# Online Learning: Fundamental Algorithms

Daniel Khashabi

Summer 2014
Last Update: October 20, 2016

## 1 Introduction

The *Learning Theory* [1], is somewhat least practical part of Machine Learning, which is most about the theoretical guarantees in learning concepts, in different conditions and scenarios. These guarantees are usually expressed in the form of probabilistic concentration of some measure, around some optimal value which is unknown and need to discovered. These bounds are functions of problem specifications, for example,

- The number of samples: the more samples we have, there is a better chance of learning.
- The easiness of the underlying distribution need to be learned.
- The generalization power (or flexibility) of the family of the functions which is being used to approximate the target distribution.

To begin with let's start with a simple example. Suppose we have a sequence of binary observations:

$$y_1, \dots, y_n, \forall y_i \in \{0, 1\}$$

Consider the following tasks:

- **Estimation:** One task is to model this binary sequence. We want to take the easy way and model it with Binomial distribution, with parameter $p$. We call a good parameter for our model $\hat{p}$ that models the original sequence is called the *Estimation* problem. Suppose someone tells us that the original sequence is sampled i.i.d. from a Bernoulli distribution with parameter $p^*$. In this case we can create an averaging estimation $\hat{p} = \frac{1}{n}\sum_{i=1}^{n} y_i$ and have the following bound on our estimation:

$$|p - p^*| < \epsilon(n, \delta) \text{ with probability at least } 1 - \delta \tag{1}$$

  where $\epsilon(n, \delta)$ is proportional to $\frac{1}{\sqrt{n}}$. In general, we are looking for finding tighter estimations ( i.e. $\epsilon(n, \delta)$ getting smaller with faster rate).
  In finding the above bound, we assumed that the original samples are from a Bernoulli distribution. One challenge in real world is that, we don't know from which distribution

[1] Some people that are on the *statistics* side usually call it, *Statistical Learning Theory*, and others at the *computational* side, call it *Computational Learning Theory*. I was fortunate to work with people from the both sides, and these notes will include both *computational* and *statistical* tastes of the problems.

the data is coming from, and therefore finding bounds on estimations is not very trivial. Estimation in real problems means, learning the best model which describes the model, which is our first problem in the learning theory.

- **Batch prediction:** Suppose, given a model for the sequence we want to predict the next coming number $y_{n+1} \in \{0, 1\}$. What is the best prediction and how do we define it? A good measure is to compare the number of the mistakes our prediction might make, relative to the total number of the mistakes that the best algorithm could make in retrospect.

$$R_n = \frac{1}{n} \sum_i \mathbf{1}\{\hat{y}_i \neq y_i\} - \min_{y'_i} \mathbb{E}_{y_i \sim \mathbb{P}} \mathbf{1}\{y'_i \neq y_i\} \tag{2}$$

  The above value is usually called *Regret*. The second term $\min_{y'_i} \mathbb{E}_{y_i \sim \mathbb{P}} \mathbf{1}\{y'_i \neq y_i\}$ is the minimum achievable expected loss, on observations sampled from the distribution $\mathbb{P}$. Without making any assumptions about $\mathbb{P}$, evaluating $\min_{y'_i} \mathbb{E}_{y_i \sim \mathbb{P}} \mathbf{1}\{y'_i \neq y_i\}$ is almost impossible. Let's say the data is coming from a Bernoulli distributions with parameter $p^*$ (i.e. $\mathbb{P} = \text{Ber}(p^*)$) and we are using our previous estimate $\hat{p}_n = \frac{1}{n} \sum_{i=1}^n y_i$. In this case, the expected error by the best estimation is (why?):

$$\min_{y'_i} \mathbb{E}_{y_i \sim \text{Ber}(p^*)} \mathbf{1}\{y'_i \neq y_i\} - \min\{p^*, 1 - p^*\}$$

  The best estimator for $y_{n+1}$ (minimizer of the regret) is the majority vote (why?):

$$y_{n+1} = \mathbf{1}\{\hat{p}_n > 0.5\}$$

  If $p^* = \hat{p}$, our prediction is exact. But with limited data we have the bound 1 on estimation of $\hat{p}$, which means that we can find a similar bound on our prediction. Again, like the estimation, one challenge is that, we don't know if the data is coming from a Binomial distribution or not, which adds further complication.

- **Online prediction:** This scenario is very similar to the previous case, but instead of having all the data together, we ate getting one by one up to the $i$-th instance, and we are suppose to make the best prediction for the $(i + 1)$-th label and then see the real value. This is called online learning since we learn the model by seeing the data sequentially, and at each observation we make prediction based on previous observations and past predictions. What is the best best prediction? Again we resort to the Regret in Equation 2. At each step $j \leq n$ we use choose the $y_{j+1}$ which minimizes $R_j$. Like the previous type, the best prediction is via majority vote *up to step* $j$:

$$y_{j+1} = \mathbf{1}\{\hat{p}_j > 0.5\}, \quad \text{where} \quad \hat{p}_j = \frac{1}{j} \sum_{i=1}^{j} y_i$$

  Although the answer to online and batch prediction in this example were very similar, in more complicated problems their answers can be very different.

**Remark 1.** *Consider the strong Law of Large numbers, which says there exists a predictor, without any assumptions on the generative process of data, such that*

$$\limsup_{n \to \infty} R_n \to 0, \quad \textit{a.s.}$$

*although it does not say anything about this magical method!*

# 2 Several early algorithms

## 2.1 The halving algorithm

Consider this form of the problem. Define an *expert* to be a function which scores the input. Note that an expert is not necessary a good predictor!

Suppose we have $n$ experts, and we want to choose the best one. Here we present a relatively naive algorithm for that, which is based on sequential halving of the search space.

At the beginning suppose there is at least one perfect expert (an expert which does not make any mistake on any of the instances). Consider the following algorithm:

> For each instance, run all the experts, and choose the output label predicted by the majority of the experts, and remove the experts which don't make the majority prediction.

This algorithm would make $\log n$ mistakes until it finds the best expert. It can easily proved by observing that, at each step we remove almost half of the experts. To be more exact, we remove *at least* 25% of the remaining experts (why?). why?

Now consider the case when there is not a perfect expert. We can do the same strategy as we had before, but once we cross off all of the experts start over. how? If the number of mistakes the best expert makes in the hindside is $OPT$, this algorithm would make $O((OPT+1)\log n)$ how? .

## 2.2 Weighted Majority algorithm

Here instead of crossing off experts, we use a weighted strategy; instead of removing an expert, we halve its weight by half. For decision, choose the prediction, which has the biggest weight.

**Theorem 1.** *The weighted majority algorithm has mistake bound* $2.4(m + \log n)$*, where* $n$ *is the number of the mistakes, and* $m$ *is the number of the mistake the best expert has made so far.*

*Proof.* Define the following two variables:

$$\begin{cases} M = \#\text{number of the mistakes made so far.} \\ W = \#\text{total weight.} \end{cases}$$

First, we know that the initial value of the $W$ is $n$. At each mistake, the total weight $W$ drops by at least %25. proof? So after $M$ mistakes, the total weight is at most $n(3/4)^M$. Also the weight of the best expert is $(1/2)^m$. Then:

$$(1/2)^m \leq n(3/4)^M \Rightarrow M \leq 2.4(m + \lg n)$$

□

**Exercise 1.** *Define an alternative strategy for the weighted majority algorithm and prove that its mistake bound is $O(m + \log n)$, where $n$ is the number of the experts, and $m$ is the number of the mistakes the best expert makes in the hind sight. Just like before we choose the expert with the maximum weight, but instead of halving the weights of all the wrong experts, we apply the halving, only if the weight of the expert is at least $1/4$ of the average weight of all the experts.*

*Proof.* Just like the example we showed, define the following variables:

$$\begin{cases} M = \#\text{number of the mistakes made so far.} \\ W_{begin} = \#\text{total weight at the begining of the interval.} \\ W_{end} = \#\text{total weight at the end of the interval.} W(t) = \#\text{total weight at time } t. \end{cases}$$

First observation is that, the weight of each expert is at least the $W(t)/(8n)$. The reason is that, the weight of each expert gets halved only if its weight is more than $W(t)/(8n)$. Therefore no weight goes bellow $W(t)/(8n)$. Since this holds for all of the weights, essentially this also holds for the weight of the best expert.

Since the weight of the best expert gets halved for $m$ times, the weight of the best expert is lower bounded by $\left(\frac{1}{2}\right)^m W_{start}/(8n)$.

Another observation is that, at each mistake at most $W/4$ of the total weight is fixed, and at least $W/2 - W/4 = W/4$ gets cut in half. why? In other words,

$$W_{end} = \left(\frac{7}{8}\right)^M W_{start}$$

$$\left(\frac{1}{2}\right)^m W_{start}/(8n) \leq \left(\frac{7}{8}\right)^M W_{start}$$

Which would give us the desired bound. □

Can we make better? yes! If make the strategy randomized, it will give us an improved bound. more examples here: link

### 2.3 Randomized Weighted Majority algorithm

In the randomized strategy, instead of choosing the decision with maximum weight, we use the weights as probability distribution and choose the label randomly with respect to the weight distribution. Also, let's make the strategy a little more complicated, by multiplying (or dividing) by $1 - \epsilon$ (instead of $1/2$).

**Theorem 2.** *The (expected) mistake bound of the randomized weighted majority algorithm is:*

$$M \leq \frac{m \ln \frac{1}{1-\epsilon} + \ln n}{\epsilon}$$

*The followings are special cases for different values of $\epsilon$:*

$$\epsilon = 1/2 \to M \leq 1.39m + 2 \log n$$
$$\epsilon = 1/2 \to M \leq 1.15m + 4 \log n$$
$$\epsilon = 1/2 \to M \leq 1.07m + 8 \log n$$

*Proof.* Suppose at time $t$, $\alpha_t$ is the fraction of the weights (of the experts) that have made mistake. So, we remove $\epsilon\alpha_t$ of the total weight at step $t$:

$$W(T) = n\prod_{t=1}^{T}(1-\epsilon\alpha_t)$$

$$\ln W(T) = \ln n + \sum_{t=1}^{T}\ln(1-\epsilon\alpha_t) \leq \ln n + \epsilon\sum_{t=1}^{T}\alpha_t$$

Note that:

$$M = \mathbb{E}\left[\#\text{ of mistakes }\right] = \sum_{t=1}^{T}\alpha_t$$

$$\ln W(T) \leq \ln n - \epsilon M$$

If the best expert makes $m$ mistake up to time $T$, we know that:

$$(1-\epsilon)^m \leq W(T) \Rightarrow m\log(1-\epsilon) \leq \ln W(T)$$

$$\Rightarrow m\ln(1-\epsilon) \leq \ln n - \epsilon M$$

$$\Rightarrow M \leq \frac{1}{\epsilon}\left[-m\ln(1-\epsilon) + \ln n\right] \approx (1+\epsilon/2)m + \frac{1}{\epsilon}\ln n \tag{3}$$

□

**Remark 2.** *The bound in the Equation 3 could be written in the following form: how?*

$$\mathbb{E}\left[\#\ \textit{of mistakes}\ \right] \leq (1+\epsilon)OPT + \epsilon^{-1}\log n$$

*To make tighter bound (and assuming that we know $OPT$, and $OPT \geq logn$), we set $\epsilon = (logn/OPT)^{1/2}$. With simplification how? it can be shown that*

$$\frac{\mathbb{E}\left[\#\ \textit{of mistakes}\ \right]}{T} \leq \frac{OPT}{T} + O\left(\sqrt{\frac{\log n}{T}}\right)$$

*As it can be seen, as $T \to +\infty$, the regret goes to zero. Algorithms with this property are usually called "no-regret" algorithms.*

more examples here: link

*http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15859−f11/www/notes/lecture16.pdf*

*http://cseweb.ucsd.edu/ yfreund/papers/gameslong.pdf*

### 2.4 Sleeping expert's problem

*http : //www.cs.cornell.edu/w8/ yogi/v/research/publications/kleinberg−niculescumizil−sharma−colt*2008.p

**Proposition 1.** *An online learning algorithm is lazy if it changes its state only when it makes a mistake. We can show that for any deterministic online learning algorithm $A$ achieving a mistake bound $M$ with respect to a concept class $C$, there exists a lazy algorithm $A'$ that also achieves $M$ with respect to $C$.*

*Recall the definition: Algorithm $A$ has a mistake bound $M$ with respect to a learning class $C$ if $A$ makes at most $M$ mistakes on any sequence that is consistent with a function in $C$.*

*Here is a brief proof. Assume that there exists a sequence of examples on which $A'$ makes $M' > M$ mistakes. Cross out all examples on which $A'$ doesnt make a mistake, and let $S$ denote the resulting sequence. Both $A$ and $A'$ behave identically on $S$, and $A$ makes at most $M$ mistakes on any sequence of examples, including $S$. This leads to the desired contradiction. (Obviously, if the original sequence is consistent with a concept in $C$, so is $S$.)*

## 3 Follow-The-Perturbed-Leader

The whole thing: adasd Follow the perturbed leader `http://theory.stanford.edu/~tim/f13/l/l17.pdf` 4 12 12 `http://www.cs.cornell.edu/courses/cs683/2007sp/lecnotes/week7.pdf` `http://ocobook.cs.princeton.edu/OCObook.pdf` `http://classes.soe.ucsc.edu/cmps290c/Spring09/` `https://web.stanford.edu/class/cs229t/notes.pdf`

## 4 Winnow algorithm

Developed in more than 20 years ago, in [1]. It is very relevant and similar to the perceptron algorithm, but with different properties. The only difference is that, in the Winnow algorithm, the updates are *multiplicative*. During this note, we will represent multiple versions of Winnow, but all of them have similar properties and analysis.

The key property of the Winnow update rule [1] is that the number of examples required to learn a linear function grows linearly with the number of relevant features and only logarithmically with the total number of features. This is a desirable property if there are many irrelevant features in the instances, while the relevant features might be small.

The versions we introduce here all have positive weights, although the Winnow algorithm is not limited to positive weights LTFs.

Therefore, it is typically not expressive enough for applications. Using the "duplication trick" [?, ?].

### 4.1 Winnow for LTF (Linear Thresholding Function)

Consider the following definition for hypothesis class of LTF:

$$\text{Predict positive if } \sum_i w[i]x[i] \geq n$$

The Winnow algorithm is as following:

- Initialize all weights $w_i = 1$.
- Loop over instances $(x_j, y_j)$
  - Given input instance $x_j$, make prediction $h(x_j)$.
  - If $h(x_j) \neq y_j$ and $y_j = 1$, then $w[i] \leftarrow 2 \times w[i]$, for all $i$.
  - If $h(x_j) \neq y_j$ and $y_j = -1$, then $w[i] \leftarrow 0$, for all $i$.

Note that, disjunction functions are special case of LTF (see ??). In other words, any disjunction could represented with an LTF. Therefore, our presented Winnow could solve disjunctions.

**Theorem 3.** *Suppose the target function is a (r out of n) disjunction function. The mistake bound of the algorithm presented above is the following:*

$$1 + r(1 + \lg n) = O(r \lg n)$$

*Proof.* To prove this, we first prove that, the number of the mistakes on positive instances is bounded. At first, each of weights are 1. Also, on positive instances, only the weights that are less than $n$ can be updated (doubled). In other words, each weight can be updated at most $\log_2 n + 1$ times. Thus the number of the mistakes on positive instances is bounded by:

$$M_+ \leq r(1 + \log_2 n)$$

Then we bound the number of the mistakes on negative instances by a constant factor of the number of the mistakes on positive instances, which would result in the final bound. First, note that, when making mistake on negative instances, the value of the total weight decreases at least by $n$(why?). Also, when making mistake on a positive instance, the total weight increases by at most $n$. Also, initially the total weight is $n$. Then,

$$n + M_+ n - M_- n > 0 \Rightarrow M_- < M_+ + 1$$

This would give the over bound on mistakes:

$$M = M_- + M_+ \leq 2r(1 + \log_2 n) + 1 \in O(r(1 + \log_2 n))$$

□

link

## 4.2 A general form for Winnow

Here is a more general algorithm:

- Initialize all weights $w_i = 1$, and $\epsilon = 1/(2k)$.
- Loop over instances $(x_j, y_j)$
  - Given input instance $x_j$, make prediction $h(x_j)$.
  - If $h(x_j) \neq y_j$ and $y_j = 1$, then $w[i] \leftarrow w[i](1+\epsilon)$, for all $i$.
  - If $h(x_j) \neq y_j$ and $y_j = -1$, then $w[i] \leftarrow w[i]/(1+\epsilon)$, for all $i$ that $x[i] = 1$.

**Theorem 4.** *The Winnow algorithm with the above representation makes at most $O(rk \log n)$ mistakes.*

*Proof.* Suppose the number of the mistakes on positive examples is $M_+$, and the number of the mistakes on negative examples is $M_-$. This the mistake bound is $M = M_+ + M_-$. Define the total sum to be $S = \sum_i w[i]$ Also define the following short hands:

$$\begin{cases} \text{positive-instance-mistake} = \text{p.i.m} \\ \text{negative-instance-mistake} = \text{n.i.m} \end{cases}$$

We know at first:

$$w[i] = 1, \quad \forall i,$$

and

$$S = \sum_i w[i] = n.$$

On any p.i.m. the increase of $S$ is at most $\epsilon n$. Thus for all the p.i.m. the increase of $S$ is $\epsilon n M_+$. On any n.i.m. the decrease of $S$ is at least $\frac{\epsilon}{1+\epsilon} n$. Thus for all the n.i.m. the decrease of $S$ is $\frac{\epsilon}{1+\epsilon} n M_-$. At the end of the algorithm, the sum is:

$$S = n + \epsilon n M_+ - \frac{\epsilon}{1+\epsilon} n M_-$$

Since the sum never goes negative then we have the constraint that,

$$S = n + \epsilon n M_+ - \frac{\epsilon}{1+\epsilon} n M_- \geq 0. \tag{4}$$

We can make another inequality by focusing on the number of the relevant features (attribute) which has size $k$. For any positive instance, the number of the active features is at least $k$ (in particular the $k$ relevant one), and in the negative instances, the number of the active relevant features is at most $k-1$. For any feature, suppose we increase its weight $c_+$ times, and decrease its weight $c_-$ times. We know that its weight shouldn't be more than $n$ (unless all predictions become positive). Thus,

$$\frac{(1+\epsilon)^{c_+}}{(1+\epsilon)^{c_-}} \leq n \Rightarrow (1+\epsilon)^{c_+ - c_-} \leq n$$

which means the absolute number of increases $c_+ - c_-$ [2] is upper bounded by $\log_{1+\epsilon} n$. Since this holds for any $k$ feature, the sum of absolute number of increases (sum of coins; see the footnote) for all the relevant features must be less than $k \log_{1+\epsilon} n$. For any p.i.m, the number of the coins added is $k$ (since at least $k$ active relevant feature), and in the n.i.m at most $k-1$ active instances. And we need to have:

$$kM_+ + (k-1)M_- \leq k \log_{1+\epsilon} n. \tag{5}$$

Combining Equations 4 and 5 we get the following:

$$M \in O(rk \log n)$$

□

**Problem 1.** *Suppose there exists a $w^*$ such that,*

$$\begin{cases} w^*.x_i \geq c & \text{for all } i \text{ such that } y_i = +1 \\ w^*.x_i \leq c - \gamma & \text{for all } i \text{ such that } y_i = -1 \end{cases}$$

*Then for the choice of $\epsilon = \gamma/2$, the mistake bound for the Winnow algorithm is $O((l_1(w^*)/\gamma)^2 \log n)$. Hint: think about the variations of $\sum_i w^*[i] \log_{1+\epsilon} w[i]$*

*Proof.* Define the potential function at time $t$ to be the following:

$$\Phi^t = \sum_i w^*[i] \log_{1+\epsilon} w^t[i]$$

We know the updates could be written in the following short form:

$$w^{t+1}[i] \leftarrow w^t[i](1+\epsilon)^{yx[i]}$$

where $y \in \{\pm 1\}$, and $x[i] \in \{0, 1\}$. Also given this update we know that:

$$\begin{aligned} \Phi^{t+1} &= \sum_i w^*[i] \log_{1+\epsilon} w^{t+1}[i] = \sum_i w^*[i] \log_{1+\epsilon} \left( w^t[i](1+\epsilon)^{yx[i]} \right) \\ &= \sum_i w^*[i] \log_{1+\epsilon} \left( w^t[i](1+\epsilon)^{yx[i]} \right) \\ &= \sum_i w^*[i] \log_{1+\epsilon} w^t[i] + \sum_i yx[i] = \Phi^t + \sum_i y.w^*[i].x[i] \end{aligned}$$

$$\Rightarrow \Phi^{t+1} = \Phi^t + \sum_i y.w^*[i].x[i] \tag{6}$$

Based on the assumption in the question, we can make the following conclusions.

1. On positive instances ($y = +1$), the potential function $\Phi^t$ increases by at least $c$.
2. On negative instances ($y = -1$) , the potential function $\Phi^t$ decreases by at most $c - \gamma$.

[2] Increases minus decreases; some people like to think about $c_+$ as adding coin on each relevant feature, and $c_-$ removing relevant feature from the feature.

At each step we know that (otherwise we never make mistake on positive instances)

$$w[i] \leq n(1+\epsilon).$$

Then,

$$\log_{1+\epsilon} w[i] \leq \log_{1+\epsilon} n(1+\epsilon) = \log_{1+\epsilon} n + 1.$$

$$\Rightarrow \sum_i w^*[i] \log_{1+\epsilon} w[i] \leq \sum_i w^*[i] \left\{\log_{1+\epsilon} n + 1\right\} = l_1(w^*) \left(1 + \log_{1+\epsilon} n\right).$$

Then using the results we got from the Equation 6:

$$M_+ c - M_-(c-\gamma) \leq l_1(w^*) \left(1 + \log_{1+\epsilon} n\right) \tag{7}$$

Also from the original analysis, remember the inequality:

$$n + M_+ \epsilon n - M_- \frac{\epsilon}{1+\epsilon} n \geq 0 \tag{8}$$

Combining these two inequalities proper choice of $\epsilon$ as a function of $\gamma$, would give us the following bound:

$$M = M_+ + M_- \in O\left(\left(\frac{l_1(w^*)}{\gamma}\right)^2 \log n\right)$$

If we set $l_1(w^*) = 1$, it would give the desired bound. □

**Remark 3.** *One interesting open question is that, can we learn a computationally efficient decition list with mistake bound poly$(L, logn)$? Note that, although the halving algorithm attains this mistake bound, it is not computationally efficient.*

link2

# 5 Perceptron Algorithm

The perceptron algorithm is one of the oldest algorithms for learning linear separators which is invented around 1960s. Although being very simple, it is still being widely used. Even many fancier and newer algorithms, are modifications of the this algorithm.

Suppose we have linear combination of features $x$ and their weights $w$, as $w.x$. If $w.x \geq 0$, we consider the prediction positive, otherwise we consider it negative. Here is the algorithm in its simplest form:

- Initialize the weights $w = 0$
- If mistake on positive example,
$$w \leftarrow w + x$$
- If mistake on negative example,
$$w \leftarrow w - x$$

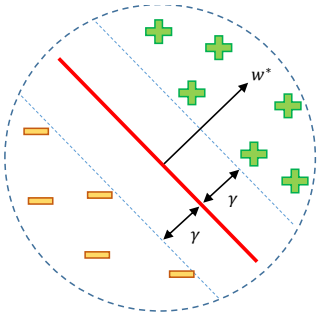


Figure 1: A representation of Perceptron Algorithm updates, and its margin.

**Theorem 5.** *Suppose the data (scaled to to the unit ball) is consistent with some LTF with $w^*$, such that*

$$\begin{cases} w^*.x > 0 \\ \|w^*\| = 1 \\ \gamma = \min_x \frac{|w^*.x|}{\|x\|} \end{cases}$$

*Then the number of mistakes is less than $1/\gamma^2$.*

*Proof.* The key to prove this, is in analyzing the behavior of $\|w\|$ and $w.w^*$. First, we show that each mistake increases $w.w^*$ by at least $\gamma$.

$$(w + x).w^* = w.w^* + x.w^* \geq w.w^* + \gamma$$

And after $M$ mistakes:

$$w.w^* \geq \gamma M$$

Each mistake increases $\|w\|^2 = w.w$ by at most one.

$$\|w + x\|^2 = (w + x).(w + x) = w.w + 2w.x + x.x \leq w.w + 1$$

And after $M$ mistake,

$$\|w\| \leq \sqrt{M}$$

Combining the two inequalities, by knowing that $w.w^* \leq \|w\|$ (since $w^*$ is a unit vector) we get:

$$\gamma M \leq w.w^* \leq \|w\| \leq \sqrt{M}$$

And we get $M \leq \frac{1}{\gamma^2}$. □

One can prove that the bound that we got for the perceptron is optimal in terms $\gamma$. The next theorem is proving this.

**Theorem 6.** *The Perceptron algorithm is optimal in terms of $\gamma$. No deterministic algorithm can have mistake bound less than $1/\gamma^2$, which maintaining a separation margin of $\gamma$.*

*Proof.* For proof, we design a set of $1/\gamma^2$ data points, such that they always satisfy the margin separation of $\gamma$, and show that, for any algorithm, we can label then in a way that it makes mistake on all the data points.

Suppose our data points are the unit vectors $\mathbf{e}_i$ in dimension $i$. $x_1 = \mathbf{e}_1, \ldots, x_{1/\gamma^2} = \mathbf{e}_{1/\gamma^2}$. Consider the following family of target functions:

$$w^* = \gamma(\pm x_1 \pm x_2 \ldots \pm x_{1/\gamma^2})$$

For any of these target functions (with any arbitrary signs) $\|w^*\| = 1$, and it separates points with margin $\gamma$ (with respect to any of $\mathbf{e}_i$). So, for any deterministic algorithm, as adversary could choose the signs of the target separator in a way that, it makes mistake on all of the data points. □

By now, we have seen that, if the data is linearly separable by some (relatively big) $\gamma$ (large-margin condition) the Perceptron algorithm is a very good algorithm.

In the previous case, the analysis were dependent on having a separator and the separation width, $\gamma$. But what if we don't have such grantees on our data? Can we get other bounds on our algorithm? (In other words, $w^*$ is not perfect)

**Problem 2.** *Remember the Problem 1. We can make a similar statement in the perceptron algorithm. Specifically, suppose there exists a $w^*$ such that,*

$$\begin{cases} w^*.x_i \geq c & \text{for all } i \text{ such that } y_i = +1 \\ w^*.x_i \leq c - \gamma & \text{for all } i \text{ such that } y_i = -1 \end{cases}$$

*Then the mistake bound for the Perceptron algorithm is $O((l_2(w^*)l_2(X)/\gamma)^2)$.*

# 6 Bibliographical notes

These notes initiated during the *Statistical Learning Theory*, by Maxim Raginsky [3] and *Machine Learning Theory*, by Avrim Blum(while his sabbatical at UIUC). In particular some sample questions from Maxim's homework assignments. I have also used Sasha Rakhlin and Karthik Sridharan's class notes. Some of the material for *Online Learning* is borrowed from [4, 5], and some are borrowed from Nina Balcan's notes.

# References


[1] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine learning*, 2(4):285–318, 1988.

[2] Arkadiĭ Nemirovsky. Problem complexity and method efficiency in optimization.

[3] Maxim Raginsky. Lecture notes: Ece 299: Statistical learning theory. *Tutorial*, 2011.

[4] Alexander Rakhlin and A Tewari. Lecture notes on online learning. *Draft, April*, 2009.

[5] Shai Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194, 2011.