^*) \geq \min\{\gamma, c^*(x_t)(\mathbf{x}_t \cdot \mathbf{w}^*)\} = \gamma - \max\{0, \gamma - c^*(x_t)(\mathbf{x}_t \cdot \mathbf{w}^*)\}$.

Suppose we had a separation between the positive examples and the negative examples by the hyperplane $(\mu_1, \ldots, \mu_n)$ where

$$\sum_{i=1}^{n} \mu_i \cdot x_i \ \geq \ 1 \ \Leftrightarrow c^*(\mathbf{x}) = 1$$

$$\sum_{i=1}^{n} \mu_i \cdot x_i \ \leq \ 1 - \gamma \ \Leftrightarrow c^*(\mathbf{x}) = 0$$

where γ will be the margin of the true target function and will influence the algorithm's mistake bound.

Algorithm WINNOW2 will maintain weights at time t: $w^t = (w_1^t, \ldots, w_n^t)$ initialized to 1 at time 1, i.e., $w_i^1 = 1$. In addition there is a threshold $\theta \geq 1$. The algorithm WINNOW2 predict 1 on input $\mathbf{x}_t = (x_1^t, \ldots, x_n^t)$ iff $\sum_i w_i^t x_i^t \geq \theta$. As before, we update only when the algorithm makes an error. WINNOW2 has a parameter $\beta > 1$. We distinguish between two error types. For False-Negative errors, we multiply the weights by the factor β, and call this a *promotion step*. For False-Positive errors, we divide the weights by β, and call this a *demotion* step. We use $\beta = 1 + \frac{\gamma}{2}$ in our implementation of WINNOW2. See Figure 4.4 for the summary of WINNOW2 updates in response to mistakes. (In case there is no mistake, no update is done.)

Name	Update Scheme	target	prediction
Demotion	$\forall x_i = 1$ *set* $w_i = w_i/\beta$	0	1
Promotion	$\forall x_i = 1$ *set* $w_i = \beta \cdot w_i$	1	0

Figure 4.4: The update scheme used by WINNOW2

Theorem 4.8 *If there exists a hyperplane $(\mu_1, \ldots, \mu_n)$ with margin $\gamma \in (0, 1)$, and we run* WINNOW2 *with $\beta = 1 + \frac{\gamma}{2}$ and $\theta \geq 1$, then the number of mistakes is bounded by:*

$$O\left(\frac{1}{\gamma^2} \cdot \frac{n}{\theta} + \left(\frac{1}{\gamma} + \frac{\ln \theta}{\gamma^2}\right) \sum_{i=1}^{n} \mu_i\right)$$

For $\theta = n/2$ we get $O(\frac{1}{\gamma^2} + \frac{\ln n}{\gamma^2} \sum_{i=1}^{n} \mu_i)$.

In order to prove the theorem we first prove three lemmas. We start by distinguishing between the mistakes according to their cause. Let u – the number of promotion steps, and v – the number of demotion steps. The following lemma shows that the number of demotion steps cannot be much larger than the number of promotion steps.

Lemma 4.9 *For any u and v,*

$$v \;\leq\; \frac{\beta}{\beta-1}\cdot\frac{n}{\theta} \;+\; \beta\cdot u$$

Proof: Consider $W^t = \sum_{i=1}^n w_i^t$ the sum of the weights at time t. A promotion step increases W^t by at most $(\beta-1)\theta$, since we had $\sum_i w_i^t x_i^t \leq \theta$ and

$$W^{t+1} = \sum_i w_i^t \beta^{x_i^t} = \sum_i w_i^t + \sum_i (\beta-1) w_i^t x_i^t \leq W^t + (\beta-1)\theta.$$

A demotion step decreases the sum W^t by at least $(1-\frac{1}{\beta})\theta$, since we had $\sum_i w_i^t x_i^t > \theta$ and

$$W^{t+1} = \sum_i w_i^t / \beta^{x_i^t} = \sum_i w_i^t - \frac{\beta-1}{\beta}\sum_i w_i^t x_i^t < W^t - \frac{\beta-1}{\beta}\theta.$$

Combining the two,

$$0 \leq W^t = \sum_{i=1}^n w_i^t \leq n + (\beta-1)\theta\, u - \frac{\beta-1}{\beta}\theta\, v\ .$$

But since the weights are never negative, we have:

$$0 \leq n + (\beta-1)\theta\, u - \frac{\beta-1}{\beta}\theta\, v\ .$$

Thus,

$$v\frac{\beta-1}{\beta} \leq \frac{n}{\theta} + u(\beta-1)\ ,$$

which implies that:

$$v \leq \frac{\beta}{\beta-1}\frac{n}{\theta} + \beta u\ .$$

□

The following lemma shows that any individual weight can not be too large.

Lemma 4.10 *For all i, $w_i \leq \beta\theta$.*

Proof: Since $\theta \geq 1$ and $\beta > 1$, all the weights are initially $1 \leq \beta\theta$. For any j, the value of w_j^t is only promoted during times t in which $x_j = 1$ and $\sum_{i=1}^n w_i^t \cdot x_i^t \leq \theta$. These conditions can only occur if $w_j^t \leq \theta$ immediately prior to the promotion. Thus $w_j^t \leq \beta\theta$ after the promotion. □

Lemma 4.11 *After u promotion steps and v demotion steps: there exists an i for which*

$$\log w_i \;\geq\; \frac{u-(1-\gamma)v}{\sum_{i=1}^n \mu_i}\cdot \log\beta$$

Proof: We consider the function

$$\Phi^t = \Pi_{i=1}^n (w_i^t)^{\mu_i}$$

Promotion Step We have $\sum_{i=1}^n \mu_i \cdot x_i^t \geq 1$. Let w_i^{t+1} be the weights after the promotion step. For each $x_i^t = 1$, we have $w_i^{t+1} = w_i^t\beta$.

$$\Phi^{t+1} = \Phi^t \Pi_{i=1}^n (\beta^{x_i^t})^{\mu_i} = \Phi^t \beta^{\sum_{i=1}^n x_i^t \mu_i} \geq \Phi^t\beta$$

Demotion Step We have $\sum_{i=1}^n \mu_i \cdot x_i^t < 1-\gamma$. Let w_i^{t+1} be the weights after the demotion step. For each $x_i^t = 1$, $w_i^{t+1} = \frac{w_i^t}{\beta}$.

$$\Phi^{t+1} = \Phi^t \Pi_{i=1}^n \left(\frac{1}{\beta^{x_i^t}}\right)^{\mu_i} = \Phi^t \beta^{-\sum_{i=1}^n x_i^t \mu_i} \geq \Phi^t \beta^{-(1-\gamma)}$$

Initially, $\prod_{i=1}^n w_i^{\mu_i} = 1$. After u promotion steps and v demotion steps, we have from the above facts we can conclude that :

$$\prod_{i=1}^n (w_i^t)^{\mu_i} \geq \beta^u \beta^{-(1-\gamma)\cdot v}$$

We can take the log of both sides

$$\sum_{i=1}^n \mu_i \log w_i^t \geq [u-(1-\gamma)v]\log\beta$$

Which means that there exists a i for which:

$$\log w_i^t \geq \frac{[u-(\theta-\gamma)v]}{\sum_{i=1}^n \mu_i}\cdot\log\beta$$

□

Proof of Theorem 4.8: From Lemmas 4.10 and 4.11 we have:

$$\frac{u-(1-\gamma)v}{\sum \mu_i}\log\beta \;\leq\; \log(\beta\theta) = \log\beta + \log\theta.$$

Recall that the total number of mistakes is equal to $u+v$. Since $\beta > 1$ and $\mu_i \geq 0$ we get:

$$u - (1-\gamma)\cdot v \leq \left(1 + \frac{\log\theta}{\log\beta}\right)\sum_{i=1}^{n}\mu_i$$

and by using Lemma 4.9 it is sufficient to guarantee that,

$$u - (1-\gamma)(\frac{\beta}{\beta-1}\cdot\frac{n}{\theta} + \beta\cdot u) \leq (1+\frac{\log\theta}{\log\beta})\sum_{i=1}^{n}\mu_i.$$

We can now use the fact that $\beta = 1 + \frac{\gamma}{2}$. The above requirement can be reduced to,

$$(\frac{\gamma}{2}+\frac{\gamma^2}{2})u - \frac{(1-\gamma)(2+\gamma)}{\gamma}\frac{n}{\theta} \leq \left(1+\frac{\log\theta}{\log(1+\gamma/2)}\right)\sum_{i=1}^{n}\mu_i$$

and therefore,

$$\frac{\gamma}{2}u \leq \frac{(1-\gamma)(2+\gamma)}{\gamma}\frac{n}{\theta} + \left(1+\frac{\log\theta}{\log(1+\gamma/2)}\right)\sum_{i=1}^{n}\mu_i$$

For $\gamma \leq 1$ we get $\log(1+\gamma/2) \geq 3\gamma/8$. Thus, since we assumed that $\theta \geq 1$,

$$\frac{\gamma}{2}u \leq \frac{(1-\gamma)(2+\gamma)}{\gamma}\frac{n}{\theta} + \left(1+\frac{\log\theta}{3\gamma/8}\right)\sum_{i=1}^{n}\mu_i$$

From lemma 4.9 we have,

$$u+v \leq \frac{\beta}{\beta-1}\frac{n}{\theta} + (\beta+1)u$$

Using $\gamma \in (0,1]$ we can derive

$$u+v \leq \frac{8}{\gamma^2}\frac{n}{\theta} + \left(\frac{5}{\gamma}+\frac{14\log\theta}{\gamma^2}\right)\sum_{i=1}^{n}\mu_i$$

Which concludes the proof. ■

4.5 Winnow versus Perceptron

One can generalize the basic analysis we did for Winnow to the case of learning linear separators; the guarantee depends on the L_1, L_∞ margin of the target. In particular, if the target vector w^* is a linear separator such that $w^*\cdot x > c$ on positives and $w^*\cdot x < c-\alpha$ on negatives, then the mistake bound of Winnow is:

$$O\left(\left(\frac{L_1(w^*)L_\infty(X)}{\alpha}\right)^2 \log(n)\right),$$

And the mistake bound of Perceptron is:

$$O\left(\left(\frac{L_2(w^*)L_2(X)}{\alpha}\right)^2\right).$$

The quantity $\gamma = \frac{\alpha}{L_1(w^*)L_\infty(X)}$ is called the "L_1, L_∞" margin of the separator, and our bound is $O(\frac{1}{\gamma^2} \cdot \log(n))$. On the other hand, the Perceptron algorithm has a mistake bound of $O(\frac{1}{\gamma^2})$ where $\gamma = \frac{\alpha}{L_2(w^*)L_2(X)}$ (this called the "L_2, L_2" margin of the separator).

One thing that is lost using Winnow, is obviously the $\log(n)$, but the norms are also different.

Intuitively, if n is large but most features are irrelevant (i.e., target is sparse but examples are dense), then Winnow is better because adding irrelevant features increases $L_2(X)$ but not $L_\infty(X)$. On the other hand, if the target is dense and examples are sparse, then Perceptron is better.