Learning Theory: Basic Guarantees

Daniel Khashabi

Fall 2014
Last Update: April 28, 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!

Proposition 1. *Our method for the Bernoulli example, by minimizing regret (either batch or online case) with*

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

can be arbitrary bad, depending on the sequence. Suppose there is a sequence of $\{1, 0\}$ *with alternating signs. The predictor will also alternate between 0 and 1, but in opposite of the observation sequence. In other words, this predictor will make mistake on all of the instances. In fact, many problems are like this; without many assumption on the underlying generative process for the data, it is almost impossible to come up with prediction guarantees, for a fixed predictor.*
One other method is to find guarantees in expectation (rather than finding deterministic guarantees). We will talk more about randomization in later sections.

Problem 1 (Randomized bound)**.** *Consider the following two problems:*

1. *Define a function* $\phi(a)$ *to be stable, iff*

$$\left|\phi(a) - \phi(a')\right| \leq \frac{1}{n}, \text{ for any } a, a' \text{ with } \left\|a - a'\right\|_1 = 1$$

Suppose we find a stable function such that it upperbounds expected average of mistakes:

$$\mathbb{E}\left[\frac{1}{n}\sum_i \mathbf{1}\{\hat{y}_i \neq y_i\}\right] \leq \phi_n(y_1, \ldots, y_n), \quad \forall y_1, \ldots, y_n$$

Prove that this upper bound holds, if and only if, for a uniform distribution of $\{0, 1\}^n$*,*

$$\mathbb{E}\phi_n(y_1, \ldots, y_n) \geq \frac{1}{2}, \quad \forall y_1, \ldots, y_n$$

2. *Show that, using the previous characterization of* $\phi_n(.)$ *show existence of an algorithm that achieves the following guarantee:*

$$\mathbb{E}\left[\frac{1}{n}\sum_i \mathbf{1}\{\hat{y}_i \neq y_i\}\right] \leq \min\{\hat{p}_n, 1 - \hat{p}_n\} + Cn^{-1/2}$$

What is the smallest value of C*?*
Hint: The term $\min\{\hat{p}_n, 1 - \hat{p}_n\}$ *is at least* 0.5*.*

2 Initial steps to model learning

2.1 Some combinatorial problems and consistency

To make the progress we will start with some combinatorial problems, and define our goal. In these problems we aim to reach at *consistency*, which is defined as following:

Definition 1 (Consistency)**.** *A set of labeled examples are given as* S*. Suppose we want to to learn to predict* S*, by choosing a concept* c*, that belongs to a concept class* $\mathcal{C}$*. If* c *is consistent with all of the elements in* S*, we output it; otherwise we output that "there is no consistent concept".*

Now let's read about some problems.

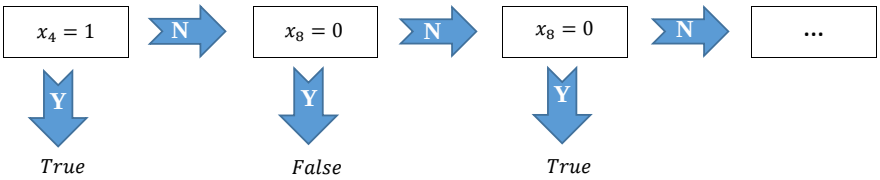

Figure 1: An example of a decision list.

Problem 2 (Monotone CNFs)**.** *Suppose we have binary feature vectors of size d. Our concept class is the set of all CNFs of the feature values (without negation). For example, one feature vector, and its associated label could be the following:*

$$1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1, \quad +$$

Obviously $d = 10$. A concept consistent with the above instance is $x_1 \vee x_2 \vee x_7 \vee x_{10}$. Now a class of instances S is given to us, and we want to find a consistent function with S. Here is an algorithm that would do this for us:

Remove all of the features (from all of the instances), that are zero in at least one positive instance (Since these features cannot be included in the target function). Take AND of the every feature remained. If nothing is remained, output that such consistent function is impossible to be made.

There is a *Monotone DNF* problem, similar to the *Monotone-CNF*, but $\vee$ replaced with $\wedge$. A consistent function for this concept class could be found using the De Morgan law and solution for the Monotone CNF.

There are similar and relevant problems: *non-monotone CNF* or simply *CNF* problem, in which we can use negations for each feature vector (Similarly *non-monotone DNF* or simply *DNF* problem). If we limit the size of each laterals to k features, we call it $k - CNF$ problem (similarly $k - DNF$ problem).

Another hypothesis class which we will talk about is *Decision List*. A Decision List is a chain of if-else rules. An example is shown in the Figure 1. This example corresponds to the following rules:

$$\begin{cases} x_4 \Rightarrow True \\ \bar{x_8} \Rightarrow False \\ x_{10} \Rightarrow True \\ \dots \end{cases}$$

Although in the figure, our conditions are on single features, in general the conditional could be laterals (conjunctions or disjunctions of single features). The case when each literal is a conjunction of k variables is commonly called k-Decision List; in the rest of these notes, we will limit ourselves to only one variables. Note that, for each variable there are four possible cases. For example for

x_1,

$$\begin{cases} x_1 \Rightarrow True \\ \bar{x_1} \Rightarrow True \\ x_1 \Rightarrow False \\ \bar{x_1} \Rightarrow False \end{cases}$$

As a result, the size of the hypothesis space is $|\mathcal{H}| = n!4^n$ ($n!$ for different orderings).

Example 1 (Algorithm to find consistent DL)**.** *Given a training data, we want to create decision list consistent with S. Here is a bottom-up algorithm for doing that:*

- *Start with an empty list.*
- *Find laterals that are consistent with a subset of the data, add it to the end of the DL, and cross-out all of the instances covered.*
- *If we fail to find a consistent lateral, output "no consistent DL with the data".*

The above construction of a consistent DL is not unique. One can do a top-down approach, by creating a list of all rules, and removing the inconsistent ones by observing instances (left as an exercise to the reader).

Exercise 1. *It is easy to show the conjunctions or disjunctions are special cases of decision lists. Show that, decision lists are special cases of Linear Thresholding Functions (LTF). In other words, given any decision list, it can be written as a weighted linear combination of some binary variables.*

$$f(x) = \sum_i w_i x_i \begin{cases} \geq 0 & \Rightarrow + \\ < 0 & \Rightarrow - \end{cases}$$

Proof. For a given decision list, one possible LTF is the following:

$$f(x) = \sum_i h(x_i) g(x_i) \times \frac{1}{2^i}$$

where,

$$g(x_i) = \begin{cases} x_i & \text{if in the decison list, the condition is } x_i \\ 1 - x_i & \text{if in the decison list, the condition is } \bar{x}_i \end{cases}$$

$$h(x_i) = \begin{cases} +1 & \text{if in the decison list outputs } T \\ -1 & \text{if in the decison list outputs } F \end{cases}$$

Proving this can be done with induction. □

2.2 PAC model

Assume we are given labeled data S, sampled i.i.d. from an unknown distribution $\mathcal{D}$ over the instance space $\mathcal{X}$, and labeled by some unknown target concept c^*. Suppose we learn concept $\hat{c} \in \mathcal{C}$. If we find a good concept class that is guaranteed to have a desired error rate, with a desired probability, we have a PAC (Probably Approximately Correct) solution for the problem. We will explain this more formally.

First let's start explaining the concept on the *Decision List* problem, which we talked about in the Section 2.1.

Example 2 (Decision List Algorithm)**.** *Given a training data, we want to create decision list consistent with S. Here is a bottom-up algorithm for doing that:*

- *Start with an empty list.*
- *Find laterals that are consistent with a subset of the data, add it to the end of the DL, and cross-out all of the instances covered.*
- *If we fail to find a consistent lateral, output "no consistent DL with the data".*

The above construction of a consistent DL is not unique. One can do a top-down approach, by creating a list of all rules, and removing the inconsistent ones by observing instances.
Now given this algorithm, we show that, if $|S|$ is of reasonable size, then the hypothesis is "approximately correct", or,

$$\mathbb{P}\left[\exists h \in \mathcal{H}_{DL} \text{ with } err(h) \geq \epsilon\right] \leq \delta$$

Proof. Consider a "bad" hypothesis h whose error probability is at least ϵ: $err(h) \leq \epsilon$. The probability that this this bad hypothesis is consistent with the training data S (given the independence assumption):

$$(1-\epsilon)^m$$

Since size of the hypothesis space is at most $n!4^n$, then the union bound for all of the hypothesis would give us,

$$\begin{aligned}\mathbb{P}\left[\exists h \in \mathcal{H}_{DL} \text{ with } err(h) \geq \epsilon \text{ is consistent }\right] &\leq |\mathcal{H}|\,(1-\epsilon)^m \\ &\leq |\mathcal{H}| e^{-\epsilon|S|}\end{aligned}$$

Equivalently, for any $\epsilon > 0, \delta > 0$, and enough training data,

$$|S| \geq \frac{1}{\epsilon}\left[\ln|\mathcal{H}| + \ln\frac{1}{\delta}\right] = \frac{1}{\epsilon}\left[n\ln n + \ln\frac{1}{\delta}\right]$$

we will have a Probably Approximately Correct prediction. □

Remark 2 (Sample complexity)**.** *Given a PAC-learnable hypothesis space, and $\epsilon > 0, \delta > 0$, the number of the data needed is called "sample complexity".*

Exercise 2. *Suppose our hypothesis class is conjunction of features. Here is an example,*

$$x_1\bar{x_2}\bar{x_3}\dots x_n\bar{x_n}$$

Given a training set S, give an algorithm that finds a consistent member of this class, and prove that this hypothesis class is PAC-learnable.
Hint: For a given, $\epsilon > 0, \delta > 0$, you should end up with sample complexity better than $\frac{1}{\epsilon}\left[n\ln(3) + \ln(1/\delta)\right]$.
Hint 2: Show that, the size of the hypothesis space is 3^n.

After giving example, we give the formal definition.

Definition 2 (PAC Learning)**.** *If for any concept class $\mathcal{C}$, and any target concept $c^* \in \mathcal{C}$, and for any distribution D over the instance space $\mathcal{X}$, and for any given arbitrary $\delta > 0, \epsilon > 0$,*

1. *the algorithm A using the labeled instances sampled with distribution D, produces a hypothesis $h \in \mathcal{H}$ (where $\mathcal{H}$ is a hypothesis class), which with probability at least $1 - \delta$ produces at most ϵ error.*

2. *A runs in time $poly(n, |S|)$, where $|S|$ is sample size, and n is size of an example.*

3. *The sample size needed to achieve this goal is $poly(1/\epsilon, 1/\delta, n, size(c^*))$*

Then the algorithm A, PAC-learns the concept class C.

Remark 3 (Proper PAC-learning)**.** *If $\mathcal{H} = \mathcal{C}$, then we call it "Proper Learning" scenario. In the practical scenarios it is usually the case that, $\mathcal{H} \neq \mathcal{C}$. We call the learning "agnostic" if we simply want to find the best $h \in \mathcal{H}$ we can, without having any prior assumption on the target concept. In this case, instead of trying to reach a constant error probability ϵ, we try to read reach the error by the best predictor, $OPT(H) + \epsilon$. This is related to another notion called "regret", which we will talk later.*

Remark 4 (Weak Learning)**.** *If instead of fixing ϵ, we require to learn a hypothesis of error at most $\frac{1}{2} - 1/poly(n)$, it is usually called "Weak Learning" scenario.*

Remark 5 (Decision Trees)**.** *Decision Trees are not known to be PAC-learnable! Although it might be easy to find a consistent hypothesis of the data (how?).*

Exercise 3. *Find out the reason that, Decision Trees are not PAC-learnable. Where does the argument break?*

Remark 6. *If the computation time is not an issue, any hypothesis class is PAC-learnable!*

Remark 7. *Some drawbacks of the PAC-learning model:*

1. *It doesn't not consider the prior knowledge or prior belief in the models.*

2. *It doesn't address the unlabeled or similar scenarios.*

2.3 Mistake Bound model

Suppose we are in the *online learning* scenario.

Definition 3. *An algorithm A is said to learn the concept class $\mathcal{C}$ with mistake-bound C, if for any ordering of the examples consistent with c, the total number of the mistakes made by A is bounded by $poly(n, size(c))$, where n is size of an example.*

Remark 8. *The definition of the mistake-bound has nothing to do with the running time of the algorithm.*

Remark 9. *Based on the definition of the mistake-bound, it doesn't make sense to talk about "how much data is needed for learning". Since the algorithm might see many instances and doesn't make any mistakes (hence, doesn't learn anything!).*

Definition 4 (MB-learnable)**.** *The concept class is called MB-learnable, if there exists an algorthm with a constant mistake bound, and running time $poly(n, s)$.*

Example 3 (Conjunction of features)**.** *Suppose our hypothesis class is conjunction of features. Given a training set S, how can we learn with this hypothesis class?*

1. *Initialize h to be $x_1\bar{x_1}x_2\bar{x_2}\dots x_n\bar{x_n}$*
2. *For each instance in the training set x_i, do prediction with $h(x)$.*
3. *If the prediction is "false" and y_i is "true", remove all the features, which given x_i are false, so that after this $h(x_i)$ will be true.*
4. *If the prediction is "true" and y_i is "false", then return "no consistent hypothesis".*
5. *Return to the step 2.*

The above algorithm will make at most n mistakes. Since the first mistake will shrink down the size of h to n, and the forthcoming mistakes will reduce the size of the h, by at most one.

It can also be shown that, there is no deterministic algorithm to learn this hypothesis class with mistake bound less than n. Consider a training set, in which all of the bits, except the i-th are zero. If there is a deterministic algorithm which correctly predicts the label for the i-th instance. One can create another data, and the label of i-th instance flipped, and the algorithm will make mistake on that. Since this argument can be repeated for any $i = 1, ..., n$, this proves that the mistake bound for any deterministic algorithm is at least n.

Exercise 4 (MB for disjunctions)**.** *Suppose our hypothesis class is conjunction of features. Given a training set S, how can we learn with this hypothesis class? Present an mistake-bounded algorithm, and find its mistake bound.*
Hint: Use a similar argument to the previous example, and show the the mistake bound in n.

Remark 10. *An algorithm A is called "conservative", when it updates its current state, when it makes mistake.*

Proposition 2. *MB-learnabality implies PAC-learnability!*

Remark 11. *Fancier transformation could give bound* $O\left(\frac{1}{\epsilon}\left[M + \ln(1/\delta)\right]\right)$*!*

Exercise 5 (Lazy decision list)**.** *Suppose a set of learning data is given. Find a consistent decision list with decision bound* $O(nL)$ *in a lazy fashion, where* L *is the size of the target decision list. Generalize this to learning* k*-decision lists with mistake bound* $O(n^k L)$*.*

Proof. There are multiple answers to this question. Here we present two answers.
Answer 1:
Create a list of all possible rules:

$$\begin{cases} x_1 \Rightarrow T \\ \bar{x}_1 \Rightarrow T \\ x_1 \Rightarrow F \\ \bar{x}_1 \Rightarrow F \\ \vdots \end{cases}$$

Also, call the two dummy rules $P = \{T \Rightarrow T, T \Rightarrow F\}$. In overall the number of the rules is $O(4n + 2)$. Now run the following algorithm:

1. Create a decision list with the rules created, with arbitrary order, call it Q. Create a copy of Q and call it Q'.
2. Loop over the following steps, until there is no inconsistency
   (a) Loop over the instances, and find the first rule that fires.
   (b) If the rule is consistent with the label remove it.
   (c) If at some point there is no rule fired, add Q' to the end of the list, and continue removing the inconsistent rules.
   (d) If after adding Q' no rule gets fired, add the two rules in C to the end of the list, and keep the consistent one.

Now we need to prove that this would result in a valid decision list (exercise). To show that this would have mistake bound of $O(nL)$, you just need to show adding new Q to the end of the decision list, wouldn't repeat more than L times(exercise).

Answer 2:
Consider the same rules we have before in Q and the dummy rules in Q'. Consider the union of the rules in $U = Q \cup Q'$. The necessity of having these rules will become clear in the correctness analysis. In overall the number of the rules is $O(4n + 2)$. In this algorithm we will have a notion called *level*, which is defined in the following form. Suppose we start with the the decision list Q, and remove rule r, and create a new level:

$$Q \qquad \text{converted to} \qquad \begin{cases} Q \setminus \{r\} \\ \text{else} \\ \{r\} \end{cases}$$

This can be repeated many times to get multiple levels. The representation of levels is just to ease the analysis. The algorithm is the following:

1. Create a decision list with the rules created, with arbitrary order.
2. Loop over the following steps, until there is no inconsistency
   (a) Loop over the instances, and find the first rule that fires.
   (b) If the rule is consistent with the label, move it to the next level.

First prove that, choosing the subset of the rules, until the end of the first trivial rule (of the rules in Q') would give a valid consistent decision list (exercise). Then prove that the mistake bound is $O(nL)$. To prove that, show that the resulting answer will not have more than L levels (exercise). Extending the above answers to the k-decision list, is a trivial extension. □

Remark 12. *In PAC model, we can just pick any consistent model. Can we do the same thing for MC models?*

2.4 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?).

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. If the number of mistakes the best expert makes in the hindside is OPT, this algorithm would make $O((OPT + 1) \log n)$.

2.5 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. 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 6. *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. 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.

2.6 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 ϵ:

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

Proof. Suppose at time t, α_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 13. *The bound in the Equation 3 could be written in the following form:*

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

To make tighter bound (and assuming that we know OPT, and $OPT \geq logn$), we set $\epsilon = (logn/OPT)^{1/2}$. With simplification 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 $T \rightarrow +\infty$, the regret goes to zero. Algorithms with this property are usually called "no-regret" algorithms.

3 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.

3.1 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, then $w[i] \leftarrow 2 \times w[i]$, for all i.
  - If $h(x_j) \neq y_j$ and y_j, 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:

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

□

3.2 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 ϵn. 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:

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

Combining Equations 4 and 5 we get the following:

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

□

Problem 3. *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:

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

[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.

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

$$\begin{aligned}
\Phi^{t+1} &= \sum_i w^*[i] \log_{1+\epsilon} w^{t+1}[i] = \sum_i w^*[i] \log_{1+\epsilon} \left(w^t[i](1+\epsilon)^{yx[i]} \right) \\
&= \sum_i w^*[i] \log_{1+\epsilon} \left(w^t[i](1+\epsilon)^{yx[i]} \right) \\
&= \sum_i w^*[i] \log_{1+\epsilon} w^t[i] + \sum_i yx[i] = \Phi^t + \sum_i y.w^*[i].x[i] \\
&\Rightarrow \Phi^{t+1} = \Phi^t + \sum_i y.w^*[i].x[i] \qquad (6)
\end{aligned}$$

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

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

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:

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

Also from the original analysis, remember the inequality:

$$n + M_+ \epsilon n - M_- \frac{\epsilon}{1+\epsilon} n \geq 0 \qquad (8)$$

Combining these two inequalities proper choice of ϵ as a function of γ, 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 14. *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.*

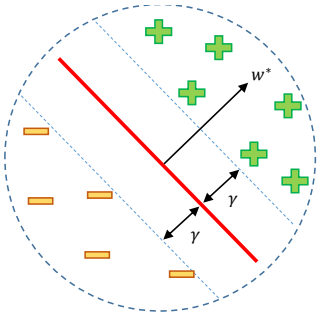

Figure 2: A representation of Perceptron Algorithm updates, and its margin.

4 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,
$$w \leftarrow w - x$$

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 γ.

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

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 (updates),

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

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 $M \leq \frac{1}{\gamma^2}$. □

Remark 15. *One way of looking at the perceptron updates is optimizing an objective function via gradient descent updates. Specifically if you define the following objective function based on the hinge loss on a given daataset:*

$$F(w) = \frac{1}{T} \sum_{t=1}^{T} \max(0, -y_t(w.x_t))$$

The gradient descent updates for minimizing this objective would result in the following updates:

$$w_{t+1} \leftarrow \begin{cases} w_t + \eta y_t x_t & y_1(w.x_t) < 0 \\ w_t & o.w. \end{cases}$$

One can prove that the mistake /update bound for the perceptron is optimal in terms γ. The next theorem is proving this.

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

Proof. For proof, we design a set of $1/\gamma^2$ data points, such that they always satisfy the margin separation of γ, 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 γ (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) γ (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, γ. 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 4. *Remember the Problem 3. 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 $O((l_2(w^*)l_2(X)/\gamma)^2)$.

- Initialize $\boldsymbol{\alpha}$ of lenght T
- for each $t \leftarrow 1$ to T
  - Receive x_t
  - $\hat{y}_t \leftarrow \text{sign}\left(\sum_{s=1}^{T} \alpha_s y_s (x_s.x_t)\right)$
  - Receive y_t
  - Update α_t:

$$\alpha_t \leftarrow \begin{cases} \alpha_t + 1 & \hat{y}_t \neq y_t \\ \alpha_t & o.w. \end{cases}$$

- Return $\boldsymbol{\alpha}$

Remark 16. *One cam make the above dual percentron kernelized (aka Kernel-Perceptron) by simply replacing the dot product $x_s.x_t$ with a kernel evaluation $K(x_s, x_t)$.*

References

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

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

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

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