Everything I Know about Learning Theory

Gene Li*

November 3, 2021

Contents

*gene@ttic.edu

1 Introduction

The fundamental goal of machine learning is to devise algorithms which learn from some observations to complete some task: may it be predicting some output, behaving intelligently, or controlling some system.

In these notes, we develop theory for machine learning, allowing us to give precise answers to such questions like:

- What does it mean for something to be "learnable"?
- How much data do we need, and what kind of data do we need?
- How do we develop algorithms for learning which are both *space* and *time* efficient?

Machine learning theory is closely intertwined with related fields, borrowing techniques and practice from algorithms, complexity theory, statistics, and information theory.

These notes are a compilation of course notes from various machine learning courses I have taken, include COS 511 at Princeton University and TTIC 31250 at TTIC.

Note: some topics from these courses have been omitted. For example, I did not scribe notes on bandits/RL problems and differential privacy. However, there are many resources available on the internet for these topics.

2 PAC Learning Framework

As a guiding example, imagine you want to write a computer program that decides which email messages are spam and which are ham. Our algorithm will take a sample of data, which have already been labeled "spam/ham", and return a classifier rule that predicts on unseen data. How well will we do? In this section, we will develop general purpose learning algorithms and mathematically analyze how well they "generalize", or perform on unseen data.

2.1 Preliminaries

We lay some groundwork for this section by introducing some terminology and notation.

For now, let's consider the learning problem described as follows. We are given some **examples** of data, which we want to predict a **label** from. How do we do this?

More precisely, let us have access to labeled examples $\{(x_i, y_i)\}_{i\in[m]}$. The **examples** x_i belong to the **domain space** $\mathcal{X}$. There are two possible labels for y_i, and without loss of generality we assume $y_i \in \{0, 1\}$.

We assume that there exists some **concept** $f : X \to \{0, 1\}$. Futhermore, we assume $f \in \mathcal{C}$ comes from a **concept class** $\mathcal{C}$. The concept f is unknown to us; our task is to devise a learning algorithm which learns the correct f given the fact that (i) our examples are perfectly labeled by c, i.e. $y_i = f(x_i)$; (ii) we know the concept class $\mathcal{C}$.

2.2 Consistency Model

As a first attempt, we introduce the consistency model, which can be summarized as "pick any concept which is consistent with the labeled samples".

Definition 1. We say that a concept class $\mathcal{C}$ is **learnable in the consistency model (CM)** if there exists an algorithm A that takes $\{(x_i, y_i)\}_{i\in[m]}$ and outputs $f \in \mathcal{C}$ which is consistent with all examples (such $f(x_i) = y_i$), or outputs that no such concept exists.

Example 1 (Learning monotone conjunctions)**.** Let the domain space be $\mathcal{X} = \{0,1\}^n$. Let the concept class be the set of monotone conjunctions, i.e. the AND of a subset of nonn-negated variables. For example, a valid concept is $f(x) = x_1 \wedge x_5 \wedge x_6$.

Monotone conjunctions can be learned in the consistency model via the following algorithm. Consider the positive training examples with $y_i = 1$. We set the output concept f' to be the monotone conjunction consisting of the indices where all the examples have a 1 bit. If f is also consistent on the negative examples, we return f'. Otherwise, we claim that no consistent concept exists.

As a remark, note that f', the concept outputted by our algorithm, must be a subset of the ground truth concept f.

Example 2 (Learning axis-aligned rectangles)**.** Let the domain space be $\mathcal{X} = \mathbb{R}^2$. Let the concept class be the set of all rectangles with sides parallel to the axis. A concept assigns $y_i = 1$ to all examples falling inside the rectangle, and $y_i = 0$ to all examples falling outside of the rectangle.

The concept class can be learned by the following algorithm. Draw the smallest possible rectangle that is consistent with the observed samples, and return it. We can do this by considering the top-most, left-most, right-most, and bottom-most positive examples. If no such rectangle exists, we claim that no consistent concept exists.

Of course, this is not the only algorithm to do it: we could have easily picked any rectangle which is consistent with the labeled example.

2.2.1 Issues with the consistency model

The consistency model is extremely simple. However, it does not give us a notion of "generalization" - how well our algorithm does on unseen examples. We want to have some control over how many samples we need in order to provably generalize how new data. In addition, it is unclear how to construct algorithms in the consistency model when we have noisy data.

2.3 PAC Learning Model

We introduce the **probably approximately correct (PAC) learning model**. The PAC learning model gives us finer grain control of the notion of generalization and sample complexity.

For PAC learning, we make the following assumptions:

1. Both the training examples and the test examples are drawn i.i.d. from some unknown target distribution $\mathcal{D}$ over $\mathcal{X}$.
2. All examples are labeled according to some unknown target concept $f : \mathcal{X} \to \{0, 1\}$, which is a member of a (known) concept class $\mathcal{C}$.
3. Our algorithm finds a hypothesis $h \in \mathcal{H}$, where $\mathcal{H}$ is called the **hypothesis class**. The hypothesis class does not have to be the same as the concept class!

Now we can define the generalization error, a notion of how well an algorithm does on unseen data. The assumptions about the drawing of examples allows us to give this probabilistic notion of how well an outputted hypothesis performs.

Definition 2. The **generalization error** is the probability of misclassifying a new example:

$$\text{err}_{\mathcal{D}}(h) := \mathbb{P}_{x \sim D}[h(x) \neq f(x)].$$

Definition 3. A concept class $\mathcal{C}$ is **PAC-learnable by** $\mathcal{H}$ if, for all $f \in \mathcal{C}$, any distribution $\mathcal{D}$, any constants $\epsilon > 0$, $\delta > 0$, there exists an algorithm A which takes a set S of $m = \text{poly}(1/\epsilon, 1/\delta)$ training examples, using time $t = \text{poly}(1/\epsilon, 1/\delta)$ and outputs a hypothesis $h \in \mathcal{H}$ with low generalization error:

$$\mathbb{P}[\text{err}_{\mathcal{D}}(h) \leq \epsilon] \geq 1 - \delta. \tag{1}$$

There are a lot of working parts in this definition. Let's break it down.

First, we note that in some sense, this is a *worst case* notion: we require that our algorithm produce "good" hypothesis for all the concepts in $\mathcal{C}$ and for any arbitrarily bad distribution $\mathcal{D}$. We can think of this as having an adversary picking distributions over the space $\mathcal{X}$ and concepts f. Once f and $\mathcal{D}$ are set, the training examples $S = \{(x_i, y_i)\}_{i \in [m]}$ are i.i.d. drawn from $\mathcal{D}$.

The constants ϵ and δ control the amount of "error" that our algorithm is afforded. Eq. (1) is saying: with probability at least $1 - \delta$, our algorithm will output a hypothesis which makes mistakes on ϵ fraction of datapoints.

This definition allows us to derive sample complexity bounds. If we want higher accuracy (corresponding to larger values of ϵ or δ), then we are allowed larger training sample sizes. However, the requirement that $m = \text{poly}(1/\epsilon, 1/\delta)$ ensures algorithmic efficiency - we aren't allowed an arbitrarily larger number of examples.

2.3.1 Some illustrative examples

Example 3 (Learning half-lines)**.** We consider the following canonical example. Let the domain space be $\mathcal{X} = \mathbb{R}$. Let the concept class be positive half-lines, i.e. Picture?

$$\mathcal{C} = \{[x, \infty) : x \in \mathbb{R}\}. \tag{2}$$

Let us denote the concept as well as the threshold value by f. When sample points $x_i \geq f$, the concept labels them as positive, and when the sample points $x_i < f$, the concept labels them as negative. We would like to determine whether this concept class is PAC-learnable by $\mathcal{H} = C$.

Consider any consistent hypothesis h which falls between the largest negative example and smallest positive example. We see that h makes mistakes on the error region $[f, h]$ (for sake of illustration, assuming $h > f$). For PAC-learnability, we must have that this error region has probability mass $< \epsilon$, and we say that h is "ϵ-good". The hypothesis h is "ϵ-bad" if h is too far to the left or right of the ground truth f, i.e. the interval $[h, f]$ or $[f, h]$ respectively has probability mass with respect to $\mathcal{D}$ of $> \epsilon$.

We will show that this happens with low probability. Let us define the two bad events B_- and B_+:

$$B_- := \{\text{h is too far to left of f}\},$$
$$B_+ := \{\text{h is too far to right of f}\}.$$

Let's try estimating $\mathbb{P}[B_+]$. Imagine a point r_+ starting at c, sweeping right until the interval $R_+ := [f, r_+]$ has probability mass ϵ. If any training examples fall in R_+, then $h < r_+$ and B_+ does not occur. (Why?) Therefore:

$$\begin{aligned}
\mathbb{P}[B_+] &\leq \mathbb{P}[\cap_{i=1}^m \{x_i \notin R_+\}] \\
&= \prod_{i=1}^m \mathbb{P}[x_i \notin R_+] && \text{(by independence)} \\
&= (1-\epsilon)^m. && \text{(definition of } R_+\text{)}
\end{aligned}$$

Here, we used the fact that the examples x_i are drawn i.i.d. and the definition of R_+ having probability mass of ϵ.

By a similar argument, $\mathbb{P}[B_-] \leq (1-\epsilon)^m$ also, so then:

$$\mathbb{P}[\text{h is } \epsilon\text{-bad}] = \mathbb{P}[B_- \cup B_+] \leq \mathbb{P}[B_-] + \mathbb{P}[B_+] \leq 2(1-\epsilon)^m \leq 2e^{-\epsilon m}.$$

Here we use union bound and the fact that $\forall x$, $1 + x \leq e^x$.

We want this error probability to be at most δ. After some computation, we see that if our sample size satisfies $m \geq \frac{1}{\epsilon} \ln \frac{2}{\delta}$, we have PAC-learnability by $\mathcal{H}$: $\mathbb{P}[\text{err}_{\mathcal{D}}(h) > \epsilon] \leq \delta$.

Example 4 (Learning intervals)**.** Now let us consider learning closed intervals of the form $[a, b]$, where $-\infty \leq a < b \leq \infty$. We can run through a similar argument as above, however the argument is modified to consider both the bad events on the left endpoint and the right endpoint. This yields that $\mathcal{C}$ is PAC-learnable by $\mathcal{H} = C$, with a larger sample size that satisfies $m \geq \frac{1}{\epsilon} \ln \frac{4}{\delta}$.

picture?

Example 5 (Learning axis-aligned rectangles). Let us consider the concept class picture?

$$\mathcal{C} = \{\text{axis-aligned rectangles}\}$$

and set $\mathcal{H} = C$. We will use the algorithm A which finds the smallest consistent rectangle $h \in \mathcal{H}$.

We sweep out 4 bands within f along the top, right, bottom, and left edges, specifying that each band has probability mass exactly $\epsilon/4$. If at least one point falls into each band, then h will be ϵ-good. By similar techniques as above, we can see that $\mathcal{C}$ is PAC-learnable by $\mathcal{H}$. (Try it!)

3 Occam's Razor: Learning with Finite Hypotheses

The previous examples have demonstrated how to prove that an algorithm accomplishes PAC-learning for a given class $\mathcal{C}$. However, we would like a general method for showing whether particular learning problems are PAC-learnable or not. In addition, note that in the previous examples, finding a consistent hypothesis was sufficient for PAC-learning - is this always true? Our rough intuition is thus:

> Learning is possible if, given enough data, we can find a consistent hypothesis that is simple.

This is essentially what's known in philosophy as **Occam's Razor**.

In this section, we will show why Occam's Razor holds in the PAC learning model. For now, our notion of a hypothesis being "simple" will be that the hypothesis class is finite, i.e. $|H| < \infty$.

Note that even in the examples PAC learning was possible with *infinite* hypothesis classes. (Even $\mathcal{C}$ in Eq. (2) is uncountably large!) Later, we will develop more complicated machinery that will allow us to reason about infinite hypothesis classes while preserving some notion that our hypothesis class is "simple".

3.1 CM Learning $\Rightarrow$ PAC Learning

We will show the following theorem, which implies that if an algorithm can find a consistent hypothesis, then PAC learning is possible.

Theorem 1 (Occam's Razor). *Say algorithm A finds hypothesis $h_A \in \mathcal{H}$ consistent with m examples, where $m \geq \frac{1}{\epsilon}(\ln|H| + \ln\frac{1}{\delta})$. Then*

$$\mathbb{P}[err_{\mathcal{D}}(h_A) > \epsilon] \leq \delta.$$

Equivalently, with probability at least $1-\delta$, if $h_A \in \mathcal{H}$ is consistent, then its generalization error is bounded as:

$$err_{\mathcal{D}}(h_A) \leq \frac{\ln|\mathcal{H}| + \ln\frac{1}{\delta}}{m}.$$

Note that the generalization error decreases as m grows. It also decreases if we select consistent hypotheses from smaller hypothesis classes. This corresponds to the previous intuition discussed at the beginning of this section.

Proof. We will show that the probability that a consistent hypothesis is ϵ-bad is upper-bounded by δ.

Let us denote the set of ϵ-bad hypotheses as $\mathcal{B} \subseteq \mathcal{H}$. Then for any $h \in \mathcal{B}$, we have:

$$\begin{aligned}
\mathbb{P}[h \text{ consistent}] &= \mathbb{P}[\cap_i \{h(x_i) = f(x_i)\}] \\
&= \prod_{i=1}^{m} \mathbb{P}[h(x_i) = f(x_i)] && \text{(by independence)} \\
&= (1-\epsilon)^m && \text{(because } h \text{ is } \epsilon\text{-bad)} \\
&= e^{-\epsilon m}.
\end{aligned}$$

Now, we calculate the probability that a consistent hypothesis h_A is ϵ-bad.

$$\begin{aligned}
\mathbb{P}[h_A \text{ consistent \& } \epsilon\text{-bad}] &\leq \mathbb{P}[\exists h \in \mathcal{H} : h \text{ consistent \& } \epsilon\text{-bad}] \\
&= \mathbb{P}[\exists h \in \mathcal{B} : h \text{ consistent \& } \epsilon\text{-bad}] \\
&\leq \sum_{h \in \mathcal{B}} \mathbb{P}[h \text{ consistent}] && \text{(union-bound)} \\
&\leq |\mathcal{B}| e^{-\epsilon m} \\
&\leq |\mathcal{H}| e^{-\epsilon m} \leq \delta.
\end{aligned}$$

□

Should we include the "too good to be true proof"

Example 6. This theorem gives us a general tool for reasoning about whether learning problems are PAC learnable. For example, going back to the monotone conjunctions problem, we see that $\mathcal{H} = 2^n$, so immediately we can see that with $m \geq \frac{1}{\epsilon}(n \ln 2 + \ln \frac{1}{\delta})$, monotone conjunctions are PAC-learnable.

Example 7 (Learning decision lists)**.** Consider learning decision lists (DLs) on n bits. An algorithm running on S can be written as follows:

- Start with an empty list.
- Find an if-then rule consistent with the data, satisfying at least one example.
- Add it to our list, and cross off examples which are "covered".

This algorithm can never get stuck if our data was generated by a valid DL: if this happened, then there would be no DL consistent with the remaining data, and therefore no DL consistent on the original data.

We can give a PAC guarantee using Occam's Razor. We note that the total number of DLs over n bits is can be upper bounded as $|\mathcal{C}| \leq n! \times 2 \times 4^n$. Therefore, we can conclude that if we get $m \geq \frac{1}{\varepsilon}\left(n \log n + \log \frac{1}{\delta}\right)$, we have PAC learning.

Example 8 (Description Length)**.** A CS viewpoint of this guarantee. Suppose we know that all the possible explanations of our data were generated by a sequence of $s := \text{size}(f)$ bits, so there are at most 2^s explanations. Then Occam's Razor says we need $m \geq \frac{1}{\varepsilon}\left[s \log 2 + \log \frac{1}{\delta}\right]$ samples.

Example 9 (Other classes we can/can't PAC-learn)**.**

1. Decision trees over $\{0,1\}^n$ are not known to be PAC-learnable in time, samples polynomial in size of size(f).
2. AND-functions, OR-functions, k-DLs (special cases/generalizations of DLs).

3.2 PAC Learning $\Rightarrow$ CM Learning

Now we go the other way: showing that *proper* PAC learnability implies that we are always able to find a consistent hypothesis. By proper, we mean that the hypothesis class is the same as the concept class, i.e. $\mathcal{H} = C$.

Theorem 2. *If we can do proper PAC learning, then we can do learning in the consistency model.*

Proof. Suppose we have an algorithm A which properly PAC learns $\mathcal{C}$. We are given a set of labeled examples $S = \{(x_i, y_i)\}_{i=1}^m$. We want to show that A can find a concept in $\mathcal{C}$ consistent with S or claim that no concept exists.

Let us construct $\mathcal{D}$ as the uniform distribution over S. Let us chose $\epsilon = \frac{1}{2m}$. Then we let A draw i.i.d samples from S under the uniform distribution $\mathcal{D}$. If the algorithm outputs h_A which is consistent with S, we return, otherwise claim there is no consistent concept in $\mathcal{C}$.

Why does this work? Clearly if no consistent f exists, then we claim correctly so. Now suppose there exists a consistent concept $f \in \mathcal{C}$. From the definition of PAC learning, w.p. at least $1-\delta$ we find $h \in C$ such that $\text{err}_{\mathcal{D}}(h) \leq \epsilon < \frac{1}{m}$. The hypothesis h must be consistent on S because if h makes any mistakes on S, $\text{err}_{\mathcal{D}}(h) \geq \frac{1}{m}$, a contradiction. □

3.3 PAC Learning without Computation Time

If we instead modified our PAC learning definition to remove the requirement that the algorithm must run in poly-time, then any finite hypothesis class is PAC-learnable. We justify this as follows.

We can run an algorithm which guesses the size(f) iteratively, and stops when it reaches something which is consistent:

1. Set $s_1 = 10, \delta_1 = \delta/2$. For $i = 1, 2, \ldots$ do:
    - Request $\frac{1}{\epsilon}\left(s_i + \log \frac{1}{\delta}\right)$ examples.

- Check if there is a function of size at most s_i consistent with these samples. If so, output and halt.
- Otherwise let $s_{i+1} = 2s_i, \delta_{i+1} = \delta_i/2$.

2. Note that the total chance of failure is at most $\sum \delta_i \leq \delta$ chance of failure.
3. The total amount of data used is: $\mathcal{O}\left(\frac{1}{\varepsilon}\left(\text{size}(f) + \log(\text{size}(c)) \log \frac{1}{\delta}\right)\right)$. (This needs to be carefully worked out, but is straightforward.)

4 Learning with Infinite Hypotheses

Previously, we have shown that finite hypotheses classes are PAC-learnable, provided we can find a consistent hypothesis $h \in \mathcal{H}$. However, a more realistic scenario is when we have an infinite hypothesis class. For example, the simple examples of learning positive half lines, intervals, and axis aligned rectangles are all examples of infinite hypothesis classes. We showed ad hoc method for proving that these classes are PAC-learnable. In this section, we will give more general methods for proving that infinite hypothesis classes are PAC-learnable in terms of (i) the growth function, and (ii) VC dimension.

4.1 Growth Functions

Intuitively, if many hypotheses in H are very similar, we shouldn't have to pay in terms of the cardinality $|H|$. Our techniques for dealing with infinite hypothesis classes begin with a crucial observation:

> For a sample S with m datapoints, there are a finite amount of labelings which a hypothesis class $\mathcal{H}$ can produce.

This is trivially true, after all, we have at most 2^m possible labelings. However, often the number of labelings can be much smaller: for example, when our hypothesis class is positive half lines, we only have $m + 1$ possible labelings!

This motivates our approach.

Definition 4 (Growth Function)**.** Denote the set of possible labelings of a sample as $\Pi_{\mathcal{H}}(S) := \{(h(x_1), ..., h(x_m)) : h \in H\}$. Then the **growth function** of $\mathcal{H}$ is defined as:

$$\Pi_{\mathcal{H}}(m) := \max_{S:|S|=m} |\Pi_{\mathcal{H}}(S)|.$$

That is, it is the largest possible cardinality of a set of possible labelings over all samples S of size m.

Example 10. The growth function of the class of positive half-lines is $\Pi_{\mathcal{H}}(m) = m + 1$. For intervals, it is $\Pi_{\mathcal{H}}(m) = \binom{m}{2} + m + 1$.

Some more intuition: we will later show that there are two possible cases for the growth function:

- The "nice" case: the growth function is polynomial in m, i.e. $\Pi_{\mathcal{H}}(m) = O(m^d)$. We will call d the *VC dimension*.
- The "worst" case: the growth function is exponential: $\Pi_{\mathcal{H}}(m) = 2^m$. This means that for sample size m, the hypothesis class could allow for all possible 2^m labelings.

4.2 Generalized Occam's Razor

Now, we will show a generalized Occam's Razor theorem, where we have simply replaced $|H|$ with the growth function $\Pi_{\mathcal{H}}(2m)$ (the reason for the factor of 2 will be explained in the proof).

Theorem 3 (Generalized Occam's Razor)**.** *With probability at least $1-\delta$, if $h_A \in \mathcal{H}$ is consistent, then its true generalization error is bounded as:*

$$err_{\mathcal{D}}(h_A) \leq \epsilon = O\left(\frac{\ln \Pi_{\mathcal{H}}(2m) + \ln \frac{1}{\delta}}{m}\right).$$

Proof. Let us denote B as the event that there exists $h \in \mathcal{H}$ which is consistent under sample S but is ϵ-bad. Our task is to upper bound $\mathbb{P}[B]$. Let's also define:

$$M(h, S) := \text{\# mistakes of } h \text{ on } S$$

We will employ a *ghost sample* technique. We will draw a second sample of size m under $\mathcal{D}$, let's call it S'. We will show that if a hypothesis is good on S, then it is unlikely that it will be bad on S'. Under our notation:

$$B := \{\exists h \in \mathcal{H} : M(h, S) = 0 \text{ and } \epsilon\text{-bad}\}$$
$$B' := \{\exists h \in \mathcal{H} : M(h, S) = 0 \text{ and } M(h, S') \geq \frac{m\epsilon}{2}\}.$$

Note that $\mathbb{P}[B'|B] \geq 1/2$, via a Chernoff bound argument. Then: elaborate.

$$\mathbb{P}[B'] \geq \mathbb{P}[B' \cap B] = \mathbb{P}[B]\mathbb{P}[B'|B] \geq \frac{1}{2}\mathbb{P}[B].$$

Next, we use a permutation trick. For each index $i \in [m]$, we flip a coin and swap $x_i \in S$ and $x'_i \in S'$ w.p. $\frac{1}{2}$. Call the new datasets T and T', and let us define a new event:

$$B'' := \{\exists h \in H : M(h, T) = 0 \text{ and } M(h, T') \geq \frac{m\epsilon}{2}\}.$$

Let us further define a version of B'' for a fixed hypothesis:

$$b(h) := \{M(h, T) = 0 \text{ and } M(h, T') \geq \frac{m\epsilon}{2}\}$$

Because (S, S') and (T, T') have the same distribution, $\mathbb{P}[B'] = \mathbb{P}[B'']$.

We will claim the following.

Claim. For any S, S', we have $\mathbb{P}[b(h)|S, S'] \leq 2^{-m\epsilon/2}$.

We compute that:

$$\begin{aligned}
\mathbb{P}[B''] &= \mathbb{E}_{S,S'}[\mathbb{P}[B''|S, S']] && \text{(marginalizing over all samples } S, S') \\
&\leq \sup_{S,S'} \mathbb{P}[B''|S, S'] \\
&= \sup_{S,S'} \mathbb{P}[\exists h \in \mathcal{H} : b(h)|S, S'] && \text{(recalling the definition of } B'') \\
&\leq \sup_{S,S'} \mathbb{P}[\exists h \in \mathcal{H}' : b(h)|S, S'] && \text{(let } \mathcal{H}' \subseteq \mathcal{H} \text{ be the subset where for each labeling in } S \cup S', \text{ we have selected one representative hypothesis from } \mathcal{H}) \\
&\leq \sup_{S,S'} \sum_{h \in \mathcal{H}'} \mathbb{P}[b(h)|S, S'] && \text{(union-bound)} \\
&\leq |\mathcal{H}'| 2^{-m\epsilon/2} && \text{(claim)} \\
&\leq \Pi_{\mathcal{H}}(2m) 2^{-m\epsilon/2}.
\end{aligned}$$

Therefore, we see that

$$\mathbb{P}[B] \leq 2\Pi_{\mathcal{H}}(2m) 2^{-m\epsilon/2}.$$

In order for this to be $\leq \delta$, we must have:

$$\epsilon = \frac{2}{m}(\lg \Pi_{\mathcal{H}}(2m) + \lg \frac{1}{\delta} + 1) = O\left(\frac{\ln \Pi_{\mathcal{H}}(2m) + \ln \frac{1}{\delta}}{m}\right).$$

Thus, we are left with showing the claim that $\mathbb{P}[b(h)|S, S'] \leq 2^{-m\epsilon/2}$.

For each sample x_i, x_i', lets denote 1 if h makes a mistake, and 0 if h is correct.

For each index i, we consider the following cases:

- $x_i = x_i' = 1$. In this case, $\mathbb{P}[b(h)|S, S'] = 0$ because no matter how you swap, h makes a mistake on T.
- $x_i = x_i' = 0$. These indices can be ignored because swapping always is mistake-free.
- $x_i \neq x_i'$. Lets denote $r = |\{i \in [m] : x_i \neq x_i'\}|$. If $r < \frac{m\epsilon}{2}$, then clearly no matter how we swap, we cannot have $M(h, T') \geq \frac{m\epsilon}{2}$. So $\mathbb{P}[b(h)|S, S'] = 0$. If $r \geq \frac{m\epsilon}{2}$, then we need to flip the errors such that all of them are in T', and thus:

$$\mathbb{P}[b(h)|S, S'] = 2^{-r} \leq 2^{-m\epsilon/2}.$$

Thus in all cases, the claim is true. □

Note: a similar result applies for the agnostic setting in terms of uniform convergence with a rate of $1/\varepsilon^2$ instead of $1/\varepsilon$.

4.3 VC Dimension

Now we will investigate further how to quantify the size of a hypothesis class $\mathcal{H}$, in terms of the Vapnik-Chervonenkis (VC) dimension. This will allow us to derive bounds on the growth function in terms of the VC dimension.

First, we introduce the concept of shattering.

Definition 5 (Shattering)**.** A set S of size m is shattered by $\mathcal{H}$ if $|\Pi H(S)| = 2^m$, i.e. all possible labelings of the set S are realized by functions in $\mathcal{H}$.

Definition 6 (VC-Dimension)**.** $VC(\mathcal{H}) =$ cardinality of the largest set shattered by $\mathcal{H}$.

Example 11. The VC dimension of positive half-lines is 2.

Example 12. The VC dimension of axis-aligned rectangles is 4.

Example 13. The VC dimension of linear threshold functions in $\mathbb{R}^n$ is $n + 1$. The VC dimension of linear threshold functions in $\mathbb{R}^n$ which pass through the origin is n.

Claim. When $|H| < \infty$, we have $VC(\mathcal{H}) \leq \lg |H|$.

Proof. If the VC dimension is d, then there exists a shattered set of size d with 2^d ways of labeling it. Therefore, $2^d \leq |H|$ and the claim follows. □

4.4 Sauer's Lemma

Now, we will prove Sauer's Lemma, which relates the VC dimension to the growth function.

Lemma 4 (Sauer's Lemma)**.** *Let $\mathcal{H}$ be a hypothesis space with $VC(\mathcal{H}) = d$. Then*

$$\Pi_{\mathcal{H}}(m) \leq \Phi_d(m) := \sum_{i=0}^{d} \binom{m}{i}.$$

Proof. We proceed by induction on $m + d$. When $m = 0$, there is one possible labeling, i.e. $\Pi_{\mathcal{H}}(m) = 1 = \sum_{i=0}^{d} \binom{0}{i}$. When $d = 0$, there is a single label possible, thus $\Pi_{\mathcal{H}}(m) = 1 = \binom{m}{0}$.

So we can proceed with $m, d \geq 1$. We assume the lemma holds $\forall m', d'$ such $m' + d' < m + d$. For a fixed sample $S = (x_1, ..., x_m)$, we want to show that $|\Pi H(S)| \leq \Phi_d(m)$. Let us also set $S' = (x_1, ..., x_{m-1})$, the first $m - 1$ examples.

Recall that $\Phi_{\mathcal{H}}(S)$ is the set of distinct labelings $\mathcal{H}$ induces on S. Define $\mathcal{H}_1$ to be the set of distinct labelings $\mathcal{H}$ induces on S'. Define $\mathcal{H}_2$ to be the labelings on S' which correspond to when multiple labelings in $\Phi_{\mathcal{H}}(S)$ collapse to one label on S'. In other words, for any label $l \in \mathcal{H}_2$, (i) $l \in \mathcal{H}_1$ and (ii) $(l, 0), (l, 1) \in \Phi_{\mathcal{H}}(S)$.

Note that $|\Phi_{\mathcal{H}}(S)| = |H_1| + |H_2|$. In addition, we note the following:

1. $VC(\mathcal{H}_1) \leq d$. If $T \subseteq S'$ is shattered by $\mathcal{H}_1$, it is shattered by $\mathcal{H}$.

2. $VC(\mathcal{H}_2) \leq d-1$. If $T \subseteq S'$ is shattered by $\mathcal{H}_2$, $T \cup \{x_m\}$ is shattered by $\mathcal{H}$.

Therefore, we can conclude that $|H_1| = |\Pi_{H_1}(S')| \leq \Phi_d(m-1)$ and $|H_2| = |\Pi_{H_2}(S')| \leq \Phi_{d-1}(m-1)$.

Thus,

$$\begin{aligned}
|\Phi_{\mathcal{H}}(S)| &= |\mathcal{H}_1| + |\mathcal{H}_2| \\
&\leq \Phi_d(m-1) + \Phi_{d-1}(m-1) \\
&= \sum_{i=0}^{d} \binom{m-1}{i} + \sum_{i=0}^{d-1} \binom{m-1}{i} \\
&= \sum_{i=0}^{d} \binom{m-1}{i} + \sum_{i=0}^{d} \binom{m-1}{i-1} && \text{(reindexing)} \\
&= \sum_{i=0}^{d} \binom{m}{i} && \text{(well-known combinatorial identity)} \\
&= \Phi_d(m).
\end{aligned}$$

The inductive step holds, and we are done.

□

Note that $\Phi_d(m) = O(m^d)$. Often, we will employ this nifty bound on $\Phi_d(m)$.

Claim. If $m \geq d \geq 1$, $\Phi_d(m) \leq (\frac{em}{d})^d$.

Proof.

$$\begin{aligned}
\left(\frac{d}{m}\right)^d \sum_{i=0}^{d} \binom{m}{i} &\leq \sum_{i=0}^{d} \binom{m}{i} \left(\frac{d}{m}\right)^i && \text{(because } d \leq m\text{)} \\
&\leq \sum_{i=0}^{m} \binom{m}{i} \left(\frac{d}{m}\right)^i \\
&= \left(1 + \frac{d}{m}\right)^m && \text{(binomial expansion theorem)} \\
&\leq e^d.
\end{aligned}$$

Rearranging, the claim holds. □

Thus, it is clear that there are two cases:

- Finite VC dimension: $VC(\mathcal{H}) = d \Rightarrow \Phi_{\mathcal{H}}(m) = O(m^d)$.
- Infinite VC dimension: $VC(\mathcal{H}) = \infty \Rightarrow \Phi_{\mathcal{H}}(m) = 2^m$.

Lastly, by plugging in Sauer's Lemma into the generalized Occam's Razor theorem (Thm. 3), we get the following:

Theorem 5. *With probability at least $1-\delta$, if $h_A \in \mathcal{H}$ is consistent and $VC(\mathcal{H}) = d$, then its true generalization error is bounded as:*

$$err_{\mathcal{D}}(h_A) \leq \epsilon = O\left(\frac{d \ln \frac{m}{d} + \ln \frac{1}{\delta}}{m}\right).$$

In other words, if we have:

$$m = \mathcal{O}\left(\frac{1}{\varepsilon}\left(VC(\mathcal{H}) \log \frac{1}{\varepsilon} + \log \frac{1}{\delta}\right)\right).$$

then we can guarantee PAC learning.

4.5 Lower Bounds on Sample Complexity

Previously, we have shown results that $m = O()$ is sufficient for PAC learning. These are upper bounds on the sample complexity. However, we would also like lower bounds, too - for what sample size m is it impossible to do PAC learning? Intuitively, if we do not see enough points, we cannot know how to label the rest of the sample space. If our VC dimension is d, then if we only see a part of a shattered set, we should not be able to predict the rest of the examples because all of the labelings are possible.

Our task will be to show the following theorem:

Theorem 6. *For any algorithm A, there exists $f \in \mathcal{H}$ and distribution $\mathcal{D}$ such that if A gets a sample of size $m \leq d/2$, then:*

$$\mathbb{P}_S[err_{\mathcal{D}}(h_A) > 1/8] \geq 1/8.$$

This theorem implies that we cannot PAC-learn when $m \leq d/2$ for $\epsilon < 1/8$ and $\delta \leq 1/8$.

First, we give an incorrect proof, to illustrate a common pitfall.

Incorrect Proof. Let D be a uniform distribution on d points $z_1, ..., z_d$ which form a shattered set by $\mathcal{H}$. If we train algorithm A on S with $m \leq d/2$ examples which have been arbitrarily labeled, we get hypothesis h_A. Now pick $f \in \mathcal{H}$ such that it is consistent with labels in S and incorrect for $x \notin S$. Therefore, we have

$$\mathbb{P}_{\mathcal{D}}[f(x)s \neq h_A(x)] \geq 1/2.$$

□

This proof is incorrect because PAC learning requires us to pick $c \in \mathcal{C}$ *before* we select our sample and algorithm - here c has depended on the sample and h_A! Clearly, we need a stronger result.

Correct Proof. Again, let us choose a shattered set $z_1, ..., z_d$. Consider the class $\mathcal{C}' \subseteq \mathcal{C}$ which contains a representative for each labeling of the shattered set. We select $c \in \mathcal{C}'$ uniformly. We also let $\mathcal{D}$ be the uniform distribution over the shattered set.

Let's consider how to run the experiment.

1. The correct way under the PAC model: Pick c at random. Draw sample S under $\mathcal{D}$ and compute the labels by c. Use algorithm A to compute h_A. To test, draw x at random. Measure $\mathbb{P}[h_A(x) \neq c(x)]$.
2. The "cheating way". Choose S under $\mathcal{D}$. Compute random labels $c(x_i)$ for each $x_i \in S$. Use algorithm A to compute h_A. To test, draw x at random. If $x \notin S$ then give it a random label $c(x)$. Measure $\mathbb{P}[h_A(x) \neq c(x)]$.

These experiments are equivalent - specifically, note that in the second one, c is still chosen uniformly independent of h_A and S, albeit in a "lazy" fashion. We will work with the second experiment.

$$\begin{aligned}
\mathbb{P}_{c,S,x}[h_A(x) \neq c(x)] &\geq \mathbb{P}_{c,S,x}[x \neq S \cap h_A(x) \neq c(x)] && \text{(logically implied)} \\
&= \mathbb{P}_x[x \notin S]\mathbb{P}_{c,S}[h_A(x) \neq c(x) | x \notin S] \\
&\geq 1/2 * 1/2 = 1/4. && (c(x) \text{ randomly chosen})
\end{aligned}$$

Up to this point, we have shown that *on average*, we have high generalization error. Now we will show that such a concept $c \in \mathcal{C}'$ exists. (This is the so-called probabilistic method.)

Note that by definition of expectation:

$$\mathbb{E}_c[\mathbb{P}_{S,x}[h_A(x) \neq c(x)]] = \mathbb{P}_{c,S,x}[h_A(x) \neq c(x)] \geq 1/4.$$

Therefore, there exists c such that

$$\begin{aligned}
\mathbb{P}_{S,x}[h_A(x) \neq c(x)] &= \mathbb{E}_S[\mathbb{P}_x[h_A(x) \neq c(x)]] \\
&= \mathbb{E}_S[\text{err}_{\mathcal{D}}(h(A))] \\
&\geq 1/4.
\end{aligned}$$

Then we bound as follows:

$$\begin{aligned}
1/4 &\leq \mathbb{E}_S[\text{err}_{\mathcal{D}}(h(A))] \\
&= \mathbb{P}_S[\text{err}_{\mathcal{D}}(h(A) > 1/8] \cdot \mathbb{E}_S[\text{err}_{\mathcal{D}}(h(A)) | \text{err}_{\mathcal{D}}(h(A) > 1/8] \\
&\quad + \mathbb{P}_S[\text{err}_{\mathcal{D}}(h(A) \leq 1/8] \cdot \mathbb{E}_S[\text{err}_{\mathcal{D}}(h(A)) | \text{err}_{\mathcal{D}}(h(A) \leq 1/8] \\
&\leq \mathbb{P}_S[\text{err}_{\mathcal{D}}(h(A) > 1/8] \cdot 1 \\
&\quad + 1 \cdot 1/8
\end{aligned}$$

Rearranging, we have shown that for the fixed concept c, we have that

$$\mathbb{P}_S[\text{err}_{\mathcal{D}}(h(A) > 1/8] \geq 1/8.$$

□

An alternative statement and proof is presented below.

Theorem 7. *For any algorithm A, class $\mathcal{H}$, there exists distribution $\mathcal{D}$ and $f \in \mathcal{C}$ such that if $|S| < \frac{VC(\mathcal{H})-1}{8\varepsilon}$ then $\mathbb{E}[err_{\mathcal{D}}(A)] \geq \varepsilon$.*

Proof. Consider some $d = \text{VC}(\mathcal{H})$ shattered points. Define a distribution $\mathcal{D}$ with probability mass $1 - 4\varepsilon$ on one point and probability mass $\frac{4\varepsilon}{d-1}$ on the rest. Also pick a random labeling of the d points as the target.

$$\begin{aligned}
\mathbb{E}[\text{err}_{\mathcal{D}}(A)] &= \mathbb{P}[\text{mistake on test point}] \\
&\geq \frac{1}{2}\mathbb{P}[\text{test point not in } S] \\
&\geq \frac{1}{2}(4\varepsilon)\left(1 - \frac{4\varepsilon}{d-1}\right)^{|S|} \\
&\geq (2\varepsilon)\left(1 - \frac{4\,|S|\,\varepsilon}{d-1}\right) \\
&\geq \varepsilon.
\end{aligned}$$

□

5 Agnostic Learning, Empirical Risk Minimization

We provide a generalization of the PAC model, where we allow for inconsistencies in the hypothesis class. This is often called the *agnostic model*.

We define the distribution over the sample space *and* the labels, i.e. a sample (x, y) $\mathcal{D}$, where $\mathcal{D}$ is a distribution over $\mathcal{X} \times \{0, 1\}$. Then our error is redefined as:

$$\text{err}_{\mathcal{D}}(h) := \mathbb{P}_{(x,y)\sim D}(h(x) \neq y).$$

5.1 Bayes Optimal Hypothesis

Recall Bayes rule:

$$\mathbb{P}[x, y] = \mathbb{P}[x]\mathbb{P}[y|x].$$

In PAC model, it has the following meaning. We pick the sample point x first. Before, we chose the label deterministically as $y = c(x)$. Now, we instead choose y probabilistically,

with probabilities $\mathbb{P}[y = 1|x]$ and $\mathbb{P}[y = 0|x]$. Therefore, a optimal hypothesis h should have the property that:

$$h_{\text{opt}}(x) = \begin{cases} 1 & \text{if } \mathbb{P}[y = 1|x] > 1/2 \\ 0 & \text{if } \mathbb{P}[y = 1|x] < 1/2 \end{cases}$$

This is the so-called **Bayes optimal classifier**. The **Bayes error** is defined as:

$$\text{err}_{\mathcal{D}}(h_{\text{opt}}) = \min_h \text{err}_D(h).$$

5.2 Empirical Risk Minimization

In our previous PAC setting, we were interested in consistent hypothesis which were perfectly correct on the training data. Under this new framework, it may not be possible to be perfectly correct on the training data, especially if the class $\mathcal{H}$ is "restrictive" or "simple". So we need a notion of "training error":

Definition 7 (Training Error)**.** The **training error** of hypothesis h is defined as:

$$\hat{\text{err}}(h) := \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}\{\}\{h(x_i) \neq y_i\}.$$

This brings us to the notion of **empirical risk minimization (ERM)**: we hope that a hypothesis $\hat{h} := \arg\min_{h \in H} \hat{\text{err}}(h)$ which minimizes the training error will also have low generalization error $\text{err}_{\mathcal{D}}(\hat{h})$.

In order to show such behavior, we will explore the **uniform convergence** properties of $\mathcal{H}$, that is, with high probability $\forall h \in \mathcal{H} : |\hat{\text{err}}(h) - \text{err}_D(h)| \leq \epsilon$. Uniform convergence implies that our empirical risk minimizer $\hat{h}$ has low test error:

$$\begin{aligned} \text{err}_{\mathcal{D}}(\hat{h}) &\leq \hat{\text{err}}(\hat{h}) + \epsilon && \text{(by assumption)} \\ &\leq \hat{\text{err}}(h) + \epsilon \quad \forall h && \text{(definition of } \hat{h}) \\ &\leq \text{err}_{\mathcal{D}}(h) + 2\epsilon && \text{(by assumption)}. \end{aligned}$$

Therefore, $\hat{h}$ has generalization error within 2ϵ of the best hypothesis $h^\star \in \mathcal{H}$.

5.3 Showing Uniform Convergence

When $|H| < \infty$, we can show the following uniform convergence theorem:

Theorem 8. *Given m random examples from a distribution $\mathcal{D}$, with probability at least $1 - \delta$, the following holds $\forall h \in \mathcal{H}$:*

$$|\text{err}_D(h) - \hat{\text{err}}(h)| \leq \epsilon$$

if $m \geq \frac{1}{2\varepsilon^2} (\ln 2|H| + \ln 1/\delta)$.

Proof. We proceed by bounding the error for a fixed $h \in \mathcal{H}$, then applying union bound over all h. First, we can use the (two-sided) Hoeffding's inequality to show that:

$$\begin{aligned}\mathbb{P}[|\mathrm{err}_{\mathcal{D}}(h) - \hat{\mathrm{err}}(h)| \geq \epsilon] &= \mathbb{P}\left[|\mathbb{E}[\mathbb{1}\{h(x_i) \neq y_i\}] - \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}\{h(x_i) \neq y_i\}| > \epsilon\right] \\ &\leq 2e^{-2\epsilon^2 m}.\end{aligned}$$

Next, union bound gives:

$$\mathbb{P}[\exists h \in \mathcal{H}, |\mathrm{err}_D(h) - \hat{\mathrm{err}}(h)| \geq \epsilon] \leq 2|H|e^{-2\epsilon^2 m}.$$

Setting the RHS equal to δ gives the result. □

As a consequence of our theorem, we can bound the generalization error as:

$$\mathrm{err}_{\mathcal{D}}(h) \leq \hat{\mathrm{err}}(h) + O\left(\sqrt{\frac{\ln|H| + \ln 1/\delta}{m}}\right). \tag{3}$$

We note that this bound incorporates our intuition:

1. The lower the training error, the lower the generalization error.
2. The more examples we have, the lower the generalization error.
3. The more complicated $\mathcal{H}$ is, as measured by $|H|$, the more likely we are to overfit the data, increasing generalization error.

error vs complexity tradeoff graph?

Note that this bound is much worse than the bound we derived in the consistent case in Thm. 1. The difference lies in the fact that we have a rate that depends on $1/\epsilon^2$ now instead of ϵ. Going back and investigating our proof, we see that the ϵ^2 dependence comes from the use of Hoeffding's inequality. This is because we are asking something stronger: not just do we want to guarantee that "bad" hypotheses have high empirical error, but we want all hypotheses to have empirical error close to generalization error.

We can also use Chernoff bounds to get a rate of $1/\varepsilon$ (but the theorem is of slightly different flavor):

Theorem 9. *If* $m \geq \frac{6}{\varepsilon}\left(\log|H| + \log\frac{1}{\delta}\right)$*: then*

1. *all* $h \in H$ *with* $\mathrm{err}_D(h) > 2\varepsilon$ *have error* $\mathrm{err}_S(h) > \varepsilon$
2. *all* $h \in H$ *with* $\mathrm{err}_D(h) < \varepsilon/2$ *have error* $\mathrm{err}_S(h) < \varepsilon$.

Proof. Use a Chernoff bound for upper and lower bounds. Specifically, use the fact that

$$\begin{aligned}\mathbb{P}\left[\left|\frac{1}{m}\sum X_i - \mathbb{E}X\right| > p(1+\alpha)\right] &\leq e^{-m\mathbb{E}X\alpha^2/3} \\ \mathbb{P}\left[\left|\frac{1}{m}\sum X_i - \mathbb{E}X\right| < p(1-\alpha)\right] &\leq e^{-m\mathbb{E}X\alpha^2/8}.\end{aligned}$$

□

Note: Remember when we derived generalized Occam's Razor bounds for infinite hypotheses classes via the growth function and VC dimension? We can follow the same proof to get a sample complexity bound with growth function/VC dimension instead of $|\mathcal{H}|$ in the statement of Theorem.

Theorem 10. *For any class $\mathcal{H}$, distribution $\mathcal{D}$ over $X \times \{0, +1\}$ if*

$$m = |S| \geq \frac{8}{\varepsilon^2}\left\{\log \Pi_{\mathcal{H}}(2m) + \log \frac{2}{\delta}\right\}$$

then with probability at least $1 - \delta$, all $h \in \mathcal{H}$ have $|\mathsf{err}_{\mathcal{D}}(h) - \hat{\mathsf{err}}_S(h)| \leq \varepsilon$.

Proof. The proof essentially follows the proof of generalized Occam's Razor in Theorem 3. However, we define the "bad" event as the event that there exists $h \in \mathcal{H}$ with $\left|\hat{\mathsf{err}}'_S(h) - \hat{\mathsf{err}}_S(h)\right| \geq \varepsilon/2$. Then, we use Hoeffding's inequality to get the bound. □

add in this equation.

6 Rademacher Complexity

We will introduce a more sophisticated way to measure the complexity of the hypothesis space, which will allow us to derive bounds on generalization error.

Recall in the previous section we showed that uniform convergence is possible if $m \geq \frac{8}{\varepsilon^2}\left\{\log \Pi_{\mathcal{H}}(2m) + \log \frac{2}{\delta}\right\}$. It will be more convenient to write this as an error bound of ε in terms of m. Restated, we conclude that with high probability, all $h \in \mathcal{H}$ satisfy:

$$\mathsf{err}_{\mathcal{D}}(h) \leq \hat{\mathsf{err}}_S(h) + \sqrt{\frac{8\left(\log \Pi_{\mathcal{H}}(2m) + \log \frac{2}{\delta}\right)}{m}}.$$

Several issues to be aware of:

1. For certain classes $\mathcal{H}$, it may be hard to compute or estimate $\Pi_{\mathcal{H}}(m)$.
2. Our bounds have two sources of loss. (1) We computed a union bound over the labelings of the double sample S'', which is overly pessimistic if many labelings are similar. (2) We computed a worst case over S'', whereas we would want to do expected case or have a bound that depends on the actual training set.

Rademacher bounds get around this by computing a sample-dependent bound in terms of Rademacher complexity (to be shortly defined).

Hereafter, we remap labels from $\{0, 1\}$ to $\{-1, +1\}$. Note that this allows us to rewrite the training error expression as:

$$\hat{\mathsf{err}}(h) := \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}\{h(x_i) \neq y_i\} = \frac{1}{2} - \frac{1}{2m}\sum_{i=1}^{m} y_i h(x_i).$$

Therefore, minimizing the training error is equivalent to maximizing the quantity:

$$\max_{h \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^{m} y_i h(x_i).$$

Now, we consider the following experiment: instead of using the given labels y_i, we will use Rademacher random labels σ_i, independent from everything else. The intuition is that if our hypothesis class is rich, it will be able to fit to random labels well. Therefore, minimizing the training error too much by choosing a rich hypothesis class is equivalent to increasing the ability to fit to random noise.

As a warm up, let us consider the following quantity:

$$R := \mathbb{E}_\sigma \Big[\max_{h \in H} \frac{1}{m} \sum_{i=1}^{m} \sigma_i h(x_i) \Big]$$

Note that if $|\mathcal{H}| = 1$, we have $R = 0$ by linearity of expectation. If S is shattered by $\mathcal{H}$, we must have $R = 1$. Generally, $R \in [0, 1]$, with larger values corresponding to richer hypothesis classes.

We are now ready to define the Rademacher complexity.

Definition 8. Let $\mathcal{F}$ be a family of real-value functions, $f : Z \to \mathbb{R}$. Let $S = (z_1, ..., z_m)$ be a set of independent samples drawn from some distribution $\mathcal{D}$. Define the **empirical Rademacher complexity** as:

$$\hat{\mathcal{R}}_S(\mathcal{F}) = \mathbb{E}_\sigma \Big[\sup_{f \in F} \frac{1}{m} \sum_{i=1}^{m} \sigma_i f(z_i) \Big].$$

Define the **expected Rademacher complexity** as:

$$\mathcal{R}_m(\mathcal{F}) = \mathbb{E}[\hat{R}_S(F)].$$

Note that this definition generalizes the discussion we had previously, by allowing f to be an infinite class of real value functions.

6.1 Uniform Convergence Guarantees

Our task is to show uniform convergence for the function class $\mathcal{F}$, i.e. with high probability $\forall f \in \mathcal{F}$, the empirical mean converges to its expectation:

$$\hat{\mathbb{E}}_S[f] := \frac{1}{m} \sum_{i=1}^{m} f(z_i) \to \mathbb{E}[f] := \mathbb{E}_{z \sim D}[f(z)].$$

(This will allow us to reason about the generalization error by letting $f = \mathbb{1}\{h(x_i) \neq y_i\}$.) Thus, we will show the following theorem. This result was proved by Bartlett, Boucheron, Lugosi, and Mendelson 2000-2002.

Theorem 11. *Let $S = (z_1, ..., z_m)$ be random variables drawn i.i.d. from distribution $\mathcal{D}$. Let $\mathcal{F}$ be a family of functions $f : Z \to [0, 1]$. With probability at least $1 - \delta$, for all $f \in \mathcal{F}$:*

$$\mathbb{E}[f] \leq \hat{\mathbb{E}}_S[f] + 2\hat{\mathcal{R}}_S(\mathcal{F}) + \sqrt{\frac{\ln 2/\delta}{2m}},$$
$$\mathbb{E}[f] \leq \hat{\mathbb{E}}_S[f] + 2\mathcal{R}_m(\mathcal{F}) + 3\sqrt{\frac{\ln 2/\delta}{2m}}.$$

Proof. We note that it suffices to bound the function $\Phi(S) := \sup_{f \in F} \left(\mathbb{E}[f] - \hat{\mathbb{E}}_S[f] \right)$. First, we note that:

$$\Phi(S) \leq \mathbb{E}_S[\Phi(S))] + \sqrt{\frac{\ln 1/\delta}{2m}}.$$

This follows by an application of McDiarmid's inequality on the variables z_i, because the bounded difference is at most $1/m$.

We now turn towards estimating $\mathbb{E}_S[\Phi(S))]$. Next, we introduce the *ghost sample* $S' = (z'_1, ..., z'_m)$, drawn iid from $\mathcal{D}$. Then

$$\begin{aligned}
\mathbb{E}_S[\Phi(S))] &= \mathbb{E}_S\Big[\sup_{f \in \mathcal{F}} (\mathbb{E}[f] - \hat{\mathbb{E}}_S[f]) \Big] \\
&= \mathbb{E}_S\Big[\sup_{f \in \mathcal{F}} (\mathbb{E}_{S'} \hat{\mathbb{E}}_{S'}[f] - \mathbb{E}_{S'} \hat{\mathbb{E}}_S[f]) \Big] \\
&= \mathbb{E}_S\Big[\sup_{f \in \mathcal{F}} (\mathbb{E}_{S'} \big[\hat{\mathbb{E}}_{S'}[f] - \hat{\mathbb{E}}_S[f] \big]) \Big] \\
&\leq \mathbb{E}_S\Big[\mathbb{E}_{S'} \big[\sup_{f \in \mathcal{F}} (\hat{\mathbb{E}}_{S'}[f] - \hat{\mathbb{E}}_S[f]) \big] \Big] \\
&\leq \mathbb{E}_{S,S'} \big[\sup_{f \in \mathcal{F}} (\hat{\mathbb{E}}_{S'}[f] - \hat{\mathbb{E}}_S[f]) \big].
\end{aligned}$$

Therefore, we upper-bounded the expression with a symmetric difference of empirical averages $\hat{\mathbb{E}}_{S'}$ and $\hat{\mathbb{E}}_S$.

As with the proof on generalized Occam's Razor, we use the permutation trick: for each z_i, z'_i, flip a coin and swap them with probability $1/2$. Call the new sample sets T, T', which have the same distribution as S, S'. Note that:

$$\hat{\mathbb{E}}_{S'}[f] - \hat{\mathbb{E}}_S[f] = \frac{1}{m} \sum_{i=1}^{m} f(z'_i) - f(z_i),$$
$$\hat{\mathbb{E}}_{T'}[f] - \hat{\mathbb{E}}_T[f] = \frac{1}{m} \sum_{i=1}^{m} \sigma_i (f(z'_i) - f(z_i)).$$

To finish the proof, we compute:

$$\begin{aligned}
\mathbb{E}_{S,S'}\big[\sup_{f\in\mathcal{F}}(\hat{\mathbb{E}}_{S'}[f]-\hat{\mathbb{E}}_S[f])\big] &\le \mathbb{E}_{S,S',\sigma}\Big[\sup_{f\in\mathcal{F}}\Big(\frac{1}{m}\sum_{i=1}^m \sigma_i(f(z_i')-f(z_i))\Big)\Big] \\
&\le \mathbb{E}_{S,S',\sigma}\big[\sup_{f\in\mathcal{F}}\frac{1}{m}\sum_{i=1}^m \sigma_i f(z_i')+\sup_{f\in\mathcal{F}}\frac{1}{m}\sum_{i=1}^m -\sigma_i f(z_i')\big] \\
&= \mathbb{E}_{S,S',\sigma}\big[\sup_{f\in\mathcal{F}}\frac{1}{m}\sum_{i=1}^m \sigma_i f(z_i')\big]+\mathbb{E}_{S,S',\sigma}\big[\sup_{f\in\mathcal{F}}\frac{1}{m}\sum_{i=1}^m \sigma_i f(z_i)\big] \\
&= 2\mathcal{R}_m(\mathcal{F}).
\end{aligned}$$

To go from the expected Rademacher complexity to the empirical Rademacher complexity, we just apply McDiarmid's inequality again, giving us an extra factor of $2\sqrt{\frac{\ln 2/\delta}{2m}}$, and we are done. □

6.2 Generalization Error Bounds

We apply Thm. 11 to show uniform convergence results on the generalization error.

Corollary 12. *Let $S=\{(x_i,y_i)\}_{i\in[m]}$ be a sample set drawn i.i.d. from distribution $\mathcal{D}$. Let $\mathcal{H}$ be a hypothesis class. With probability at least $1-\delta$, for all $h\in\mathcal{H}$:*

$$\begin{aligned}
\mathrm{err}(h) &\le \hat{\mathrm{err}}(h)+\hat{\mathcal{R}}_S(\mathcal{H})+\sqrt{\frac{\ln 2/\delta}{2m}}, \\
\mathrm{err}(h) &\le \hat{\mathrm{err}}(h)+\mathcal{R}_m(\mathcal{H})+3\sqrt{\frac{\ln 2/\delta}{2m}}.
\end{aligned}$$

Proof. Let $Z=X\times\{-1,+1\}$. For each $h\in\mathcal{H}$, let $f_h(x,y)=\mathbb{1}\{h(x)\neq y\}$. Then we see that $\hat{\mathsf{err}}_S(h)=\hat{\mathbb{E}}_S[f_h]$ and $\mathsf{err}_{\mathcal{D}}(h)=\mathbb{E}[f_h]$. Lastly, we need to simply $\hat{\mathcal{R}}_S(\mathcal{F}_{\mathcal{H}})$. We write:

$$\begin{aligned}
\hat{\mathcal{R}}_S(\mathcal{F}_{\mathcal{H}}) &= \mathbb{E}_\sigma\Big[\sup_{f_h\in\mathcal{F}_{\mathcal{H}}}\frac{1}{m}\sum_{i=1}^m \sigma_i f_h(z_i)\Big] \\
&= \mathbb{E}_\sigma\Big[\sup_{f_h\in\mathcal{F}_{\mathcal{H}}}\frac{1}{m}\sum_{i=1}^m \sigma_i \mathbb{1}\{h(x_i)\neq y_i\}\Big] \\
&= \mathbb{E}_\sigma\Big[\sup_{h\in\mathcal{H}}\frac{1}{m}\sum_{i=1}^m \sigma_i\frac{1-y_ih(x_i)}{2}\Big] \\
&= \frac{1}{2}\mathbb{E}_\sigma\Big[\frac{1}{m}\sum_{i=1}^m\sigma_i+\sup_{h\in\mathcal{H}}\frac{1}{m}\sum_{i=1}^m\sigma_i h(x_i)\Big] \qquad \text{(because } y_i\in\{-1,+1\}\text{)} \\
&= \frac{1}{2}\hat{\mathcal{R}}_S(\mathcal{H}).
\end{aligned}$$

Thus, we are done. □

6.3 Application to Binary Classification

In this section, we apply Rademacher bounds to the problem of binary classification. We will develop ways of estimating Rademacher complexity.

For finite hypothesis classes, we have Massart's lemma:

Theorem 13 (Massart's Lemma)**.** *Let $S = (x_1, ..., x_m)$. If $|H| < \infty$, then*

$$\hat{\mathcal{R}}_S(\mathcal{H}) \leq \sqrt{\frac{2 \ln |\mathcal{H}|}{m}}.$$

The proof is omitted.

Note that the application of Massart's Lemma, as well as Cor. 12, gives us the same result as Eq. (3) with possibly different constants, which before we showed via Hoeffding's inequality and union-bound. So we have not done anything interesting yet.

Now, we will apply Rademacher bounds to *infinite* hypothesis classes. The key step is to leverage the fact that over a sample S, we can take $\mathcal{H}' \subset \mathcal{H}$, where $\mathcal{H}'$ is a smaller, finite hypothesis class which consists of one representative hypothesis class per labeling of sample S. Note that:

$$\hat{\mathcal{R}}_S(\mathcal{H}) = \mathbb{E}_\sigma\Big[\sup_{h \in H} \frac{1}{m} \sum_{i=1}^{m} \sigma_i h(x_i)\Big] = \mathbb{E}_\sigma\Big[\max_{h \in H'} \frac{1}{m} \sum_{i=1}^{m} \sigma_i h(x_i)\Big] = \hat{R}_S(H').$$

Now, we note that $|\mathcal{H}'| = |\Pi_{\mathcal{H}}(S)| \leq \Pi_{\mathcal{H}}(m)$, so Massart's Lemma and Sauer's Lemma gives:

$$\hat{\mathcal{R}}_S(\mathcal{H}) \leq \sqrt{\frac{2 \ln \Pi_{\mathcal{H}}(m)}{m}} \leq \sqrt{\frac{2d \ln(em/d)}{m}}.$$

As a result, we are able to get generalization bound:

$$\text{err}_{\mathcal{D}}(h) \leq \hat{\text{err}}_{\mathcal{D}}(h) + O\left(\sqrt{\frac{d \ln(m/d) + \ln 1/\delta}{m}}\right).$$

7 Boosting

We have hitherto developed sophisticated tools which allow us to reason about learning problems, giving us generalization bounds that show when learning works or it doesn't. However, we still need to develop theory on *algorithms* which allow us to do PAC learning. This section is devoted to understanding a specific algorithm called boosting, which answers the following question:

> Can we "boost" the performance of a "bad" algorithm (which gives us accuracy slightly better than random guessing) to learn arbitrarily well?

Surprisingly, the answer is **yes**.

7.1 Strong vs. Weak Learnability

Let's mathematically define what we mean.

Definition 9. A concept class $\mathcal{C}$ is **strongly PAC-learnable** if there exists algorithm A such that $\forall c \in \mathcal{C}$, $\forall \mathcal{D}$, for any $\epsilon, \delta > 0$, algorithm A is given $m = \text{poly}(1/\delta, 1/\epsilon)$ samples and generate hypothesis h_A such:

$$\mathbb{P}[\text{err}_{\mathcal{D}}(h_A) \leq \epsilon] \geq 1 - \delta.$$

Definition 10. A concept class $\mathcal{C}$ is **weakly PAC-learnable** if there exists algorithm A and constant $\gamma > 0$ such that $\forall c \in \mathcal{C}$, $\forall \mathcal{D}$, for any $\delta > 0$, algorithm A is given $m = \text{poly}(1/\delta)$ samples and generate hypothesis h_A such:

$$\mathbb{P}[\text{err}_{\mathcal{D}}(h_A) \leq 0.5 - \gamma] \geq 1 - \delta.$$

Strong PAC-learnability is what we have considered up until this point; we have assumed that algorithm A is able to generate hypotheses arbitrarily well (given enough data). However, perhaps weak PAC-learnability is a more realistic scenario. In the real world, we may not always be able to create algorithms which do arbitrarily well, and instead must settle for algorithms which perform marginally better than guessing ($\text{err}_{\mathcal{D}}(h_A) \leq 0.5 - \gamma$).

Clearly, strong PAC-learnability implies weak PAC-learnability. The converse is also true:

Theorem 14 (Boosting)**.** *A concept class $\mathcal{C}$ is weakly PAC-learnable iff it is strongly PAC-learnable.*

7.2 AdaBoost

We will prove the boosting theorem by analyzing a meta-algorithm called adaptive boosting (AdaBoost), which takes an ensemble of weak PAC-learners to generate a strong PAC-learner. AdaBoost was formulated by Yoav Freund and Robert Schapire in 1998.

Proof. The key insight that we will leverage is that an algorithm A generates a weak hypothesis h_A for any distribution $\mathcal{D}$, so we can create different distributions $\mathcal{D}_t$ and multiple h_t to combine them into a stronger hypothesis.

We will describe the steps of AdaBoost. A theoretical analysis will follow.

1. Take as input sample $S = \{(x_i, y_i)\}_{i \in [m]}$. Initialize the distribution $\mathcal{D}_1(i) = 1/m, \forall i$. For $t = 1, ..., T$, repeat the following steps:
2. Run A on $\mathcal{D}_t$ to get weak hypothesis h_t.
3. Compute the generalization error on S:

$$\epsilon_t := \text{err}_{D_t}(h_t) = \sum_{i: h_t(x_i) \neq y_i} D_t(i)$$

   We also set $\epsilon_t = 0.5 - \gamma_t$, and call γ_t the "edge", because it measures how far the error is away from random guessing.

4. Compute $\alpha_t := \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t}) > 0$.
5. Update the distribution over S:

$$\mathcal{D}_{t+1}(i) := D_t(i) \times \frac{\exp(-\alpha_t h_t(x_i) y_i)}{Z_t},$$

   where Z_t is the normalization factor $Z_t := \sum_{i=1}^{m} D_t(i) \times \exp(-\alpha_t h_t(x_i) y_i)$.
6. After the T steps are complete, compute the combined hypothesis

$$H(x) := \mathsf{sign}(\sum_{t=1}^{T} \alpha_t h_t(x)).$$

Note that $H(x)$ takes the form of a weighted majority vote over the weak hypotheses. We want to show that $H(x)$ is a strong learner.

7.2.1 Training error bound

First, we show a bound on the training error:

Lemma 15. *For AdaBoost, the training error satisfies the bound:*

$$\hat{\mathsf{err}}_S(H) \leq \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)} = \prod_{t=1}^{T} 2\sqrt{1-4\gamma_t^2} \leq \exp(-2\sum_{t=1}^{T}\gamma_t^2).$$

We can set the RHS to equal $\frac{1}{m}$, and this tells us that after $T = \mathcal{O}\left(\frac{\log m}{\gamma^2}\right)$ rounds, the hypothesis $H(x)$ will make no mistakes on the training data.

Proof. We will show the first inequality. The second inequality is due to the definition of γ_t, while the third follows by using $1 + x \leq e^x$.

We compute the final distribution $\mathcal{D}_{T+1}$ on S:

$$\mathcal{D}_{T+1}(i) = \frac{\exp(-y_i \sum_{t=1}^{T} \alpha_t h_t(x_i))}{m \prod_{t=1}^{T} Z_t} = \frac{\exp(-y_i F(x_i))}{m \prod_{t=1}^{T} Z_t}, i \in [m].$$

We denoted $F(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$ to simplify notation.

Then we note that:

$$\begin{aligned}
\hat{\mathsf{err}}_S(H) &= \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}\{H(x_i) \neq y_i\} \\
&\leq \frac{1}{m}\sum_{i=1}^{m} \exp(-y_i F(x_i)) && \text{(casework on } y_i F(x_i) > 0 \text{ or } < 0.\text{)} \\
&= \frac{1}{m} \times \sum_{i=1}^{m} \left(D_{T+1}(i) \times m \times \prod_{t=1}^{T} Z_t \right) && \text{(using previous computation)} \\
&= \left(\prod_{t=1}^{T} Z_t\right) \times \left(\sum_{i=1}^{m} D_{T+1}(i)\right) = \prod_{t=1}^{T} Z_t. && (\mathcal{D} \text{ is a prob. dist.})
\end{aligned}$$

All we need to do is compute Z_t:

$$\begin{aligned}
Z_t &= \sum_{i=1}^{m} D_t(i) \times \exp(-\alpha_t h_t(x_i) y_i) \\
&= \sum_{i:y_i \neq h_t(x_i)} D_t(i)\exp(\alpha_t) + \sum_{i:y_i = h_t(x_i)} D_t(i)\exp(-\alpha_t) \\
&= \epsilon_t \exp(\alpha_t) + (1-\epsilon_t)\exp(-\alpha_t).
\end{aligned}$$

Taking derivatives to minimize the RHS, we get $\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$, then $Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}$, and we are done. □

7.2.2 Generalization error bound

Now that the claim on the training error has been shown, we now turn towards showing the generalization error bounds. First, we measure the complexity of $H(x)$:

$$H(x) = \mathsf{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) = g(h_1(x), ..., h_T(x)),$$

where $g(z_1, ..., z_T) := \mathsf{sign}(\sum \alpha_t z_t)$.

- Note that g is a linear threshold function in T dimensions. We denote $\mathcal{G}$ to be the class of all such functions. We know that $\mathsf{VC}(\mathcal{G}) = T$.
- Each of the h_t is a weak hypothesis. We denote $\mathcal{H}$ to be the class which these hypotheses are in. By assumption, $\mathsf{VC}(\mathcal{H}) = d$.
- Lastly, we denote $\mathcal{F}$ to be the class of of boosted hypothese $H(x)$.

We can estimate the growth function of $\mathcal{F}$ in terms of $\mathcal{G}$ and $\mathcal{H}$ (proof is not shown):

$$\Pi_{\mathcal{F}}(m) \leq \Pi_{\mathcal{G}}(m) \cdot \Pi_{\mathcal{H}}(m)^T \leq \left(\frac{em}{T}\right)^T \cdot \left(\frac{em}{d}\right)^{dT}.$$

Lastly, we use the Rademacher bound to get:

$$\mathsf{err}_{\mathcal{D}}(H) \leq \hat{\mathsf{err}}_S(H) + O\left(\sqrt{\frac{\ln(\Pi_{\mathcal{F}}(m)) + \ln 1/\delta}{m}}\right) \leq \hat{\mathsf{err}}_S(H) + \tilde{O}\left(\sqrt{\frac{Td + \ln 1/\delta}{m}}\right). \quad (4)$$

Thus, we have shown a generalization bound that allows us to strongly learn, given access to a weak learner. □

Some remarks are in order. The bound Eq. (4) makes sense: Td is a suitable measure of the complexity of the boosted hypothesis, because we are combining T different weak hypotheses, each with a complexity of d as measured by their VC dimension.

This generalization bound predicts that if T is not too large, the generalization error decreases. However, once T becomes too big, the numerator grows faster than training error decreases, which causes an increase in generalization error. We again expect the train/test graph.

insert graph?

7.2.3 Boosting as a zero-sum game

Let's make a table, where the examples $x_1, \ldots, x_m$ are the columns and the hypotheses $h_1, h_2, \cdots \in \mathcal{H}$ are the rows. We put an $\times$ where h_i makes a mistake on example x_j. Consider a two-player game on this table. The row player (algorithm player) selects h_i and column player (example player) plays x_j. The row wins if $h_i(x_j)$ is correct, and column wins if $h_i(x_j)$ is wrong.

The boosting assumption corresponds to: for any distribution $\mathcal{D}$, there exsts a row that wins with probability at least $1/2 + \gamma$. By the minimax theorem, there must exist a distribution P over rows such that for every column, theres at least $1/2 + \gamma$ probability of winning for any x_j. Therefore, whp a large random sample from P will give the correct majority vote on all x_j. If we just listed the entire table and solved for the minimax optimal, we could get strong PAC learning. So in principle, boosting should be possible. The viewpoint is that boosting is an algorithm that has some weak access to this matrix, but this is good enough. (This is a separation oracle.)

7.3 Margins Theory

Previously we saw an algorithm called AdaBoost, which allowed us to combine predictions from T weak learners to produce a strong learner H. Our derivation of the generalization bound revealed that as we increase T, our test error will decrease initially as our hypotheses H become richer, but then will rise when we "overfit" when we keep increasing T.

However, this is typically not seen in practice - we do not observe the increase in test error when we increase T massively, even when training error reaches 0. In this section, we will

develop theory to explain this. We will improve upon the generalization bound developed in Eq. (4).

We need to dive deeper into what our predictions are doing. The key insight is that

> Even when training error plateaus, the predictions made by AdaBoost are getting *more and more confident*.

We will define what we mean by "confidence". Since we are using weighted majority vote, we will measure it by the difference between the h_t's voting correctly and those voting incorrectly.

Definition 11. Consider boosted hypothesis $H(x) = \mathsf{sign}(\sum \alpha_t h_t(x))$, where the weights α_t have been normalized such $\sum \alpha_t = 1$. The **margin** for training example (x, y) is defined as:

$$\begin{aligned} M_f(x,y) &:= \sum_{t:h_t(x)=y} \alpha_t - \sum_{h_t(x)\neq y} \alpha_t \\ &= \sum_t \alpha_t y h_t(x) \\ &= y f(x), \end{aligned}$$

where $f(x) := \sum_t \alpha_t h_t(x)$.

As with the proof on boosting, our task is twofold:

1. Show that the margins of examples tend to get larger as we keep running the boosting algorithm.
2. Large margins on training examples imply better generalization error.

7.3.1 Training error bound

We will show that the margins tend to be large.

Theorem 16. *Set $\theta \geq 0$. Then:*

$$\begin{aligned} \hat{\mathbb{P}}_S[yf(x) \leq \theta] &:= \frac{1}{m} \sum_{i=1}^m \mathbb{1}\{y_i f(x_i) \leq \theta\} \\ &\leq \prod_{t=1}^T 2\sqrt{\epsilon_t^{1-\theta}(1-\epsilon_t)^{1+\theta}}. \end{aligned}$$

If we also have for some γ that $\epsilon_t \leq 0.5 - \gamma \forall t \in [T]$, then:

$$\hat{\mathbb{P}}_S[yf(x) \leq \theta] \leq \left(\sqrt{(1-2\gamma)^{1-\theta}(1+2\gamma)^{1+\theta}}\right)^T.$$

Note that when we set $\theta = 0$, we have recovered the original result that:

$$\hat{\mathsf{err}}(H) \leq \prod_{t=1}^T 2\sqrt{\epsilon_t(1-\epsilon_t)}.$$

The proof is omitted because the steps are nearly identical to the proof for AdaBoost; we just need to recalculate with θ in the bounds.

7.3.2 Generalization error bound

Recall that our previous generalization bound (Eq. 4) was of the form:

$$\mathbb{P}_{\mathcal{D}}[yf(x) \leq 0] \leq \hat{\mathbb{P}}_S[yf(x) \leq 0] + \tilde{O}\left(\sqrt{\frac{Td + \ln 1/\delta}{m}}\right).$$

We want to get rid of the dependence on T and replace it with some dependence on θ.

Theorem 17. *For $\theta > 0$ and $\forall f \in co(\mathcal{H})$, with probability at least $1 - \delta$,*

$$\mathbb{P}_{\mathcal{D}}[yf(x) \leq 0] \leq \hat{\mathbb{P}}_S[yf(x) \leq \theta] + \tilde{O}\left(\sqrt{\frac{d/\theta^2 + \ln 1/\delta}{m}}\right).$$

Note that we are bounding the generalization error $\text{err}(H) = \mathbb{P}_D[yf(x) \leq 0]$ in terms of the sample margin error $\hat{\mathbb{P}}_S[yf(x) \leq \theta]$, and we have replaced dependence on T with $1/\theta^2$.

We require three lemmata first.

Lemma 18. *Suppose that $S = (x_1, ..., x_m)$. Then the empirical Rademacher complexity can be bounded as:*

$$\mathcal{R}_S(\mathcal{H}) \leq \sqrt{\frac{2d\ln(em/d)}{m}} = \tilde{O}\left(\sqrt{\frac{d}{m}}\right).$$

Proof. Shown in Sec. 6.3. □

Lemma 19. $\mathcal{R}_S(\mathcal{H}) = R_S(co(H))$.

Proof. Clearly, since $\mathcal{H} \subseteq \text{co}(\mathcal{H})$, we must have $\mathcal{R}_S(\mathcal{H}) \leq \mathcal{R}_S(\text{co}(\mathcal{H}))$. We will show the other way.

$$\begin{aligned}
\mathcal{R}_S(\text{co}(\mathcal{H})) &= \mathbb{E}_\sigma[\sup_{f \in \text{co}(\mathcal{H})} \frac{1}{m}\sum_i \sigma_i f(x_i)] \\
&= \mathbb{E}_\sigma[\sup_{\{h_t\} \in \mathcal{H}} \frac{1}{m}\sum_i \sum_t \sigma_i \alpha_t h_t(x_i)] \\
&\leq \mathbb{E}_\sigma[\sup_{h \in \mathcal{H}} \frac{1}{m}\sum_i \sigma_i h(x_i)] = \mathcal{R}_S(\mathcal{H}). \qquad (\text{since } \sum \alpha_t = 1.)
\end{aligned}$$

□

Lemma 20 (Contraction Principle)**.** *Suppose ϕ is a L-Lipschitz function, that is, $\forall u, v \in \mathbb{R}$, $|\phi(u) - \phi(v)| \leq L|u - v|$. Then:*

$$\mathcal{R}_S(\phi \circ \mathcal{F}) \leq L\mathcal{R}_S(\mathcal{F}).$$

Proof. Omitted. □

see Ramon notes or Mohri?

Now we are ready to prove Thm. 17.

Proof. Let $\mathcal{M}_f(x, y) := yf(x)$, and let $\mathcal{M}$ be the space of all such functions:

$$\mathcal{M} := \{M_f : f \in \text{co}(H)\}.$$

Then we compute:

$$\begin{aligned}
\mathcal{R}_S(\mathcal{M}) &= \mathbb{E}_\sigma[\sup_{f\in\text{co}(\mathcal{H})} \frac{1}{m}\sum_i \sigma_i y_i f(x_i)] \\
&= \mathbb{E}_\sigma[\sup_{f\in\text{co}(\mathcal{H})} \frac{1}{m}\sum_i \sigma_i f(x_i)] && (y_i \text{ are } \pm 1.) \\
&= \mathcal{R}_S(\text{co}(\mathcal{H})) = \mathcal{R}_S(\mathcal{H}) && \text{(by Lem. 19)}
\end{aligned}$$

Now we define a function ϕ which will allow us to relate between the empirical margin and the generalization error:

$$\phi(u) = \begin{cases} 1, & \text{if } u \le 0 \\ 1 - u/\theta, & \text{if } 0 < u \le \theta \\ 0, & \text{if } u > \theta \end{cases}$$

It is not too difficult to show that ϕ is $1/\theta$-Lipschitz. In addition, we note that:

$$\begin{aligned}
\mathbb{P}_\mathcal{D}[yf(x) \le 0] &= \mathbb{E}_\mathcal{D}[\mathbb{1}\{\{\}yf(x) \le 0\}] \le \mathbb{E}_\mathcal{D}[\phi(yf(x))], \\
\hat{\mathbb{P}}_S[yf(x) \le \theta] &= \hat{\mathbb{E}}_S[\mathbb{1}\{\{\}yf(x) \le \theta\}] \ge \hat{\mathbb{E}}_S[\phi(yf(x))].
\end{aligned}$$

Now, we compute a Rademacher generalization bound on the function class $\phi \circ \mathcal{M}$, using Thm. 11:

$$\begin{aligned}
\mathbb{P}_\mathcal{D}[yf(x) \le 0] &\le \mathbb{E}_\mathcal{D}[\phi(yf(x))] \\
&\le \hat{\mathbb{E}}_S[\phi(yf(x))] + 2\mathcal{R}_S(\phi \circ \mathcal{M}) + O\left(\sqrt{\frac{\ln 1/\delta}{m}}\right) \\
&\le \hat{\mathbb{E}}_S[\phi(yf(x))] + \tilde{O}\left(\sqrt{\frac{d}{m}}\right) + O\left(\sqrt{\frac{\ln 1/\delta}{m}}\right) \\
&\le \hat{\mathbb{P}}_S[yf(x) \le \theta] + \tilde{O}\left(\sqrt{\frac{d/\theta^2 + \ln 1/\delta}{m}}\right),
\end{aligned}$$

as desired. □

8 Support Vector Machines

We continue our discussion of algorithms. Now, we consider a classifier algorithm, which we explicitly train to maximize the margin of examples in $\mathbb{R}^n$.

There are a few reasons why we might want to do this:

- Small perturbations γ in an example shouldn't change how the example is classified. A dividing hyperplane consistent with a sample such that this γ is maximized is equivalent to finding a max-margin solution. The points of distance exactly γ from the dividing hyperplane are called the **support vectors**.
- While the VC dimension of linear threshold functions in n dimensions is n, if our points fall in the unit ball, the VC dimension of linear threshold functions with margin γ is $\leq (1/\gamma)^2$. This expression shrinks as γ, which in turn corresponds to a reduction in complexity.

8.1 Optimization Problem Formulation

In our problem setup, we will have a sample $S = \{(x_i, y_i)\}_{i=1}^m$, where $x_i \in \mathbb{R}^n$ and $y_i \in \{-1, +1\}$. Suppose there actually exists a margin separator of the data. This is called the realizable setting. Our max-margin problem can be written as:

Problem: Find v to maximize δ subject to $\|v\| = 1$ and for all i:

$$\begin{cases} v \cdot x_i \geq \delta & \text{if } y_i = +1 \\ v \cdot x_i \leq \delta & \text{if } y_i = -1. \end{cases}$$

An equivalent way to write this is:

Problem: Minimize $\frac{1}{2}\|w\|^2$ subject to $y_i(w \cdot x_i) \geq 1$ for all i.

This is a convex optimization problem with linear constraints, so it can easily be solved. The Lagrangian for this problem is therefore:

$$L(w, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^m \alpha_i (y_i(w \cdot x_i) - 1).$$

We can write out the KKT conditions:

$$\begin{aligned} \nabla_w L(w^\star, \alpha^\star) &= 0, \\ y_i(w^\star \cdot x_i) - 1 &\geq 0, \\ \alpha_i &\geq 0, \\ \alpha_i(y_i(w^\star \cdot x_i) - 1) &= 0. \end{aligned}$$

Computing the derivative we see:

$$\frac{\partial L(w^\star, \alpha^\star)}{\partial w_j} = w_j - \sum_{i=1}^{m} \alpha_i y_i x_{ij} = 0,$$

So therefore w is a weighted linear combination of the samples: $w = \sum_{i=1}^{m} \alpha_i y_i x_i$.

The dual of the problem is:

$$\max_{\alpha_i} \sum_{\alpha_i} -\frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$
$$\text{subject to: } \alpha_i \geq 0.$$

Note that our classifier $h(x)$ takes the form of

$$\begin{aligned} h(x) &= \text{sign}(w \cdot x) \\ &= \text{sign}(\sum_i \alpha_i y_i (x_i \cdot x)). \end{aligned}$$

By complementary slackness, $\alpha(y_i(w \cdot x_i) - 1) = 0$, so therefore if $\alpha_i \neq 0$, then we must have $y_i(w \cdot x_i) = 1$, which means that x_i is a support vector. So actually, $h(x)$ is only a weighted classifier of those examples x_i which are support vectors. Letting k be the number of support vectors, one can show that our generalization error is $\tilde{O}\left(\frac{k + \ln 1/\delta}{m}\right)$. This provides an alternative, dimension-free way to characterize the generalization error (compare to the bound which uses VC dimension of $\tilde{O}\left(\frac{1/\gamma^2 + \ln 1/\delta}{m}\right)$).

8.2 Non-separable data

What happens if there is no separating hyperplane? How can we use SVMs in this case? In this case, we introduce the soft margin problem.

If our data is "almost" linearly separable, we can try to move the offending points a bit: for fixed i, ξ_i represents the distance than an example x_i must be moved to have margin at least 1. Then we write our optimization problem as:

$$\min_w \frac{1}{2} \|w\|^2 + C \sum_{i \in S} \xi_i$$
$$\text{subject to } y_i(w \cdot x_i) \geq 1 - \xi_i,$$
$$\xi_i \geq 0.$$

C is a hyperparameter that we can tune. Note that ξ_i is exactly the hinge loss: $\max\{0, 1 - y_i(w \cdot x_i)\}$. Intuitively, the hinge loss is an upper bound on the $0-1$ loss for the training example. The

first term $\|w\|^2$ is roughly an upper bound on the amount of overfitting. Together, the objective is proportional to a rough upper-bound on the true error.

Now, lets compute the dual form of the SVM:

$$\begin{aligned}
\max_a \min_w L(w,a) &= \max_{a_1\geq 0, a_2\geq 0} \min_{w,\xi} \frac{1}{2}\|w\|^2 + C\sum_i \xi_i + \sum_i \alpha_{i1}(1-\xi-i-y_i(w\cdot x_i)) - \sum_i \alpha_{i2}\xi_i \\
&= \max_{a\in[0,C]} \min_w \frac{1}{2}\|w\|^2 + \sum_i \alpha_i(1-y_i(w\cdot x_i)) \\
&= \max_{a\in[0,C]} \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j (x_i\cdot x_j) + \sum_i \alpha_i - \sum_i \alpha_i y_i \sum_j a_j y_j (x_j\cdot x_i) \\
&= \max_{a\in[0,C]} \sum_i \alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j (x_i\cdot x_j)
\end{aligned}$$

where we first took derivative w.r.t. ξ_i to get $\alpha_{i1}+\alpha_{i2}=C$, then we take derivative wrt w to get $w=\sum_i \alpha_i y_i x_i$.

Notice that this is kernelizable, we can replace $x_i\cdot x_j$ with $K(x_i,x_j)$ everywhere (see next section).

8.3 Kernel trick

The idea is if we project our data into a higher dimensional space, it will become linearly separable and we can apply the SVM algorithm.

picture of dots separated by circle.

For example, we can define the mapping $F:\mathbb{R}^2\to\mathbb{R}^6$:

$$(x_1,x_2)\to F(x)=(1,x_1,x_2,x_1x_2,x_1^2,x_2^2).$$

Then an SVM would be trying to find a vector $v=(a,b,c,d,e,f)$ which defines the hyperplane $v\cdot F(x)=a+bx_1+cx_2+dx_1x_2+ex_1^2+fx_2^2$.

In $\mathbb{R}^6$, this is a hyperplane. However, in the original space $\mathbb{R}^2$, this is a conic section.

Based on this discussion, we can generally start with an n-dimensional space and use mappings to map to a space of dimension $O(n^k)$. This is perfectly valid from a mathematical perspective; however, computationally we have some questions with using SVMs in high dimension problems:

- Do we need more data with bigger dimensions? Our theoretical analysis of SVMs indicates that the answer is no, as our generalization bounds are dimension free.
- How do we deal with exponentially increasing computational and storage costs? Note that the only operation that we need in the SVM algorithm is the inner product. This leads us to the *kernel trick*.

Going back to our example, we would like to compute inner products of the form $F(x) \cdot F(z)$ for $x, z \in \mathbb{R}^2$. First, we modify F by some constants:

$$(x_1, x_2) \to F(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2).$$

It is not too hard to see that:

$$F(x) \cdot F(z) = (1 + x \cdot z)^2.$$

In other words, we can compute inner products via computing inner products in the original space $\mathbb{R}^2$! In our SVM algorithm, we never will have to explicitly map to the high-dimensional space.

This trick generalizes:

Definition 12 (Informal.)**.** A **kernel** is a real-valued function $K : (x, z) \to K(x, z)$ that satisfies

1. K is symmetric, i.e. $K(x, z) = K(z, x)$.
2. K is positive definite, i.e. $K(x, z) \geq 0$ for all x, z.
3. There exists ϕ such that $K(x, z) = \phi(x) \cdot \phi(z)$.

Example 14. The **polynomial kernel** is defined as $K(x, z) = (1 + x \cdot z)^k$

Example 15. The **Gaussian kernel** is defined as $K(x, z) = \exp(-c \|x - z\|^2)$.

8.4 SVMs vs. Boosting

We will investigate the connections between SVMs and Boosting.

In the SVM algorithm, we are given inputs x such that $\|x\|_2 = 1$ (WLOG). We want to find an unknown vector v such that $\|v\|_2 = 1$. Our prediction for new examples is then $\text{sign}(v \cdot x)$, and our notion of margin is $y(v \cdot x)$.

We can view boosting in terms of learning weights of weak hypotheses. Lets say our hypothesis space $\mathcal{H}$ is finite with cardinality N, then we can define $h(x) \in \{-1, +1\}^N$:

$$h(x) = (h_1(x), ..., h_N(x)).$$

Therefore, we have that the "inputs" $h(x)$ satisfy $\|h(x)\|_\infty = 1$. Next, we note that boosting requires us to find weights $a = (a_1, ..., a_N)$ such that $\|a\| = 1$. Our prediction for new examples is $\text{sign}(a \cdot h(x))$, while our margin is $y(a \cdot h(x))$.

Concluding, we see that boosting and SVMs are very similar. The main difference arises from the norm constraints on the input and prediction vector.

9 Statistical Query Model

We've discussed how to learn in the agnostic setting, and analyzed how many samples we need. Usually its $\mathcal{O}(1/\varepsilon^2)$ instead of $\mathcal{O}(1/\varepsilon)$. However, in regards to poly-time algorithms, this is harder. For example, finding the approximate best conjunction is NP-hard. In general, even the ERM step is not computationally tractable. One way to make progress is to make additional assumptions on the noise, thus simplifying the problem.

In this section, we will provide an introduction to the statistical query (SQ) model, which restricts how the learning algorithm can interact with the data. It was introduced by Kearns in order to handle learning in the presense of *random classification noise*.

In the SQ model, we cannot interact with the data directly via labeled examples. Instead, we have access to a noisy black-box oracle that asks for statistical properties of the data $\mathcal{D}$. We can indeed show that learning in RCN can be efficiently done if such a black-box oracle can be efficiently simulated.

9.1 Random Classification Noise

First, we will describe the random classification noise setting.

Definition 13 ((PAC+RCN)-learnable)**.** An algorithm A PAC-learns $\mathcal{C}$ from RCN if for any $f \in \mathcal{C}$, any distribution $\mathcal{D}$, any $\eta < 1/2$, any $\varepsilon, \delta > 0$ if: given access to $\mathsf{EX}^{\eta}(f, \mathcal{D})$, A finds a hypothesis h that is ε-close to f ($\mathsf{Pr}_{\mathcal{D}}[h(x) \neq f(x)] < \varepsilon$) with probability at least $1 - \delta$. We require sample/time complexity that is $\mathrm{poly}(1/\varepsilon, 1/\delta, 1/1-2\eta, n, \mathsf{size}(f))$.

The example oracle $\mathsf{EX}^{\eta}(f, \mathcal{D})$ is defined as follows:

- Example (x, y) is drawn from $\mathcal{D}$ on $\mathcal{X} \times \{0, 1\}$.
- With probability $1 - \eta$, set $\ell(x) = f(x)$ and with probability η set $\ell(x) = 1 - f(x)$. Return $(x, \ell(x))$. That is, we flip the true label with probability $\eta(x)$.

So the RCN model is somewhere inbetween agnostic and realizable, because we are heavily constraining the behavior of the noise.

Notice that we are asking the learner to get ε-close to the true f, which can closer than the data that the learner receives. However, this is fine because minimizing the noisy error rate will allow us to minimize the true error rate. That is, if a given hypothesis h has non-noisy error p, its noisy error rate is $p(1-\eta) + (1-p)\eta = \eta + p(1-2\eta)$. So minimizing the noisy error rate to some $\eta + \varepsilon$ will give us true error rate of $\varepsilon/(1-2\eta) = \mathcal{O}(\varepsilon)$.

Another notation we will use in the sequel is denoting Pr to be probability of an event with respect to the non-noisy distribution and Pr_{η} for probability with respect to the noisy distribution.

9.2 Learning Monotone OR-functions

In this section we will show that monotone OR-functions can be learned from RCN. Recall that in the PAC model, we learned monotone OR-functions by taking a large sample S and deleting variables set to 0 in positive examples of S. In this way, we always have a one sided error, in that the true target $f \subseteq h$.

However, this will not work in the RCN setting. Instead, suppose that we know the noise error rate η. For each index $i \in [n]$, we denote $p_i := \mathsf{Pr}[f(x) = 0 \text{ and } x_i = 1]$. In words, p_i is an upper-bound on the additional error on positive examples we would incur by including x_i in our hypothesis. Note that if x_i is in the target f, then $p_i = 0$. We observe that any h which includes all x_i such that $p_i = 0$ and no x_i such that $p_i > \varepsilon/n$ is "good". (The overall probability of error is $\sum p_i < \varepsilon$ then.)

So we actually just need to estimate the quantities p_i up to an error of $\pm\varepsilon/2n$, and include all $x_i : \hat{p}_i < \varepsilon/2n$ in our hypothesis h. Now define $q_i = \mathsf{Pr}[f(x) = 0 | x_i = 1]$. We have $p_i = q_i \times \mathsf{Pr}[x_i = 1]$. The quantity $\mathsf{Pr}[x_i = 1]$ is unaffected by noise, so it can be estimated by samples of EX^η. Note that:

$$\Pr_\eta[\ell(x) | x_i = 1] = q_i(1-\eta) + (1-q_i)\eta = \eta + q_i(1-2\eta).$$

Therefore, to derive high confidence bounds on p_i, it suffices to derive high confidence bounds on the quantities $\mathsf{Pr}_\eta[\ell(x)|x_i = 1]$ and $\mathsf{Pr}[x_i = 1]$, which we can do by a large enough sample of $\mathsf{EX}^\eta(f, \mathcal{D})$ and applying Chernoff bounds.

Lastly, note that if the noise rate η is not known, we can estimate it with the smallest value of $\mathsf{Pr}_\eta[\ell(x)|x_i = 1]$ (which is an upper bound on the true η).

9.3 Statistical Query Model

Using the previous example of learning OR-functions, we can generalize the algorithm to the notion of "statistical query". The basic idea was to see how we can learn in a non-noisy model by asking probabilities about certain events with some "slop". Then we try to learn in the noisy model by breaking events into parts which are predictably affected by noise, and parts which are unaffected by noise.

Really, what we wanted to know was a noisy answer to some statistical question. In the monotone OR functions example, the question was: "what is the probability that $x_i = 1$ and the (true) label is negative?". Formally:

Definition 14. We say an algorithm A learns a class $\mathcal{C}$ by a class $\mathcal{H}$ in the SQ model if for any $f \in C$, any distribution $\mathcal{D}$, any $\eta < 1/2$, we have:

- Define the query function $\chi : \mathcal{X} \times \{0,1\} \to [0,1]$, with a tolerance $\tau \in (0,1]$ such that $\tau \geq 1/\mathrm{poly}(\dots)$. The function χ must be poly-time computable.
- For such a function χ, there exists a value $P_\chi := \mathbb{E}_{x\sim\mathcal{D}}[\chi(x, f(x))]$.
- The world returns a value $P'_\chi \in [P_\chi - \tau, P_\chi + \tau]$.

- The algorithm can do this for poly number times, as well as ask for unlabeled data. At the end, it must output $h \in \mathcal{H}$ with error $\leq \varepsilon$.

This definition is a bit hard to unpack. A few examples of query functions follow.

1. In the previous example of learning monotone OR-functions we had $\chi(x, f) := \mathbb{1}\{x_i = 1, f(x) = 0\}$. We wanted to learn $\mathsf{Pr}(f(x) = 0 \text{ and } x_i = 1)$ up to $\tau = \varepsilon/2n$ accuracy. We will return an OR of all x_i with $P'_\chi \leq \varepsilon/2n$.
2. We can ask for the error rate of the current hypothesis by defining $\chi(x, f) = \mathbb{1}\{h(x) \neq f(x)\}$.
3. In the perceptron algorithm, the function in question is the vector-valued $\chi(x, f) = Z^{-1} f(x)x$ if $h(x) = f(x)$, and 0 otherwise, where $Z := \mathsf{Pr}[h(x) \neq f(x)]$. The value we receive is $\approx \mathbb{E}[f(x)x | h(x) \neq f(x)]$.
4. Hill-climbing or gradient descent algorithms ask the question: What is the error rate of h? What would it be if I made this tweak?

Next, we show the following theorems, which relates the SQ model to the PAC model, even with RCN!

Theorem 21. *SQ-learnable $\Rightarrow$ PAC-learnable. More precisely, if A makes M queries of tolerance τ to learn $\mathcal{C}$ to error ε, a sample size of $\mathcal{O}\left(\frac{M}{\tau^2}\log\frac{M}{\delta}\right)$ suffices to learn $\mathcal{C}$ to error ε with probability at least $1-\delta$ in the PAC model.*

Proof. Suppose we requested a sample of size $\mathcal{O}\left(\frac{M}{\tau^2}\log\frac{M}{\delta}\right)$. We will break it into M pieces $S_1, \ldots, S_M$, each of size $\mathcal{O}\left(\frac{1}{\tau^2}\log\frac{M}{\delta}\right)$. To answer query i, we will use the dataset S_i. Since we are estimating the value of a $[0,1]$-bounded random variable, a sample size of $\mathcal{O}\left(\frac{1}{\tau^2}\log\frac{M}{\delta}\right)$ is sufficient to produce the estimate within $\pm\tau$ with probabilty $\geq 1-\delta/M$. By union bound we have the result. □

In fact, we can also learn in the RCN model:

Theorem 22. *SQ-learnable $\Rightarrow$ (PAC+RCN)-learnable. More precisely, if A makes M queries of tolerance τ to learn $\mathcal{C}$ to error ε, a sample size of $\mathcal{O}\left(\frac{M}{\tau^2(1-2\eta)^2}\log\frac{M}{\delta}\right)$ suffices to learn $\mathcal{C}$ to error ε with probability at least $1-\delta$ in the PAC model.*

Therefore, any SQ algorithm can automatically be simulated in the presence of RCN.

Proof sketch. [1] Given a query χ, we want to show that we can estimate it with noisy data up to some error τ. At a high level, we will break the query into a part predictably affected by noise and a part unaffected. We will estimate these parts separately, either by drawing fresh examples for each query or estimate many queries from the same sample if the VC dimension of the query space is small (using uniform convergence bounds).

[1] A different proof of this fact can be found in these notes. At a high level, they are doing the same thing.

We define:

$$\mathsf{CLEAN} = \{x \in \mathcal{X} : \chi(x, 0) = \chi(x, 1)\}$$
$$\mathsf{NOISY} = \{x \in \mathcal{X} : \chi(x, 0) \neq \chi(x, 1)\}$$

That is, in the clean subset, the label does not affect the value of the query. In the noisy subset, the label does affect the value of the query. Given an example x, the learner knows whether $x \in \mathsf{CLEAN}$ or $x \in \mathsf{NOISY}$. For example, when learning monotone conjunctions with $\chi(x, f) = \mathbb{1}\{x_i = 1 \wedge f(x) = 0\}$, the set CLEAN is precisely those x with $x_i = 0$.

Then:

$$\mathsf{Pr}(\chi(x, f(x)) = 1) = \mathsf{Pr}(\chi(x, f(x)) = 1 \wedge x \in \mathsf{CLEAN}) + \mathsf{Pr}(\chi(x, f(x)) = 1 \wedge x \in \mathsf{NOISY})$$

We will estimate the two things on the RHS separately using standard Hoeffding bounds.

Step 1. The first term is easy to estimate from noisy data because we will just run the query on the labeled data and use sample averages and concentration of measure. Note that even though we have noisy data $(x_i, \ell(x_i))$, we can still estimate $\chi(x, f(x))$ because when $x \in \mathsf{CLEAN}$, the actual label doesn't matter.

Step 2. We can estimate $\mathsf{Pr}(x \in \mathsf{NOISY})$, as well as the quantity $P_\eta \stackrel{\text{def}}{=} \mathsf{Pr}_\eta(\chi(x, f(x)) = 1 | x \in \mathsf{NOISY})$. However, we actually wanted to estimate $P \stackrel{\text{def}}{=} \mathsf{Pr}(\chi(x, f(x)) = 1 | x \in \mathsf{NOISY})$.

No worries. Note that $P_\eta = \eta + P(1 - 2\eta)$, so we can actually estimate $P = \frac{P_\eta - \eta}{1 - 2\eta}$, too. We just need to estimate P_η up to additive error $\tau(1 - 2\eta)$.

Step 3. To conclude, we have shown how to estimate the query function $\mathsf{Pr}(\chi(x, f(x)) = 1)$ up to error τ. The proof can be completed by Hoeffding's inequality and union bound over M queries. Lastly, if we don't know η, we can use a "guess and check" wrapper. □

9.4 Statistical Query Dimension

In the SQ model, we can precisely characterize what we can and cannot learn in it, via a quantity called the SQ dimension.

Definition 15. We say f, g are *uncorrelated* if $\mathsf{Pr}_{x \sim \mathcal{D}}[f(x) = g(x)] = \frac{1}{2}$.

Definition 16. We define the *statistical query dimension* of a class $\mathcal{C}$ with respect to $\mathcal{D}$ is the size of the largest set $C' \subseteq \mathcal{C}$ such that for all $f, g \in C'$:

$$\left| \mathop{\mathsf{Pr}}_{\mathcal{D}}[f(x) = g(x)] - \frac{1}{2} \right| < \frac{1}{|C'|}.$$

We denote it as $\mathsf{SQDIM}_{\mathcal{D}}(\mathcal{C})$.

In other words, $\mathsf{SQDIM}_{\mathcal{D}}(\mathcal{C})$ is the size of the largest set of nearly uncorrelated functions in $\mathcal{C}$.

We have the following theorems which characterize whether a class $\mathcal{C}$ is learnable by SQ algorithms.

Theorem 23. *If $\mathsf{SQDIM}_{\mathcal{D}}(\mathcal{C}) = \mathrm{poly}(n)$ then you can weak-learn $\mathcal{C}$ over $\mathcal{D}$ by SQ algorithms.*

Proof. Let's denote $d = \mathsf{SQDIM}_{\mathcal{D}}(\mathcal{C})$. Consider a maximal subset $H \subseteq C$ with

$$\left| \Pr_{\mathcal{D}}[h_i(x) = h_j(x)] - \frac{1}{2} \right| < \frac{1}{d+1} \text{ for all } i, j.$$

We can see that $|H| \leq d$.

To learn, we can just try each $h_i \in H$ and use an SQ to estimate its error. At least one of h_i or $-h_i$ must be a weak predictor. If not, we could have also added the true $f^\star \in \mathcal{C}$ to the set H because $\Pr_{\mathcal{D}}[h_i(x) = f^\star(x)] = 1/2$, contradicting the fact that H is maximal. Since there are $\mathrm{poly}(n)$ hypotheses in H, we just require $\mathrm{poly}(n)$ queries with $\tau \approx 1/2d$ to find a weak learner. □

Theorem 24. *If $\mathsf{SQDIM}_{\mathcal{D}}(\mathcal{C}) > \mathrm{poly}(n)$ then you cannot weak-learn $\mathcal{C}$ over $\mathcal{D}$ by SQ algorithms.*

Actually, we will show the weaker statement:

Theorem 25. *If C has $n^{\omega(1)}$ uncorrelated functions, and the target is a random one of them, then with high probability any SQ algorithm that makes $\mathrm{poly}(n)$ queries of tolerance $1/\mathrm{poly}(n)$ fails to weak learn.*

The key tool in this proof will be the Fourier analysis of boolean functions. It will be helpful to think of functions as vectors of 2^n entries:

$$f : \{0,1\}^n \to \{-1,+1\} = \left(\sqrt{D[000]} f(000), \sqrt{D[001]} f(001), ... \right)$$

In other words, we represent f as its truth table, weighted by the probability of x under distribution $\mathcal{D}$. Note that $\langle f, f \rangle = 1$ and $\langle f, g \rangle = \sum_x \Pr(x) f(x) g(x) = \mathbb{E}_{\mathcal{D}}[f(x)g(x)]$, the correlation of f and g (which is exactly the previously defined view of correlation up to an affine transformation). Thus, we have defined a valid vector space of dimension 2^n.

Let's fix an orthonormal basis, call it $\phi_1, \ldots, \phi_{2^n}$. We can write any vector f in this basis as: $f = \sum_i \hat{f}_i \phi_i$, where $\hat{f}_i = \langle f, \phi_i \rangle$ is the projection of f onto basis vector ϕ_i. These $\hat{f}_i$ are called the *Fourier coefficients of f* in the ϕ basis.

By Parseval's identity, we know that $\sum_i \hat{f}_i^2 = \sum_i f_i^2 = \sum_i D[i] = 1$. Therefore, at most t^2 of the coefficients $\hat{f}_i = |\langle f, \phi_i \rangle| \geq 1/t$. In other words, any given boolean function f can be $\geq 1/t$-correlated with at most t^2 functions in the orthogonal set. Any given f can be weakly correlated with at most a poly number of functions in the orthogonal set.

Now we will prove the "simpler" theorem.

Proof. Let's pick $\phi_1, \ldots, \phi_m$ orthogonal (uncorrelated) functions in C, and extend it to the basis $\phi_1, \ldots \phi_{2^n}$. While we know the first m are boolean functions in C, we note that the vectors in the extension may not necessarily be.

Consider a query $\chi : \{0,1\}^n \times \{-1,+1\} \to [-1,+1]$, which is actually a vector in 2^{n+1} dimensions. We'll extend our basis to this higher-dimensional space, i.e., create a new basis $\tilde{\phi}_i$, $i \in [2^{n+1}]$. Also we extend our distribution $\mathcal{D}' \stackrel{\text{def}}{=} D \times \text{Unif}\{-1,+1\}$

1. For the first $i \in [2^n]$, we set the entries $g_i(x,y) \stackrel{\text{def}}{=} \phi_i(x)$, ignoring the label y. It is easy to check that these g_i are still orthogonal:

$$\Pr_{\mathcal{D}'}[g_i(x,y) = g_j(x,y)] = \Pr_{\mathcal{D}}[\phi_i(x) = \phi_j(x)] = \frac{1}{2},$$

   using the orthogonality of the original ϕ_i.

2. We need 2^n more basis functions. We will define h_i for $i \in [2^n]$ as having the entries $h_i(x,y) \stackrel{\text{def}}{=} y\phi_i(x)$. It is easy to check that h_i, h_j are orthogonal for $i \neq j$, as well as h_i and g_j even if $i = j$.

Armed with this new basis, we write $\chi = \sum_i \alpha_i g_i + \sum_i \beta_i h_i$, and by Parseval's identity $\sum_i \alpha_i^2 + \sum_i \beta_i^2 = 1$. Thus, the expected value of the query is:

$$\begin{aligned}
\mathbb{E}_{\mathcal{D}}[\chi(x,c(x))] &= \mathbb{E}_{\mathcal{D}}\left[\sum_i \alpha_i g_i(x,c(x)) + \sum_i \beta_i h_i(x,c(x))\right] \\
&= \sum_i \alpha_i \mathbb{E}_{\mathcal{D}}[\phi_i(x)] + \sum_i \beta_i \mathbb{E}_{\mathcal{D}}[c(x)\phi_i(x)] \\
&= \sum_i \alpha_i \mathbb{E}_{\mathcal{D}}[\phi_i(x)] + \beta_{i^\star}.
\end{aligned}$$

In the last line, we used the fact that the target c was drawn randomly from $\phi_1, \ldots, \phi_m$, and we defined $c = \phi_{i^\star}$. Note with high probablility, $\beta_{i^\star}$ cannot be too big, because at most $\text{poly}(n)$ of them have correlation $\geq 1/\text{poly}(n)$.

Therefore, with high probability the adversary can always return the value $\sum_i \alpha_i \mathbb{E}_{\mathcal{D}}[\phi_i(x)]$, which does not depend on the target c, and the learner cannot hope to learn whether their hypothesis h is correlated with the target c. □

Some comments:

1. The very last step of the proof had the adversary convert $\sum_i \alpha_i \mathbb{E}_{\mathcal{D}}[\phi_i(x)] + \beta_i$ to $\sum_i \alpha_i \mathbb{E}_{\mathcal{D}}[\phi_i(x)]$. We can also make this work in the "honest SQ" model, where it's estimated from a random sample, not given adversarially.

2. We can also use SQ-dim to prove that certain (C, D) pairs have no large-margin kernels (kernels where every $c \in \mathcal{C}$ looks like a large-margin separator in the implicit space).

Example 16. Consider learning parity functions $c(x) = c \cdot x \bmod 2$. Let $\mathcal{D}$ be the uniform distribution on $\{0,1\}^n$. We see that any two parity functions are uncorrelated, and there are 2^n of them, so $\mathsf{SQDIM}_{\mathcal{D}}(\mathcal{C}) = 2^n$. Applying the theorem, we see that parity functions are not efficiently learnable in the SQ model. In addition, because any parity function of size $\log n$ can be described as a size n decision tree, poly-sized decision trees are not SQ-learnable either. This is because there are approximately $\binom{n}{\log n}$ parity functions of size $\log(n)$, which is superpolynomial, so there are at least $n^{\omega(1)}$ uncorrelated poly-sized decision trees.

However, we can learn parity functions with *non-SQ* algorithms. This is a linear system, so we can solve the system of equations with Gaussian elimination.

This raises the question: can we learn parities with random classification noise? In fact, this is an open problem, see "learning with noise".

10 Computation Hardness Results for Learning

So far, we have discussed efficient algorithms for various problems, such as:

- Given a dataset S, find a consistent DL if one exists.
- Given a dataset S, find a consistent LTF if one exists.
- Given a dataset S, find a consistent disjunction if one exists.

The goal of this section is to prove hardness results for some problems, i.e., show that it is impossible to find algorithms which solve these problems in poly-time *without* making distributional assumptions.

10.1 Some hard problems

We will discuss learning intersection of halfspaces, 2-clause CNFs, and minimizing ERM for LTFs. In all cases, these problems are NP hard so we should not expect to be able to do this efficiently.

10.1.1 Intersection of 2 halfspaces

We want to see if given a collection of data, find 2 LTFs such that their AND is consistent with the data. We will show this is NP-hard by giving a reduction from the NP-hard problem of the hypergraph 2-color problem.

Problem 1 (Hypergraph 2-Color Problem)**.** Given m subsets $S_1, \ldots S_m$ of n nodes, color each node red or blue such that each S_i has at least one red node and at least one blue node.

We will show that given an instance of the hypergraph 2-color problem, we can create a set S of positive and negative examples such that S is consistent with an intersection of 2

halfspaces iff the given instance was 2-colorable. Therefore, if we can solve the intersection of 2 halfspaces problem in poly-time, then we could solve the hypdergraph 2-color problem in poly-time, a contradiction!

1. Consider points in $\mathbb{R}^n$. Label the origin $+$, each coordinate vector $\hat{x_j}$ to negative.
2. For each set S_i, label indicator vector for that set as positive. For example, if a set contains points 1 and 2, we would label $e_1 + e_2$ to be positive.

Proposition 26. The data is consistent with an intersection of 2 halfspaces iff the given instance was 2-colorable.

Proof. Given a 2-coloring, we can define the halfspaces:

$$\sum w_i x_i \leq 0.5, \text{ where } w_j = \begin{cases} 1 & j \text{ is red} \\ -n & j \text{ is blue.} \end{cases}$$

$$\sum w'_i x_i \leq 0.5, \text{ where } w'_j = \begin{cases} 1 & j \text{ is blue} \\ -n & j \text{ is red.} \end{cases}$$

Why does this work? First note that the intersection of these two halfspaces labels the origin to $+$, as expected. Next, consider any set S_i, with the indicator being positive. This S_i must contain at least one red vertex and one blue vertex, so the sum of weights for both halfspaces will have a term which is $-n$, thus making it negative. Now, for every coordinate vector, one halfspace will label it positive and the other negative, so it does not lie in the intersection.

What about the other direction? If someone gives us two halfspaces, lets call one "red" and one "blue". If $\hat{x}_j$ is separated from the origin by the red halfspace, color j red, if separated by the blue halfspace, color j blue; otherwise pick arbitrarily.

There is no 1-color set because the convex hulls of $\hat{x}_j \in S_i$ and $\{0, \vec{S_i}\}$ overlap. This implies that at least one point has to lie to one side of the convex hull, and one has to lie to the other, thus they get different colorings. □

10.1.2 2-clause CNFs

It turns out theres a similar reduction for learning 2-clause CNFs, which are defined below.

Definition 17. A 2-clause CNF is defined as an AND of 2 OR functions. Example:

$$(x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6).$$

So we cannot properly learn 2-clause CNFs.

However, you can learn 2-clause CNFs using a 2-DNF representation (a DNF formula where each term has at most 2 literals). This makes sense because any 2-clause CNF can be multiplied out to get a 2-DNF, e.g.,:

$$x1x_4 \wedge x_1x_5 \wedge x_1x_6 \ldots$$

We can learn using a list-and-cross-off algorithm, but note that we have blown up the size to $\mathsf{VC} = \Theta(n^2)$.

10.1.3 Solving ERM for LTFs

We know how to find a consistent LTF when one exists. We can also minimize total hinge loss. But what about minimizing the training error? Turns out this is NP-hard. The reduction is quite similar to the previous one, using the NP-hard *maximum independent set problem* in graphs.

Problem 2 (Maximum Independent Set)**.** Given a graph G, find a set of vertices such that no two vertices have an edge between them.

A similar reduction works:

1. Set the origin to be positive.
2. Set each coordinate vector $\hat{x}_i$ to $-$.
3. For each edge (i, j) put a $+$ value at $\hat{x}_i + \hat{x}_j$.

Proposition 27. The maximum independent set corresponds to the largest set of negatives that can be linearly separated from the positives.

This shows that the problem "find the LTF that correctly classifies all positives and makes fewest mistakes on negatives" is NP-hard. This is not quite what we wanted to show. To finish off the argument, we will replicate each positive example $n+1$ times, so the min error will only makes mistakes on negatives. Therefore, if an algorithm could actually find the minimum error LTF on the training data, then it could solve the maximum independent set problem, a contradiction because maximum independent set is NP-hard.

10.2 Representation-Independent Hardness

We have shown how to show hardness for consistency learning *using* a particular hypothesis class $\mathcal{H}$. This also gives us hardness for PAC learning, because we can set a uniform distribution on the output of the reduction and set $\varepsilon < 1/n$. Now, we will show representation-independent hardness, where we talk about hardness based on the complexity of the target, allowing the learner to use any representation it wants.

Examples:

1. Parity functions require $2^{\Omega(n)}$ SQs of tolerance $1/\mathrm{poly}(n)$ to learn in the SQ model.
2. Decision trees, DNFs require $n^{\mathcal{O}(\log n)}$ SQs of tolerance $1/\mathrm{poly}(n)$ to learn in the SQ model.

Furthermore, the hardness was for even doing better than random guessing. However, the issue with these SQ models was that while they didn't restrict representation used by the learning algorithm, they did restrict the way the algorithm can interact with the data. What if we don't want to restrict either one?

We will use cryptographic assumptions instead.

Definition 18. Define a function $f : \{0,1\}^n \to \{0,1\}^m$ where $m > n$ to be a *pseudorandom generator* if for any poly-time algorithm A, any constant c:

$$\left| \Pr_{v \sim \{0,1\}^m} [A(v) = 1] - \Pr_{x \sim \{0,1\}^m} [A(f(x)) = 1] \right| = o\left(n^{-c}\right).$$

In other words, no poly-time algorithm A can distinguish pseudorandom strings of length m (the result of running f on random input of length n) and truly random strings of length m.

It is crucial that our algorithm run in poly-time. If we allowed infinite time, we could just run the algorithm on all 2^n inputs and check the outputs.

There have been a series of results that show that if you can construct a generator f such that breaking f would give a poly-time algorithm for factoring. So, f is a PRG if factoring is hard.

Any algorithm that can even weak-learn arbitrary $\mathcal{O}(\log n)$-depth AND/OR networks over uniform random examples in polynomial time would give a poly-time algorithm for factoring. The high level idea is:

1. Think of PRG with significant stretch: $n \to \text{poly}(n)$.
2. The network has PRG input I built in, computes $f(I)$, and outputs the j-th bit, wehre j is given by low-order $\lg(\text{poly}(n))$ bits of example x.
3. If the algorithm can learn, then it can distinguish PRG output from the true random.

Note: We can even extend this to pseudo-random functions.

A more recent result:

Theorem 28 (Daniely 2016)**.** *Assume that "refuting random k-XORs is NP-hard". (Given $m < n^{\sqrt{k}\log k}$ examples over $\{0,1\}^n$ where k bits are set to 1, it is hard to distinguish the case that: the examples and labels are random and independent vs. there is a parity function with error $\leq 10\%$.)*

Then for any constant c there is no poly-time algorithm that, given any S on n^c points in $\{-1,+1\}^n$ can distinguish whp the case that:

1. *The labels are just uniform random coin flips*
2. *There exists a linear separator of error at most 10%. (can replace this with any constant).*

So, no poly-time algorithm can even weak-learn under the second condition! This is quite sad.

11 Halving, WMA, and RWMA

Up until now, the learning problems we have been considering have all been batch learning: algorithms received a batch of samples S and compute a single hypothesis h_A from

these, which is used to predict unknown examples. We measure the performance of h_A via generalization error.

Now, we'll turn our focus to an entirely different topic: online learning. In the online learning setting, an algorithm receives examples one at a time, and must generate a hypothesis each time. Training and testing happen at the same time. One important point about online learning is that we can analyze it without making any statistical assumptions about the data. As a natural consequence, our metrics for performance are *mistake* and *regret* bounds.

An example of online learning is when we wake up everyday, and want to predict weather At the beginning of the day, we predict whether it will rain or not. By the end of the day, we have observed the true outcome.

11.1 Mistake-bound Model

First, we introduce the mistake-bound (MB) model, which is an online learning model.

Definition 19 (Mistake-bound model)**.** An algorithm A **learns class** $\mathcal{C}$ with mistake bound M if A makes $\leq M$ mistakes on any sequence of examples consistent with some $f \in \mathcal{C}$.

Let n be the size of each example, and s be the complexity of the target.

$\mathcal{C}$ is **learnable in the MB model** if there exists an algorithm with mistake bound and running time per example poly(n, s).

Example 17 (Disjunctions)**.** Disjunctions of n bits are learnable in the MB model with at most n mistakes.

Definition 20. An algorithm is **conservative** if it only changes its state when it makes a mistake.

Claim. If $\mathcal{C}$ is learnable with mistake-bound M, then it is learnable by a conservative algorithm.

Proof. Let A be an algorithm which learns $\mathcal{C}$. We modify it to produce a conservative algorithm A' by running A, but does not update when it classifies an example A classifies correctly. Suppose the data $x_1, ..., x_t$ was received and A' made updates on $i1, ..., ik$. A' performs as if A had seen the sequence $x_{i1}, ..., x_{ik}$, but there are at most M of these. □

Next, we show that MB learnable implies PAC learnable.

Theorem 29. *Suppose a concept class $\mathcal{C}$ is learnable in the MB model. Then it is also learnable in the PAC model.*

Proof. First note that we can assume that A is conservative WLOG using the claim. We run the algorithm A until it produces a hypothesis h that gets correct $\frac{1}{\varepsilon} \log \frac{M}{\delta}$ examples in a row, and return h.

The algorithm for PAC learning fails if it returns a bad hypothesis, that is, one with err$(H) > \varepsilon$. In m samples, this happens with probabilty at most $(1-\varepsilon)^m < \frac{\delta}{M}$. The algorithm produces at most M hypotheses because it is conservative, so we union bound over all hypothese h to get an error rate of $< \delta$. □

11.2 Halving Algorithm

What if we did not care about computational time? We can try to enumerate all hypotheses. At each round, remove the wrong hypthoses and continue. Consider the following problem, called learning with experts.

Problem: Let there be N experts. At each time $t = 1, ..., T$, each expert offers a prediction $h_i(x_t) \in \{0, 1\}$. A learner must predict $\hat{y} \in \{0, 1\}$ using the expert advice. Afterwards, they observe the actual outcome $y \in \{0, 1\}$. We declare a mistake if $\hat{y} \neq y$.

Let's analyze this in the mistake-bound model.

Theorem 30. *Suppose that there exists at least one expert that makes no mistakes at all. Then the* ***halving algorithm*** *makes at most $m \leq \lg N$ mistakes.*

Halving algorithm: whenever an expert makes a mistake, the learner will never listen from them again. The learner always predicts based on majority vote of the remaining experts who have never made a mistake.

Proof. Define W as the number of experts which haven't made mistakes yet. We have $W = N$ initially, with W decreasing over time. We also have assumed $W \geq 1$. Whenever the learner makes a mistake, at least one half of the remaining experts has also made a mistake (because we use majority vote). Therefore, after m mistakes, we have $1 \leq W \leq (\frac{1}{2})^m N$. Solving for m yields the desired conclusion. □

Thus, we have shown that the halving algorithm has a mistake bound of $\log N$. We can extend this in numerous ways. For example, if we had a prior p on $\mathcal{C}$, we could weight the vote according to p. Then we will make at most $\log(1/p_f)$, where f is the target function. Moreover, if f was actually chosen to p, the expected number of mistakes we make would be upper bounded by $\sum p_h \log(1/p_h) = H(p)$.

11.3 Online PAC-Learning

We will draw a connection to PAC Learning, which will allow us to reason about whether we can do better than the halving algorithm.

Note that the expert learning problem can be rewritten as follows. Let there be a finite hypothesis space $\mathcal{H} = \{h_1, ..., h_N\}$. There's a target concept $c \in \mathcal{H}$. At each round, an adversary gives us $x \in \mathcal{X}$. Each expert computes $\xi_i = h_i(x)$. Our algorithm predicts $\hat{y}$ and we observe $y = c(x)$.

Let's define $M_A(\mathcal{H}) := \max_{c,x}(\#\text{ mistakes made by A})$. Then the best deterministic algorithm is defined opt$(\mathcal{H}) := \min_A M_A(H)$. We already know:

$$\text{opt}(\mathcal{H}) \leq M_{\text{Halving}}(\mathcal{H}) \leq \lg |H|.$$

Now, we will show the following:

Theorem 31. *$VC(\mathcal{H}) \leq opt(\mathcal{H})$.*

Proof. Let A be some arbitrary deterministic algorithm, and let $d = VC(H)$. There exists some $x_1, ..., x_d$ which can be shattered by $\mathcal{H}$. Let's choose $c \in \mathcal{H}$ such that for $t = 1, ..., d$, our adversary presents x_t, A predicts $\hat{y}_t$, and we have $y_t = c(x_t) \neq \hat{y}_t$. We can always choose c because of the shattering property; moreover, the adversary can simulate A exactly because it is deterministic. □

11.4 Weighted Majority Algorithm

Let's go back to the learning with experts problem. What if we don't have a perfect expert? In this case, the halving algorithm will prune all the experts and we won't have any to listen to eventually. Therefore, instead we will keep the experts around, but adjust their influence on our decision making when they make mistakes. Also, instead of measure the mistakes, we measure the difference between the number of mistakes we make vs. that of the best expert.

This is the Weighted Majority Algorithm (WMA).

1. Set some parameter $\beta \in [0, 1)$.
2. We initialize the weights $w_i = 1$ for each expert.
3. For each round $t = 1, ..., T$:
   - Get prediction ξ_i from each expert. Define $q_0 := \sum_{i:\xi_i=0} w_i$ and $q_1 := \sum_{i:\xi_i=1} w_i$.
   - Predict $\hat{y} = 1$ if $q_1 > q_0$ and 0 otherwise.
   - Observe y; for all experts where $\xi_i \neq y$ update $w_i \leftarrow w_i\beta$.

We can bound the number of mistakes WMA makes.

Theorem 32.

$$(\#\textit{ mistakes of WMA}) \leq a_\beta(\#\textit{ mistakes of best expert}) + c_\beta \lg N.$$

where

$$a_\beta := \frac{\lg(1/\beta)}{\lg(2/(1+\beta))}, c_\beta := \frac{1}{\lg(2/(1+\beta))}.$$

By investigating different values of β, we see that there is a tradeoff between a_β and c_β. For example, if we set $\beta = 0.5$, then we see that $\alpha_\beta = c_\beta \approx 2.4$.

In addition, by dividing both sides by T, we see that $c_\beta \lg N/T \to 0$, so asymptotically, the rate at which WMA makes mistakes is bounded by a constant times the rate that the best expert makes mistakes.

Proof. Let's define $W := \sum_{i=1}^{N} w_i$. Initially we know $W = N$. On a round, let w_i be the current weights, and w_i' be the updated weights after this round. Suppose that $y = 0$ ($y = 1$ is similar) and that the learner made a mistake. Then we have:

$$\begin{aligned}
W' &= \sum_{i=1}^{N} w_i' \\
&= \sum_{i:\xi_i=0} w_i + \sum_{i:\xi_i=1} w_i\beta \\
&= q_0 + \beta q_1 \\
&= W - (1-\beta)q_1. \\
&\leq W - (1-\beta)\frac{W}{2} && \text{(because learner made a mistake)} \\
&= \frac{1+\beta}{2}W.
\end{aligned}$$

Thus after m mistakes,

$$W \leq \left(\frac{1+\beta}{2}\right)^m.$$

Now we seek a lower bound on W. Let L_i be the number of mistakes that expert i makes. Therefore by the end $w_i = \beta^{L_i}$. Clearly $W \geq w_i$, so we have the two-sided bound:

$$\beta^{L_i} \leq W \leq \left(\frac{1+\beta}{2}\right)^m.$$

Solving and minimizing over L_i we have:

$$m \leq \frac{\lg \frac{1}{\beta} \min_i L_i + \lg N}{\lg \frac{2}{1+\beta}}$$

as desired. □

11.5 Randomized Weighted Majority Algorithm

We introduce randomness in our algorithm, which will allow us the sharpen the constants. Here, we discuss the Randomized Weighted Majority Algorithm (RWMA). The only difference with WMA is that we choose predictions as $\mathbb{P}[\hat{y} = 1] = q_1/W$ and $\mathbb{P}[\hat{y} = 0] = q_0/W$.

Intuitively, RWMA is smothing out the worst case: if our algorithm is 49/51 in favor of the correct label, before with deterministic voting we would get it wrong, but now we have 49% chance of getting it right.

We can show the following result on RWMA.

Theorem 33.

$$\mathbb{E}[(\#\textit{ mistakes of RWMA})] \leq a_\beta(\#\textit{ mistakes of best expert}) + c_\beta \lg N$$

where

$$a_\beta := \frac{\ln(1/\beta)}{1-\beta}, c_\beta := \frac{1}{1-\beta}.$$

We can see that $a_\beta \to 1$ as $\beta \to 1$, which is an improvement for large T.

Proof. Define l as the probability that RWMA makes a mistake on a round.

$$l = \mathbb{P}[\hat{y} \neq y] = \frac{\sum_{i:\xi_i \neq y} w_i}{W}.$$

As before, we see how W changes in every round:

$$\begin{aligned} W' &= \sum_{i:\xi_i \neq y} w_i\beta + \sum_{i:\xi_i = y} w_i \\ &= lW\beta + W - lw \\ &= W(1 - l(1-\beta)). \end{aligned}$$

Therefore

$$\begin{aligned} W^{\text{final}} &= N \cdot \prod_{t=1}^{T}(1 - l_t(1-\beta)) \\ &\leq N \cdot \prod_{t=1}^{T} \exp(-l_t(1-\beta)) \\ &= N \exp(-(1-\beta)\sum_{t=1}^{T} l_t). \end{aligned}$$

As before, we have:

$$\beta^{L_i} \leq W^{\text{final}} \leq N \exp(-(1-\beta)\sum_{t=1}^{T} l_t).$$

.

Rearranging and taking the minimum over L_i we see that:

$$\sum_{t=1}^{T} l_t = \mathbb{E}[(\#\text{ mistakes of RWMA})] \leq \frac{\ln(1/\beta)}{1-\beta} \min_i L_i + \frac{1}{1-\beta} \ln N.$$

□

We can make an approximation of the previous theorem. Let $\beta = 1-\varepsilon$. Then the previous theorem reads:

$$\begin{aligned}\mathbb{E}[(\#\text{ mistakes of RWMA})] &\leq -\frac{\ln(1-\varepsilon)}{\varepsilon}\min_i L_i + \frac{1}{\varepsilon}\ln N. \\ &\approx (1+\varepsilon)\min_i L_i + \frac{1}{\epsilon}\ln N.\end{aligned}$$

If we knew an upper bound on $\min_i L_i$, then we can take derivative w.r.t ε to set $\varepsilon := \sqrt{\frac{\ln N}{K}}$, thus giving us the bound:

$$\mathbb{E}[(\#\text{ mistakes of RWMA})] \leq \min_i L_i + 2\sqrt{K\ln N} \leq \min_i L_i + 2\sqrt{T\ln N}.$$

Thus, we can see that RWMA is a **no-regret algorithm**, because the average regret goes to 0 as $T \to \infty$.

11.6 Regret Lower Bound

Note that we can usually have $\min_i L_i \leq T/2$; for any expert, we can always create the counter expert which predicts the opposite, and therefore one of them has an error $\leq T/2$.

Thus we have some result of the form:

$$\frac{\mathbb{E}[(\#\text{ mistakes of RWMA})]}{T} \leq \frac{\min_i L_i}{T} + \sqrt{\frac{\ln N}{2T}} + \frac{\lg N}{2T}.$$

Therefore when $T \to \infty$, we see that the rate that the learner is making mistakes will approach the corresponding rate for the best expert, with regret $O(\sqrt{\frac{T\ln N}{2}})$.

We would like to see if we can show a matching lower-bound on regret - is it true that any algorithm has regret at least $\sqrt{\frac{T\ln N}{2}}$?

If we consider the setting where all of the experts are randomly guessing, and y is randomly chosen, we must have $\mathbb{E}[(\#\text{ mistakes of RWMA})] = T/2$. However, it can be shown that $\mathbb{E}[\min_i L_i] \approx T/2 - \sqrt{\frac{T\ln N}{2}}$.

Therefore, $\mathbb{E}[(\#\text{ mistakes of RWMA})] \gtrsim \mathbb{E}[\min_i L_i] + \sqrt{\frac{T\ln N}{2}}$. We have achieved the desired lower bound in expectation. This also shows that we do just as well in the stochastic setting as the adversarial one.

12 Perceptron Algorithm

Up until now, we have been evaluating algorithms against the best expert. However, there may not be a best expert, but instead a *committee* of experts that votes well, and we want

to give a weighting of these experts $w_t \in \mathbb{R}^N$. Another interpretation is learning a "large margin" linear separator in $\mathbb{R}^N$. We will formalize this new setup as follows.

Again, there are N experts, each of which gives us a label $x_i \in \mathbb{R}$. Thus, at each time step $t = 1, ..., T$ the learner receives a vector $x_t \in \mathbb{R}^N$. The learner predicts $\hat{y}_t \in \{-1, +1\}$ and observes the outcome $y_t \in \{-1, +1\}$.

We will focus on an algorithm of the form where the learner keeps track of a vector w_t. The learner predicts $\hat{y}_t = \text{sign}(w_t \cdot x_t)$. After each time step, the learner iteratively updates $w_{t+1} \leftarrow F(w_t, x_t, y_t)$. The hope is that $w_t \to w^\star$.

We will first look at the Perceptron algorithm. Here, $w_1 = 0$ is initialized to the all 0's vector. The function $F(w_t, x_t, y)$ is defined as:

$$w_{t+1} = \begin{cases} w_t + y_t x_t, & \text{if learner made a mistake } (\hat{y} \neq y) \\ w_t. & \text{otherwise} \end{cases}$$

12.1 Proof with Margin Assumption

Note that this algorithm is a conservative algorithm, because we only update when we make a mistake.

We will show the following result on Perceptron.

Theorem 34. *Assume that:*

1. *For all t, $\|x_t\| \leq R$.*
2. *There exists $w^\star, \delta$ such that $\|w^\star\| = 1$ and $y_t(w^\star \cdot x_t) \geq \delta > 0$. The second condition implies that every example is classified correctly with margin at least δ.*

Then the Perceptron algorithm makes at most R^2/δ^2 mistakes.

Proof. We will use something called a *potential function* Φ_t, which is used in physics to measure how much energy there is left in a system at time t.

In our case, we set the potential function to be the cosine similarity between w_t and $w^\star$, i.e.

$$\Phi_t := \frac{w_t \cdot w^\star}{\|w_t\|_2} \leq 1.$$

Similar to how we have been proving things, we will derive upper and lower bounds on w_{T+1} in terms of the relevant quantities. WLOG, we can assume that our algorithm makes a mistake on every example.

First, note that

$$\begin{aligned} w_{T+1} \cdot w^\star &= (w_t + y_t x_t) \cdot w^\star \\ &= w_t \cdot w^\star + y_t(w^\star \cdot x_t) \\ &\geq T\delta. \end{aligned}$$

The last step unrolls the recursion and uses the minimum margin assumption.

Next, note that:

$$\begin{aligned}\|w_{T+1}\|^2 &= (w_t + y_t x_t) \cdot (w_t + y_t x_t) \\ &= w_t \cdot w_t + 2y_t(w_t \cdot x_t) + y_t^2 x_t \cdot x_t \\ &\leq \|w_t\|^2 + 0 + R^2. && \text{(mistake assumption and } \|x_t\| \leq 1.) \\ &= T.\end{aligned}$$

Therefore:

$$\frac{T\delta}{\sqrt{T}} \leq \Phi_{T+1} \leq R \Rightarrow T \leq \frac{R^2}{\delta^2}.$$

□

12.2 A Lower Bound

It turns out that Perceptron is optimal for deterministic algorithms:

Claim 1. It is not possible to get $< R^2/\delta^2$ mistakes with a deterministic algorithm.

Proof. Consider $n = R^2/\delta^2$ coordinate vectors scaled in $\mathbb{R}^n$ to length R. These will be our inputs. We build a sequence of labels $y_1, ..., y_n$ by simulating our algorithm and making sure that each y_i is opposite of the algorithm's label $\hat{y}_i$. Therefore, if the sequence of labels is $y_1, ..., y_n$ then the algorithm must make a mistake for every input.

Next, we justify that there exists $w^\star$ such that $w^\star \cdot x_i = y_i$ for all $i \in [n]$. We can pick $w^\star$ of the form:

$$w^\star = (\pm x_1 \pm x_2 ... \pm x_{R^2/\delta^2})/R.$$

Note that $|w^\star \cdot x| = 1$ for all input vectors, so we can force all mistakes with the sign of each x_i. Lastly, it is clear that the margin is $\delta = 1/\|w^\star\|$. □

12.3 What if there is no perfect separator?

In this case, it may be possible for $|w \cdot w^\star|$ to drop for incorrect examples. In this case, we need to modify our analysis. We define the *hinge loss* on $w^\star$ to be: $L_{\text{hinge}}(x, y) = \max(0, 1 - y(w^\star \cdot x))$.

Theorem 35. *On any sequence of examples, the perceptron algorithm makes at most*

$$\min_{w^\star} \left\{ \|w^\star\|^2 R^2 + 2L_{hinge}(w^\star, S) \right\}.$$

where $L_{hinge}(w^\star, S)$ is the total hinge loss of $w^\star$ on the set S.

Proof. Suppose we make a mistake at time t, then:

$$w_{t+1} \cdot w^\star = w_t \cdot w^\star + y_t(x_t \cdot w^\star) \geq w_t \cdot w^\star + 1 - L_{\text{hinge}}(x_t, y_t).$$

After M mistakes, we know that $w \cdot w^\star \geq M - L_{\text{hinge}}(w^\star, S)$. We also know that $\|w\|^2 \leq MR^2$. Using Cauchy-Schwarz, we conclude that:

$$\begin{aligned}
&(M - L_{\text{hinge}})^2 \leq w \cdot w^\star \leq MR^2 \|w^\star\|^2 \\
&\Rightarrow M \leq R^2 \|w^\star\|^2 + 2L_{\text{hinge}} - L_{\text{hinge}}^2/M \\
&\Rightarrow M \leq R^2 \|w^\star\|^2 + 2L_{\text{hinge}},
\end{aligned}$$

as claimed. □

12.4 Some Additional Comments

- Recall that an algorithm's regret is at least the VC dimension of the hypothesis class. In this case, $\mathcal{H}$ is linear threshold functions with margin at least δ, so we have:

  $$VC(\mathcal{H}) \leq \#\text{mistakes} \leq 1/\delta^2,$$

  so we have come up with an indirect proof on the VC dimension of this hypothesis class!

- Let's investigate the minimum margin condition. Perceptron works well when there is a small, but good subcommittee and we want to select it. However, let's rewrite this norm expression in terms of k, the subcommittee size, and N, the number of experts. It seems that $y_t(u \cdot x_t) \geq 1/\sqrt{Nk}$, so therefore the number of mistakes is at most Nk. This bound may perform poorly when N become large. This motivates our discussion of the Winnow algorithm. discuss this better

- The perceptron can easily be extended to kernels, by replacing dot products with the kernel.

13 Winnow Algorithm

Previously, we analyzed the perceptron algorithm, which can be thought of as online learning a weighting of experts. However, perceptron may perform poorly due its dependence on the total number of experts N (see the subcommittee example in the previous section). Winnow was invented to resolve this issue.

In the Winnow algorithm, we initialize all weights $w_1 = (\frac{1}{N}, ..., \frac{1}{N})$ to the uniform distribution. Then the function $F(w_t, x_t, y)$ is defined as:

- If $\hat{y} \neq y$:

  $$w_{t+1,i} = w_{t,i} \frac{\exp(\eta y_i x_{t,i})}{Z_t} \quad \forall i.$$

  ($\eta > 0$ is a parameter; Z_t is a normalization factor.)

- Otherwise: $w_{t+1} = w_t$.

Theorem 36. *Assume that:*

1. *The learner makes a mistake at every round, i.e.* $T = (\#\text{ mistakes of Winnow})$.
2. *For all* t, $\|x_t\|_\infty \leq 1$.
3. *There exists* u, δ *such that* $\|u\|_1 = 1$, $u \geq 0$, *and* $y_t(u \cdot x_t) \geq \delta > 0$.

Then the Winnow algorithm makes at most $\frac{\ln N}{\eta\delta + \ln(2/e^\eta + e^{-\eta})}$ *mistakes. When* $\eta := \frac{1}{2}\ln\frac{1+\delta}{1-\delta}$, *Winnow makes at most* $\frac{2\ln N}{\delta^2}$ *mistakes.*

The details can be worked out, but for the subcommittee example, we will get a mistake bound of $2k^2 \log N$, improving our dependence on N but worsening our dependence on k.

Proof. Again, we will use potential functions to give useful upper and lower bounds. We want to measure some "distance" between w_t and the ground truth u. Therefore, we choose relative entropy as our potential function:

$$\Phi_t = \text{RE}(u||w_t).$$

where recall that $\text{RE}(P||Q) := \sum_i P_i \ln(P_i/Q_i)$.

Let's see how Φ_t changes in a round:

$$\begin{aligned}
\Phi_{t+1} - \Phi_t &= \text{RE}(u||w_{t+1}) - \text{RE}(u||w_t) \\
&= \sum_i u_i \ln\frac{u_i}{w_{t+1,i}} - \sum_i u_i \ln\frac{u_i}{w_{t,i}} \\
&= \sum_i u_i \ln\frac{w_{t,i}}{w_{t+1,i}}. \\
&= \sum_i u_i \ln(Z_t \exp(-\eta y_t x_{t,i})) && \text{(by the update rule)} \\
&= \ln(Z_t) - \eta y_t \sum_i u_i x_{t,i}. \\
&\leq \ln(Z_t) - \eta\delta && \text{(by margin assumption.)}
\end{aligned}$$

Now we are left with bounding $Z_t := \sum_i w_{t,i} \exp(\eta y_t x_{t,i})$.

Note that $y_t x_{t,i} \in [-1, +1]$, so we want to bound $e^{\eta z}$ in $[-1, +1]$. We see that:

$$e^{\eta z} \leq \frac{e^\eta + e^{-\eta}}{2} + \frac{e^\eta - e^{-\eta}}{2} z = \cosh(\eta) + z\sinh(\eta).$$

Therefore:

$$Z_t \leq \sum_i w_i(\cosh(\eta) + y_t x_{t,i}\sinh(\eta)) \leq \cosh(\eta).$$

This follows because the w_i for a probability distribution, and because $w_t \cdot x_t$ has a different sign as y_t, and the second term is negative.

Putting this all together, we see that

$$\Phi_{t+1} - \Phi_t \leq \ln(\frac{e^\eta + e^{-\eta}}{2}) - \eta\delta$$

After calculating the potential function at the beginning where w_1 is the uniform distribution, we see that $\Phi_1 = \ln N$. We also know that relative entropy is nonnegative, i.e. $\Phi_T \geq 0$. Putting this together, we see that:

$$T = (\#\text{mistakes}) \leq \frac{\ln N}{\eta\delta + \ln(2/e^\eta + e^{-\eta})}.$$

The choice of η and resulting bound are easily verified by bounding the relative entropy. □

As a quick side note, we previously supposed that all $u_i \geq 0$. What happens if we want to relax this condition?

Consider an example where $u = (0.5, -0.2, 0.3)$ and $x = (1, 0.7, -0.4)$. We map these to vectors $u' = (0.5, 0, 0.3, 0, 0.2, 0)$ and $x' = (1, 0.7, -0.4, -1, -0.7, 0.4)$. which clearly satisfy the original problem assumptions. This is called *balanced Winnow*.

We have doubled the vector as $N \to 2N$, but since the regret bound is logarithmic in N, we do not pay too much.

14 Linear Regression

Often, we are more concerned with predicting a value $y \in \mathbb{R}$, rather than a label as in classification. We will analyze this setting by considering regression.

The key to regression is the construction of a **loss function**, which measures the error between our prediction $h(x)$ and the ground truth y. For example, we can minimize the expected *square loss* (also called the *risk*):

$$\mathbb{E}_{(x,y)\sim\mathcal{D}}[(h(x) - y)^2].$$

By minimizing this risk, we are forcing $h(x)$ to be close to the function $p(x) = \mathbb{E}[y|x]$:

Claim. $\mathbb{E}[(h(x) - p(x))^2] = \mathbb{E}[(h(x) - y)^2] - \mathbb{E}[(p(x) - y)^2]$.

Proof. We use marginalization:

$$\mathbb{E}_{x,y} = \mathbb{E}_x \mathbb{E}_{y|x},$$

which shows that it suffices to prove the theorem for fixed x and then take expectation on both sides with respect to x. On the LHS, we have $\mathbb{E}_{y|x}[(h-p)^2] = (h-p)^2$, since h and p are constant.

On the RHS, we have:

$$\begin{aligned}
\mathbb{E}_{y|x}[(h(x)-y)^2] - \mathbb{E}_{y|x}[(p(x)-y)^2] &= \mathbb{E}_{y|x}[h^2 - 2hy + y^2] - \mathbb{E}_{y|x}[p^2 - 2py + y^2] \\
&= h^2 - 2h\mathbb{E}_{y|x}y - p^2 + 2p\mathbb{E}_{y|x}y \\
&= h^2 - 2hp + p^2 \qquad \text{(since } p = \mathbb{E}_{y|x}\text{)} \\
&= (h-p)^2.
\end{aligned}$$

Taking expectation over x on both sides, the claim holds. □

As with classification algorithms, we must use empirical risk minimization: view minimalization of the empirical risk $\hat{E}[(h(x)-y)^2] = \frac{1}{m}\sum_{i=1}^{m}(h(x_i)-y_i)^2$ as a *proxy* to the true risk $\mathbb{E}[(h(x)-y)^2]$. Therefore, we require some sort of uniform convergence result on the class h. We can generalize our previous approaches which use tools like Chernoff bounds, VC-dimension, growth functions, and so on to this problem of regression.

We do not pursue this further, but instead focus on using online learning algorithms. We will examine one such algorithm for linear regression, due to Widrow and Hoff.

The setup for the online linear regression problem is as follows. We want to learn linear functions $h(x) = w \cdot x$ for some parameter vector w.

We first initialize some w_1. Then for each step $t = 1, ..., T$, we get a sample vector x_t, make prediction $\hat{y}_t = w_t \cdot x_t$, observe y_t, and update w_t via some rule. We will evaluate our algorithm according to the loss $L_A = \sum_{t=1}^{T}(\hat{y} - y)^2$. We will measure regret against the best linear predictor $\min_u L_u := \sum_{t=1}^{T}(u \cdot x_t - y_t)^2$.

14.1 Widrow-Hoff

Here we present the Widrow-Hoff algorithm for online linear regression, along with its analysis. Essentially, Widrow-Hoff is parameterized by a "step size" $\eta > 0$ which controls how fast we change w. We initialize $w_1 = 0$. At each time step, we update using the following rule:

$$w_{t+1} = w_t - \eta(w_t \cdot x_t - y_t)x_t.$$

There are two interpretations to this update rule:

- Readers may recognize this as stochastic gradient descent on the loss function $L(w, (x, y)) = (w \cdot x - y)^2$. We hope that at each step, a good update step will decrease the loss on the most recently seen example, and we can do this by following the gradient $\nabla_w L = 2(w \cdot x - y)x$.

- We want our new weight vector w_{t+1} to satisfy two properties (i) achieve smaller loss on current example (x_t, y_t) (ii) stay close to the current weight w_t because we don't want to throw away all our progress. Therefore, we minimize the weighted sum:
$$\eta(w_{t+1} \cdot x - y_t)^2 + \|w_{t+1} - w_t\|_2^2 .$$
Solving this, we get
$$w_{t+1} = w_t - \eta(w_{t+1} \cdot x_t - y_t)x_t,$$
which is close to our update rule, except we have w_{t+1} on the RHS too. When w_{t+1} is close to w_t, we can replace it with w_t.

We show the following result on Widrow-Hoff:

Theorem 37. *If $\|x_t\|_2 \leq 1$ for all t, then the following bound holds for L_{WH}:*
$$L_{WH} \leq \min_{u \in \mathbb{R}^n} \left(\frac{L_u}{1-\eta} + \frac{\|u\|_2^2}{\eta} \right).$$

Proof. Again, we use a potential function argument. But first, some notation.

- We let the potential function be the squared loss $\Phi_t := \|w_t - u\|_2^2$.
- Let $l_t := w_t \cdot x_t - y_t = \hat{y}_t - y_t$; l_t^2 is the loss of WH at round t.
- Let $g_t := u \cdot x_t - y_t$; g_t^2 is the loss of weight vector u at round t.
- Let $\Delta_t := \eta(\hat{y}_t - y_t)x_t = \eta l_t x_t$. Note that $w_{t+1} = w_t - \Delta_t$.

First we make the following claim:

Claim. $\Phi_{t+1} - \Phi_t \leq -\eta\ell_t^2 + \frac{\eta}{1-\eta} g_t^2$.

Proof. In words, this is giving us a bound on the potential function decrease at every round. The first term corresponds to the loss of WH (as WH suffers loss, the potential goes down) and the second corresponds to the loss of the best linear predictor u (as u suffers loss, the potential goes up).

The calculation is straightforward:
$$\begin{aligned}
\Phi_{t+1} - \Phi_t &= \|w_{t+1} - u\|_2^2 - \|w_t - u\|_2^2 \\
&= \|w_t - u - \Delta_t\|_2^2 - \|w_t - u\|_2^2 \\
&= -2(w_t - u) \cdot \Delta_t + \|\Delta_t\|_2^2 \\
&= -2\eta\ell_t(w_t \cdot x_t - u \cdot x_t) + \eta^2 \ell_t^2 \|x_t\|_2^2 \\
&\leq -2\eta\ell_t(w_t \cdot x_t - y_t + y_t - u \cdot x_t) + \eta^2\ell_t^2 && \text{bounding } \|x_t\|_2^2 \text{ by 1} \\
&= \eta^2\ell_t^2 - 2\eta\ell_t^2 + 2\eta\ell_t g_t \\
&\leq \eta^2\ell_t^2 - 2\eta\ell_t^2 + \eta\left(\frac{g_t^2}{1-\eta} + \ell_t^2(1-\eta)\right) && \text{using trick } ab \leq (a^2+b^2)/2 \\
&= -\eta\ell_t^2 + \frac{\eta}{1-\eta} g_t^2 .
\end{aligned}$$

□

Now we show how the theorem follows from the above claim.

$$\begin{aligned}
-\|u\|_2^2 &= -\Phi_1 && \text{because } w_1 = 0 \\
&\leq \Phi_{T+1} - \Phi_1 && \text{because } \Phi_{T+1} \geq 0 \\
&= \sum_{t=1}^{T} (\Phi_{t+1} - \Phi_t) \\
&\leq \sum_{t=1}^{T} \left(-\eta \ell_t^2 + \frac{\eta}{1-\eta} g_t^2 \right) && \text{by claim} \\
&= -\eta \sum_{t=1}^{T} \ell_t^2 + \frac{\eta}{1-\eta} \sum_{t=1}^{T} g_t^2 \\
&= -\eta L_{WH} + \frac{\eta}{1-\eta} L_u.
\end{aligned}$$

Solving for L_{WH} gives the statement of the theorem. □

14.2 Online to Batch Conversion

So far, we have discussed two main models of learning:

- **Batch learning.** This includes the PAC model. The learner is given a set of random examples offline and asked to minimize generalization error.
- **Online learning.** The learner is given a stream of possibly adversarial examples and asked to minimize cumulative loss (measured against a best performer in hindsight).

A natural question to ask is whether we can relate these two models. Clearly, the online setting is stronger, because there we made no assumption on the *distribution* of examples coming to us. More precisely, we would like to understand if the batch learning problem can be reduced to the online learning problem; that is, given black box access to an online learner, can we learn in the batch setting?

The formal setup is as follows. We are given access to a set of examples $S = \{(x_i, y_i)\}_{i \in [m]}$ drawn i.i.d. from $\mathcal{D}$. We would like to use an online learner on our dataset S to learn a vector v with low risk: $R_v = \mathbb{E}_{(x,y)\sim\mathcal{D}}[(v \cdot x - y)^2]$.

Here, we will give a result employing the Widrow-Hoff algorithm to show that we can do this in the regression setting:

1. Run Widrow-Hoff for T=m rounds on S in the random order they were given to us.
2. Widrow-Hoff gives us a sequence of weight vectors $w_1, \ldots w_m$.
3. Return the average of these vectors: $v = \frac{1}{m} \sum_{t=1}^{m} w_t$.

This is also known as "one-pass SGD".

Theorem 38. *The expected risk R_v can be bounded as follows:*

$$\mathbb{E}_S[R_v] \leq \min_{u \in \mathbb{R}^n} \left(\frac{L_u}{1-\eta} + \frac{\|u\|_2^2}{\eta m} \right).$$

Here, the expectation is taken over draws of the random sample S.

Proof. We denote $\mathbb{E}$ to be the expectation taken with respect to the random sample S as well as a new test point (x, y) drawn from $\mathcal{D}$. Fix any $u \in \mathbb{R}^n$. Then:

1. $(v \cdot x - y)^2 \leq \frac{1}{m} \sum_{t=1}^m (w_t \cdot x - y)^2$. This follows by Jensen's inequality.
2. $\mathbb{E}[(u \cdot x_t - y_t)^2] = \mathbb{E}[(u \cdot x - y)^2]$. Because (x_t, y_t) and (x, y) are drawn from $\mathcal{D}$.
3. $\mathbb{E}[(w_t \cdot x_t - y_t)^2] = \mathbb{E}[(w_t \cdot x - y)^2]$. Because (x_t, y_t) and (x, y) are drawn from $\mathcal{D}$, and w_t is independent of (x_t, y_t).

Now we calculate that:

$$\begin{aligned}
\mathbb{E}_S[R_v] &= \mathbb{E}[(v \cdot x - y)^2] \\
&\leq \frac{1}{m} \sum_{t=1}^m \mathbb{E}[(w_t \cdot x - y)^2] && \text{using 1} \\
&= \frac{1}{m} \sum_{t=1}^m \mathbb{E}[(w_t \cdot x_t - y_t)^2] && \text{using 3} \\
&\leq \frac{1}{m} \mathbb{E}\left[\frac{\sum_{t=1}^m (u \cdot x_t - y_t)^2}{1-\eta} + \frac{\|u\|_2^2}{\eta} \right] \\
&= \frac{1}{m} \mathbb{E}\left[\frac{\sum_{t=1}^m (u \cdot x - y)^2}{1-\eta} + \frac{\|u\|_2^2}{\eta} \right] && \text{using 2} \\
&= \frac{L_u}{1-\eta} + \frac{\|u\|_2^2}{\eta m}.
\end{aligned}$$

Taking the infimum over u gives the theorem statement. □

15 Density Estimation

In this section, we will introduce the problem of *density estimation*: modeling a probability distribution given samples $x \sim \mathcal{P}$. For example, suppose we want to learna classifier for speech recognition. We can model the distribution first, and then use this distribution to predict labels via Bayes rule. This is often called a "generative approach", and it is in contrast to the "discriminative approach" which directly tries to learn an accurate classifier.

15.1 Principle of Maximum Likelihood

To formalize the setup, suppose we are given samples $x_1, \ldots, x_m \sim \mathcal{P}$, which is a discrete distribution over $\mathcal{X}$. Also, we have a set of candidate distributions $\mathcal{Q}$. For any $q \in \mathcal{Q}$, we define $q(x)$ to be the probability of x under q.

Definition 21 (Likelihood)**.** We define the **likelihood of the data under** q to be the probability of seeing $x_1, \ldots, x_m$ under the distribution q:

$$L(q) = \prod_{i=1}^{m} q(x_i).$$

We would like to find the q that maximizes the likelihood of the data - this is called the principle of maximum likelihood.

Example 18. Let's try a simple example. Consider a weighted coin that gives 1 with probability p. We flip it m times and get h heads. Let $\mathcal{Q} = [0, 1]$. The likelihood of a given q is:

$$L(q) = q^h(1-q)^{m-h}.$$

To maximize the likelihood, we take the derivative, and set it equal to 0 to get $q = {}^h/_m$.

In general, maximizing the likelihood means:

$$\arg\max_q L(q) = \arg\max_q \sum_{i=1}^{m} \log q(x_i) = \arg\min_q \frac{1}{m} \sum_{i=1}^{m} -\log q(x_i).$$

The quantity $-\log q(x_i)$ measures how well q fits x_i; we typically call it the **log loss** function. Thus, maximizing the likelihood is equivalent to minimizing the empirical risk.

Definition 22. We define the **true risk** of a distribution q to be:

$$\mathbb{E}_{x \sim \mathcal{P}}[-\log q(x)] = RE(\mathcal{P}||q) + H(\mathcal{P}),$$

where RE and H are the relative entropy and entropy. Note that here $H(\mathcal{P})$ doesn't depend on q, so minimizing the true risk is equivalent to minimizing the relative entropy $RE(\mathcal{P}||q)$ over $q \in \mathcal{Q}$.

15.2 Maximum Entropy Modeling

Now let us consider a practical setting. Suppose you are a researcher trying to model the habitat of a butterfly species on an island: you have a sample of butterfly sightings as well as features associated with each siting (altitude, annual rainfall, temperature, etc.) Assume that there exists a true distribution $\mathcal{D}$ that properly models the species, with the sightings sampled from $\mathcal{D}$. Also assume that we can get every bit of data for each feature for the entire map (which is our domain $\mathcal{X}$.)

More formally, let $|\mathcal{X}| = N$ and let $x_1, \ldots, x_m \sim D$. Let there be n features $f_1, \ldots, f_n$.

We consider two approaches. As we will see, they are equivalent.

Approach 1: Maximum Entropy. The main idea is to solve for a "likely" distribution under some constraints on our features. For feature j, we define the true expectation under q as $\mathbb{E}_q[f_j(x)]$, as well as the empirical average $\hat{\mathbb{E}}[f_j(x)] = \frac{1}{m}\sum_{i=1}^m f_j(x_i)$. We would like our distribution q to satisfy the condition that for each j, $\mathbb{E}_q[f_j] = \hat{\mathbb{E}}[f_j]$.

Now, we want our distribution q to minimize relative entropy, and because we have no prior beliefs, we measure it against the uniform distribution. Thus, we want to minimize $RE(q||\text{Unif}) = \log N - H(q)$; or maximize $H(q)$. Thus, we write the constrained optimization problem:

$$\begin{aligned} \min_{q\in\mathcal{P}} \quad & H(q) \\ \text{subject to} \quad & \mathcal{P} = \left\{ q : \mathbb{E}_q[f_j] = \hat{E}[f_j] \text{ for all } j \in [n]. \right\} \end{aligned}$$

Approach 2: Exponential-Family/Gibbs Distributions. We can also approach this problem by assuming that distribution we are looking for has a particular form. We will use the **exponential family** or **Gibbs** distributions:

$$q(x) = \frac{\exp\left(\sum_{j=1}^n \lambda_j f_j(x)\right)}{Z_\lambda},$$

where Z_λ is the partition function that makes $q(x)$ a valid probability distribution. Call $\mathcal{Q}$ the set of distributions with this form.

Our task becomes to find the suitable values of λ_j that maximize the likelihood of the data, i.e., $\max_{q\in\mathcal{Q}} \sum_{i=1}^m \log q(x_i)$. Technically such a maximum may not exist, but in such a case there is a limit point of $\mathcal{Q}$ so we can instead find $\sup_{q\in\mathcal{Q}} \sum_{i=1}^m \log q(x_i) = \max_{q\in\overline{\mathcal{Q}}} \sum_{i=1}^m \log q(x_i)$.

As promised, we will show the equivalence of these two approaches.

Theorem 39. *The following are equivalent:*

1. $q^\star = \arg\max_{q\in\mathcal{P}} H(q)$.
2. $q^\star = \arg\max_{q\in\overline{\mathcal{Q}}} \sum_{i=1}^m \log q(x_i)$
3. $q^\star \in \mathcal{P} \cap \overline{\mathcal{Q}}$

Furthermore, any one of these three conditions uniquely determines $q^\star$.

From part (3) of the theorem, we can see that it is both necessary and sufficient to find an element in $\mathcal{P}$ and $\overline{\mathcal{Q}}$. This is especially useful in analyzing the convergence of algorithms for this setting. We will not prove this equivalent, but instead just prove the equivalence of the two approaches through duality.

Proof sketch. We will sketch the proof via Lagrange multipliers. The Lagrangian for the first problem can be written as:

$$\mathcal{L}(q,\lambda,\gamma) = \sum_x q(x)\log q(x) + \sum_{j=1}^n \lambda_j \left(\hat{\mathbb{E}}[f_j] - \sum_x q(x) f_j(x)\right) + \gamma\left(\sum_x q(x) - 1\right).$$

We minimize over the primal variables $q(x)$:

$$\frac{\partial \mathcal{L}}{\partial q(x)} = 1 + \log q(x) - \sum_j f_j(x) + \gamma \stackrel{\text{set}}{=} 0.$$

$$\Rightarrow \quad q(x) = \exp\left(\sum_{j=1}^{n} \lambda_j f_j(x) - \gamma - 1\right) = Z_\lambda^{-1} \exp\left(\sum_j \lambda_j f_j(x)\right),$$

where $Z_\lambda \stackrel{\text{def}}{=} e^{\gamma+1}$ is the normalization factor. Thus, we do get the exponential family distribution!

Plugging this back into $\mathcal{L}(q, \lambda, \gamma)$ and maximizing over the dual variables gives us:

$$\begin{aligned}
\max_{\lambda,\gamma} \min_q \mathcal{L}(q, \lambda, \gamma) &= \max_{\lambda,\gamma} \left\{ \sum_x q(x) \left(\sum_j \lambda_j f_j(x) - \log Z_\lambda \right) - \sum_j \lambda_j \sum_x q(x) f_j(x) + \sum_j \lambda_j \hat{\mathbb{E}}[f_j] \right\} \\
&= \max_\lambda \left\{ -\log Z_\lambda + \frac{1}{m} \sum_j \lambda_j \sum_i f_j(x_i) \right\} \\
&= \max_\lambda \left\{ \frac{1}{m} \sum_i \left(\sum_j \lambda_j f_j(x_i) - \log Z_\lambda \right) \right\} = \frac{1}{m} \sum_i \log q(x_i).
\end{aligned}$$

Thus, the solution maximizes the log likelihood, or equivalently minimizes the negative empirical risk. □

15.3 Solving the Optimization Problem

Now we will consider how to actually solve the optimization problem in order to find the optimal distribution $q^\star$. We will attempt to solve the exponential-family problem (which is essentially maximum likelihood). We write it in the following form:

$$\min_{\lambda \in \mathbb{R}^n} \quad L(\lambda) \stackrel{\text{def}}{=} -\frac{1}{m} \sum_{i=1}^{m} \log q_\lambda(x_i) \qquad \text{(the negative log-likelihood)}$$

$$\text{subject to} \quad g_\lambda(x) = \sum_{j=1}^{n} \lambda_j f_j(x) \text{ and } q_\lambda(x) = Z_\lambda^{-1} \exp(g_\lambda(x)).$$

At a high level, we desire an iterative algorithm that converges to a good parameter vector λ. In particular, we want to design an update rule "compute λ_{t+1} from λ_t" such that $L(\lambda_t) \to \inf_\lambda L(\lambda)$. We will come up with an approximation to the difference $L(\lambda_{t+1}) - L(\lambda_t)$ and exactly minimize the approximation.

First, we will make a simplifying assumption.

Assumption. For each $x \in \mathcal{X}$, the features lie on the probability simplex. That is:

$$\sum_{j=1}^{n} f_j(x) = 1 \text{ and } f_j(x) \geq 0 \text{ for all } j.$$

This assumption holds without loss of generality! To see this:

1. Let $c_j = \min_x f_j(x)$ for each feature j. Then make the transformation $f'_j(x) = f_j(x) - c_j$, so that $g'_\lambda(x) = \sum_j \lambda_j f'_j(x) = g_\lambda(x) + \sum_j \lambda_j c_j$. We have only added a constant to g_λ so $q'_\lambda(x) = q_\lambda(x)$ because the change has been normalized.

2. Rescale each feature to $[0, 1/n]$ by multiplying by $b_j \stackrel{\text{def}}{=} 1/(n \max_x f_j(x))$. Scaling does not change which distributions can be represented, because we can always push the scaling into the λ_i.

3. We have nonnegative features that sum up to at most 1 for each x. We can add a dummy feature $f_0(x) = 1 - \sum_j f_j(x)$ that takes care of the slack. The term $\lambda_0(1 - \sum_j f_j(x))$ is added to $g_\lambda(x)$, but this does not limit the functions that can be represented because λ_0 is just a constant and the terms $-\lambda_0 f_j(x)$ can be absorbed into $\lambda_j f_j(x)$.

15.3.1 Step 1: bound the change in likelihood

Now we look for an approximation for the change in loss $L(\lambda') - L(\lambda)$. For notational convenience we define $\alpha = \lambda' - \lambda$. We calculate that:

$$\begin{aligned}
\Delta L &\stackrel{\text{def}}{=} L(\lambda') - L(\lambda) \\
&= -\frac{1}{m}\sum_{i=1}^{m} \log q_{\lambda'}(x_i) + \frac{1}{m}\sum_{i=1}^{m} \log q_\lambda(x_i) \\
&= -\frac{1}{m}\sum_{i=1}^{m} \log\left(Z_{\lambda'}^{-1} e^{g_{\lambda'}(x_i)}\right) + \frac{1}{m}\sum_{i=1}^{m} \log\log\left(Z_\lambda^{-1} e^{g_\lambda(x_i)}\right) \\
&= \frac{1}{m}\sum_{i=1}^{m} (g_\lambda(x_i) - g_{\lambda'}(x_i)) + \log\frac{Z_{\lambda'}}{Z_\lambda}.
\end{aligned}$$

We bound the first term:

$$\begin{aligned}
\frac{1}{m}\sum_{i=1}^{m} (g_\lambda(x_i) - g_{\lambda'}(x_i)) &= \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n} \left(\lambda_j f_j(x) - \lambda'_j f_j(x)\right) \\
&= -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n} \alpha_j f_j(x) \\
&= -\sum_{j=1}^{n} \alpha_j \hat{\mathbb{E}}[f_j].
\end{aligned}$$

Now we bound the second term:

$$\begin{aligned}
\frac{Z_{\lambda'}}{Z_\lambda} &= \frac{\sum_{x\in\mathcal{X}} \exp\left(g_\lambda(x) + \sum_{j=1}^n \alpha_j f_j(x)\right)}{Z_\lambda} \\
&= \sum_{x\in\mathcal{X}} q_\lambda(x) \exp\left(\sum_{j=1}^n \alpha_j f_j(x)\right) \\
&\leq \sum_{x\in\mathcal{X}} q_\lambda(x) \sum_{j=1}^n f_j(x) e^{\alpha_j} && \text{by Jensen's inequality } (f_j \text{ form a prob. dist.}) \\
&= \sum_{j=1}^n e^{\alpha_j} \mathbb{E}_{q_\lambda}[f_j].
\end{aligned}$$

Putting it all together, we have the bound:

$$\Delta L \leq -\sum_{j=1}^n \alpha_j \underbrace{\hat{\mathbb{E}}[f_j]}_{:=\hat{E}_j} + \log\left(\sum_{j=1}^n e^{\alpha_j} \underbrace{\mathbb{E}_{q_\lambda}[f_j]}_{:=E_j}\right).$$

Thus, the change in likelihood can be upper bounded by a quantity which depends both on the empirical average feature $\hat{E}_j$ as well as the expected average feature E_j under distribution q_λ.

15.3.2 Step 2: minimize the approximation

We have the approximation $B(\alpha) = -\sum_j \alpha_j \hat{E}_j + \ln\left(\sum_j \exp(\alpha_j) E_j\right)$. Taking derivatives w.r.t α_j to minimize:

$$\frac{\partial B}{\partial \alpha_j} = -\hat{E}_j + \frac{E_j e^{\alpha_j}}{\sum E_j e^{\alpha_j}} \stackrel{\text{set}}{=} 0. \Rightarrow \alpha_j = \log \frac{\hat{E}_j}{E_j}.$$

To conclude, we have constructed the iterative update on the dual variables λ:

$$\boxed{\lambda_j^{(t+1)} = \lambda_j^{(t)} + \log \frac{\hat{\mathbb{E}}[f_j]}{\mathbb{E}_{q_{\lambda^{(t)}}}[f_j]}}.$$

This expression can be readily computed as long as we can compute the expectation of f_j over q_λ (in time linear in $|\mathcal{X}| = N$).

15.3.3 Intuition

To gain some intuition, we can view this update rule in terms of the primal space of distributions. Let $p_t = q_{\lambda_t}$. Our distribution update is of the form:

$$p_{t+1}(x) \propto p_t(x) \prod_{j=1}^{n} \left(\frac{\hat{\mathbb{E}}[f_j]}{\mathbb{E}_{p_t}[f_j]} \right)^{f_j(x)}.$$

Recall that we want to reach a distribution $q^\star \in \mathcal{P} \cap \overline{\mathcal{Q}}$. Specifically if $q^\star \in \mathcal{P}$, then we must have $E_{q^\star}[f_j] = \hat{\mathbb{E}}[f_j]$ for all j. If all of these equations are satisfied, then we are at a fixed point. Otherwise, suppose that $\mathbb{E}_{p_t}[f_j] < \hat{\mathbb{E}}[f_j]$. This update rule will increase the probability distribution mass when $f_j(x)$, thus increasing $\mathbb{E}_{p_{t+1}}[f_j]$.

15.4 Convergence Guarantees

We will show the following theorem:

Theorem 40. $p_t \to q^\star$.

Proof. We will first define an auxiliary function A, and show that if such a function exists then p_t must converge to $q^\star$. Lastly, we will show that the approximation we chose satisfies the criteria for the auxiliary function.

Definition 23. A function defined on the probability simplex $A : \Delta_{\mathcal{X}} \to \mathbb{R}$ is an **auxiliary function** if:

1. A is continuous.
2. $L(\lambda_{t+1}) - L(\lambda_t) \leq A(p_t) \leq 0$.
3. If $A(p) = 0$ then $p \in \mathcal{P}$.

Claim. If an auxiliary function A exists for the process λ_t then $p_t \to q^\star$.

Proof. Note that the log-loss $L(\lambda)$ is lower bounded by 0. We know that the values $L(\lambda_t)$ must be monotonically decreasing by property (2), and therefore $L(\lambda_{t+1}) - L(\lambda_t) \to 0$. This implies that $A(p_t) \to 0$ also.

Now assume that the limit of p_t exists. We want to show that $p = \lim_{t\to\infty} p_t \in \mathcal{P} \cap \overline{\mathcal{Q}}$. Trivially, $p \in \overline{\mathcal{Q}}$ because $p_t \in \mathcal{Q}$. By property (1), we see that:

$$A(p) = A(\lim_{t\to\infty} p_t) = \lim_{t\to\infty} A(p_t) = 0.$$

Therefore by property (3) we also have $p \in \mathcal{P}$, so $p \in \mathcal{P} \cup \overline{\mathcal{Q}} = q^\star$ as desired.

Lastly, to tidy up, note that we did not need to assume that the limit existed. The p_t live in the probability simplex, a compact space. Therefore there exists a convergent subsequence $p_{t_k} \to q^\star$. The sequence p_t can only have a single limit point (by uniqueness of $q^\star$, so the entire sequence actually converges to $q^\star$.) □

Lastly, we show that the auxiliary function A exists. We simply define $A(p_t) = -\sum_j \alpha_j \hat{\mathbb{E}}[f_j] + \ln\left(\sum_j \exp(\alpha_j)\mathbb{E}_{p_t}[f_j]\right)$ and check that it satisfies the properties.

First, we note that when we plugged in the choice of $\alpha_j = \log(\hat{\mathbb{E}}[f_j]/\mathbb{E}_{p_t}[f_j])$, we enforced that $\sum_{j=1} e_j^{\alpha}\mathbb{E}_{p_t}[f_j] = 1$. Therefore, we see that:

$$A(p_t) = -\sum_{j=1}^{n} \hat{E}[f_j] \log \frac{\hat{\mathbb{E}}[f_j]}{\mathbb{E}_{p_t}[f_j]} = -RE(\hat{\mathbb{E}}[f]||\mathbb{E}_{p_t}[f]).$$

From here, the three properties are more or less obvious. Property 1 is satisfied because relative entropy is continuous. Property 2 is satisfied because relative entropy is nonnegative. Property 3 is satisfied because when $A(p) = 0$, we must have $\hat{\mathbb{E}}[f] = \mathbb{E}_p[f]$. □

16 Online Log-loss

Essentially, the problem we analyzed in density estimation was minimizing a log-loss in the *batch setting*, which was the maximum likelihood estimator. Now, we will consider the online log-loss problem, where our task is to output distributions q_t and do well against a best expert.

More formally, let N be the number of experts. At each round $t = 1, \ldots, T$, the learner receives a distribution $p_{t,i}$ from expert i. The learner must combine these predictions into a distribution q_t. The world then responds with a point $x_t \in \mathcal{X}$, and the learner suffers loss $= -\log q_t(x_t)$. We would like low regret against the best expert, that is:

$$-\sum_{t=1}^{T} \log q_t(x_t) \leq \min_{i \in [N]} -\sum_{t=1}^{T} \log p_{t,i}(x_t) + \text{ small.}$$

16.1 Universal Compression Interpretation

From our previous discussion, density estimation is a motivation for studying the online log-loss problem. Another very different motivation for online log-loss comes from the field of *coding theory*.

Consider the following scenario. Suppose Alice wants to send Bob a single English letter from $\mathcal{X} = \{a, b, \ldots\}$. They both know the distribution of each letter $p(x)$. It is well known that the optimal way to encode x is using $-\log_2 p(x)$ bits, which will achieve the minimum average coding length $H(x)$.

What if Alice wants to send a string of letters? Because the prediction for the next letter depends on the previous one, Alice can assign different probabilities to each letter given the previous letters already sent, potentially using less bits. Mathematically, Alice's task is to estimate the distribution of x_t given the sequence x_1^{t-1}.

To put this into the framework of online log-loss, suppose Alice was given N coding methods (the "experts") which predict $p_{t,i}(x_t)$ to be the probability of x_t conditioned on receiving x_1^{t-1}. Alice does not know which coding method is best ahead of time, but she wants

to combine these coding methods such that the coding length is nearly as short as the best coding method. In this analogy:

- $-\sum_{t=1}^{T} \log q_t(x_t)$ is the number of bits Alice requires to transmit the entire message.
- $-\sum_{t=1}^{T} \log p_{t,i}(x_t)$ is the number of bits that coding method i requires.

Therefore, if we are able to show low regret for this online problem, it would imply that for any message, the algorithm Alice uses will be near optimal (hence "universal compression").

16.2 Bayes Algorithm

We will introduce an algorithm called the Bayes Algorithm and show that it has low regret for the online log-loss problem:

- Initialize weights $w_{1,i} = 1/N$ for all $i \in [N]$.
- for $t = 1, \ldots, T$:
  1. Expert i predicts $p_{t,i}$ over $\mathcal{X}$.
  2. The leaner combines predictions into distrubtion $q_t(x) \stackrel{\text{def}}{=} \sum_{i=1}^{N} w_{t,i} p_{t,i}(x)$.
  3. The world response with $x_t \in \mathcal{X}$.
  4. The learner suffers loss $= -\log q_t(x_t)$.
  5. The learner updates weights $w_{t+1,i} = Z^{-1} w_{t,i} p_{t,i}(x_t)$ for all $i \in [N]$, where Z is a normalization term.

Note that the update rule (5) is precisely a multiplicative weights update of the general form $w_{\text{new}} = Z^{-1} w_{\text{old}} \beta^{\text{loss}}$, with $\beta = 1/e$.

16.2.1 Intuition for update rule

First, we will give some intuition for the update rule as a Bayes update. We will view the probability measure as conditional probabilities:

$$p_{t,i}(x_t) = p_i(x_t | x_1^{t-1}) \text{ for all } i, \quad q_t(x_t) = q(x_t | x_1^{t-1}).$$

In reality, $p_{t,i}$ are distributions the learner receives from black-box predictors, and q is the distribution that algorithm is predicting. Nonetheless, this viewpoint will be helpful.

Furthermore, we will pretend that the data is random according to the process to be described - but the formal regret analysis will hold for any sequence of data.

Data generating process.

1. An expert $i^\star$ is chosen $\sim \mathsf{Unif}(N)$, i.e., $\mathsf{Pr}[i^\star = i] = 1/N$.
2. We pretend $(x_1, \ldots, x_T)$ is generated by $p_{i^\star}$: $\mathsf{Pr}[x_t | x_1^{t-1}] = p_{i^\star}(x_t | x_1^{t-1})$.

Naturally, the algorithm's prediction $q(x_t|x_1^{t-1})$ is defined to be the conditional probability of x_t given x_1^{t-1} according to this random process, which we can solve as:

$$\begin{aligned} q(x_t|x_1^{t-1}) &\stackrel{\text{def}}{=} \mathsf{Pr}[x_t|x_1^{t-1}] \\ &= \sum_{i=1}^{N} \mathsf{Pr}[i^\star = i|x_1^{t-1}]\,\mathsf{Pr}[x_t|i^\star = i, x_1^{t-1}] \\ &= \sum_{i=1}^{N} w_{t,i}\,\mathsf{Pr}[x_t|i^\star = i, x_1^{t-1}]. \qquad \text{(we let } w_{t,i} \stackrel{\text{def}}{=} \mathsf{Pr}[i^\star = i|x_1^{t-1}]) \end{aligned}$$

From here, we will use Bayes Rule to solve for the precise form of $w_{t,i}$. First, we trivially have $w_{1,i} = \mathsf{Pr}[i^\star = i] = 1/N$. Then we can recursively compute $w_{t+1,i}$ as:

$$\begin{aligned} w_{t+1,i} &= \mathsf{Pr}[i^\star = i|x_1^t] \\ &= \frac{\mathsf{Pr}[i^\star = i|x_1^{t-1}] \times \mathsf{Pr}[x_t|x_1^{t-1}, i^\star = i]}{\mathsf{Pr}[x_t|x_1^{t-1}]} \\ &= \frac{w_{t,i}p_i(x_t|x_1^{t-1})}{q(x_t|x_1^{t-1})}. \end{aligned}$$

The denominator $q(x_t|x_1^{t-1})$ can be thought of as a normalization term, and therefore we have recovered the update rule (5).

16.2.2 Regret guarantee

We will now show the following regret bound:

Theorem 41. *The Bayes algorithm has a regret given by:*

$$-\sum_{t=1}^{T} \log q_t(x_t) \le \min_{i\in[N]} -\sum_{t=1}^{T} \log p_{t,i}(x_t) + \log N.$$

Proof. First we present 3 facts.

- We compute the probability of any sequence x_1^T according to the random process:

$$\mathsf{Pr}[x_1^T] = \prod_{t=1}^{T} \mathsf{Pr}[x_t|x_1^{t-1}] = \prod_{t=1}^{T} q(x_t|x_1^{t-1}).$$

- Likewise, we compute the probability of an given base expert predicting the sequence:

$$\mathsf{Pr}[x_1^T|i^\star = i] = \prod_{t=1}^{T} p_i(x_t|x_1^{t-1}).$$

- We can also lower bound the total probability of the sequence:

$$\Pr[x_1^T] = \sum_{i=1}^{T} \Pr[i^\star = i] \Pr[x_1^T | i^\star = i] \geq \frac{1}{N} \Pr[x_1^T | i^\star = i] \text{ for any } i.$$

Putting it all together, we see that:

$$\begin{aligned} -\sum_{t=1}^{T} \log q_t(x_t | x_1^{t-1}) &= -\log \prod_{t=1}^{T} q_t(x_1 | x_1^{t-1}) \\ &= -\log \Pr[x_1^T] \\ &\leq -\log \left(\frac{1}{N} \Pr[x_1^T | i^\star = i] \right) \\ &\leq -\sum_{t=1}^{T} \log p_i(x_t | x_1^{t-1}) + \log N. \end{aligned}$$

Since this holds for every expert i, it must hold for the best expert, proving the theorem. □

There are two ways to think about this theorem. First, dividing both sides by T, we see that the learner is a no-regret algorithm because $\log N/T \to 0$. Alternatively, dividing both sides by a constant to convert from $\ln$ to $\log_2$, we see that the algorithm will never use more than $\lg N$ bits than the best coding method for every sequence x_1^T. Consider another algorithm where the learner tries all N methods, picks the best one for x_1^T, and sends over the index of that method together with the encoding. This requires an overhead of $\lg N$ bits, so we get the bound on the RHS. But the Bayes algorithm is online and universal, which is particularly nice!

We can improve the Bayes algorithm by using a non-uniform prior when initializing the weights, and the resulting bound would be stated in terms of the prior:

$$-\sum_{t=1}^{T} \log q_t(x_t) \leq \min_{i \in [N]} \left[-\sum_{t=1}^{T} \log p_i(x_t | x_1^{t-1}) - \log \pi_i \right].$$

Thus, we should choose a prior π_i that gives higher probability to experts we think are good.

16.3 Shifting Experts

Now we add a complication to our setting. Previously, we had assumed that there existed one good expert throughout the entire time. But it might be the case that there are multiple experts that are good, and the best expert is switching from time to time. We would like an algorithm whose performance is almost as good as the best switching sequence of experts. In our information theoretic story, suppose Alice wanted to transmit some text which had

both English and Spanish. Then we would want to have one expert which is good at compressing English, and another that is good at compressing Spanish.

For the purpose of analysis, we want to compare regret against the best switching sequence of experts with at most k switches over a time horizon of T, again with N total "base experts".

One approach is to define a set of "meta-experts" which is defined as a sequence of base experts. An example is "choose expert 3 from time 1-17, expert 8 from 18-93,...". We can black-box apply the Bayes algorithm to yield that the regret is logarithmic in M, the number of meta experts. With simple combinatorial arguments, we can calculate $M \approx N^{k+1}T^k$, so the regret will be

$$\log M \approx \log N + k(\log N + \log T).$$

This is not so bad! The issue, however, is *computational*. Our algorithm must compute weights for every single meta-expert, which is infeasible.

Alternatively, we can construct a meta-expert for every sequence of base experts, i.e., a meta-expert is a vector $e = (e_1, \ldots, e_T)$ where each $e_t \in [N]$. This seems less efficient than the previous approach because there are more meta-experts, but we can exploit the structure to implement the Bayes algorithm efficiently. The downside lies in the fact that in this representation, we did not limit the number of switches (there can be up to $T-1$ switches now). We will fix this by imposing a prior that gives higher weights to meta-experts with smaller number of switches.

Definition of prior. Now we define the prior $\pi(e)$ that our algorithm will use for the generated meta expert $e^\star$.

- Pick $e_1^\star \sim \mathsf{Unif}(N)$.
- With probability $1-\alpha$, we do not switch, and with probability α, we switch uniformly:

$$\mathsf{Pr}[e_{t+1}^\star | e_t^\star] = \begin{cases} 1-\alpha & \text{if } e_{t+1}^\star = e_t^\star \\ \frac{\alpha}{N-1} & \text{otherwise.} \end{cases}$$

  This captures our intuition that experts which do not switch as frequently are more likely to occur.

Using this prior, we can already compute the regret (we will show how to implement the algorithm later). The regret of an meta-expert e is captured by the quantity:

$$-\log \pi(e) = -\log \left\{ \frac{1}{N}(1-\alpha)^{T-k-1} \left(\frac{\alpha}{N-1}\right)^k \right\}.$$

If we pick $\alpha \stackrel{\text{def}}{=} \frac{k}{T-1}$, we will get the bound that is $\approx \log N + k(\log N + \log T)$ (same as before).

The weight-share algorithm. Now we will show how to implement the algorithm without incurring space/time exponential in k. We will use a **weight-share algorithm** to do so. Recall that in the Bayes algorithm, we defined the quantity:

$$q(x_t|x_1^{t-1}) \stackrel{\text{def}}{=} \mathsf{Pr}[x_t|x_1^{t-1}] = \sum_{i=1}^{N} \mathsf{Pr}[e_t^\star = i|x_1^{t-1}] \times \mathsf{Pr}[x_t|e_t^\star = i, x_1^{t-1}].$$

Now, our task is to compute the values of $v_{t,i} \stackrel{\text{def}}{=} \mathsf{Pr}[e_t^\star = i|x_1^{t-1}]$ for all $i \in [N]$. At time step 1, we know that $v_{1,i} = 1/N$. Then:

$$v_{t+1,i} = \mathsf{Pr}[e_{t+1}^\star = i|x_1^t] = \sum_{j=1}^{N} \mathsf{Pr}[e_t^\star = j|x_1^t] \times \mathsf{Pr}[e_{t+1}^\star = i|e_t^\star = j, x_1^t].$$

The first term can be computed using Bayes rule (as done before):

$$\begin{aligned}\mathsf{Pr}[e_t^\star = j|x_1^t] &= \frac{\mathsf{Pr}[x_t|e_t^\star = j, x_1^{t-1}] \times \mathsf{Pr}[e_t^\star = j, x_1^{t-1}]}{\mathsf{Pr}[x_t|x_1^{t-1}]} \\ &= \frac{v_{t,j}p_j(x_t|x_1^{t-1})}{q(x_t|x_1^{t-1})} \qquad \text{plugging in our definitions}\end{aligned}$$

In the second term, conditioned on $\{e_t^\star = j\}$, the event $\{e_{t+1}^\star = i\}$ is independent of x_1^t. Actually, it is just $(1-\alpha)$ if $i = j$ and $\frac{\alpha}{N-1}$ otherwise.

Putting this togther, we get the following update rule:

$$\text{for all } i: \quad v_{t+1,i} = \sum_{j=1}^{N} \frac{v_{t,j}p_j(x_t|x_1^{t-1})}{q(x_t|x_1^{t-1})} \times \begin{cases} 1-\alpha & \text{if } i = j \\ \frac{\alpha}{N-1} & \text{else.} \end{cases}$$

From here, we see that this update rule for N weights is $\mathcal{O}(N)$, which is exactly the same as the run-time for the Bayes algorithm. In particular, there is no dependence on k, so our algorithm is computationally efficient! (We do actually need to know k, T beforehand to set α).

17 Portfolio Selection

Next, we will consider a simple mathematical framework for modeling a market of investment options, and we will analyze some investment strategies. The setup is as follows. Suppose we have N total investment options (stocks), indexed by i. Every day, we make an investment decision where we need to decide how to divide up our entire wealth between the N stocks. Without loss of generality, suppose we start with \$1 of wealth. Define S_t to be our total wealth at the start of day t, so $S_1 = 1$.

At the end of day t, we learn about the price relative for stock i as the quantity:

$$p_t(i) = \frac{\text{price of stock } i \text{ at end of day } t}{\text{price of stock } i \text{ at beginning of day } t}.$$

The learner's task is to pick an investment allocation amongst the stocks on each day t: i.e., output a vector $w_t \in \mathbb{R}^N$ where $w_t(i) \geq 0$ and $\sum_i w_t(i) = 1$, representing the fraction of wealth we allocate to stock i at the beginning of day t. The constraint that we always invest our entire wealth is fine because we can model putting money aside with a dummy stock that does not change.

Therefore, $S_t \cdot w_t(i)$ represents the total wealth of stock i at the start of day t, and at the end of the day, the wealth becomes $S_t \cdot w_t(i) \cdot p_t(i)$. Thus, we write our new total wealth as:

$$S_{t+1} = \sum_{i=1}^{N} S_t \cdot w_t(i) \cdot p_t(i) = S_t(w_t \cdot p_t) = \prod_{t=1}^{T} (w_t \cdot p_t),$$

unrolling the recurrence and using the fact that $S_1 = 1$.

The learner wants to maximize this quantity, or equivalently minimize the negative log of this quantity, $\sum_{t=1}^{T} -\log(w_t \cdot p_t)$. We write the online learning problem as:

For $t = 1, \ldots, T$:

- The learner chooses investment allocation w_t (subject to constraints).
- The world responds with price relatives p_t.
- The learner experiences loss $= -\log(w_t \cdot p_t)$.

17.1 Applying the Bayes Algorithm

First, we will analyze an algorithm that gets low regret *compared to the best stock*. Essentially, we will just apply the Bayes algorithm previously introduced.

Theorem 42. *The Bayes algorithm yields the following regret bound for the portfolio selection problem:*

$$-\sum_{t=1}^{T} \log(w_t \cdot p_t) \leq \min_{i \in [N]} \left\{ -\sum_{t=1}^{T} \log p_t(i) \right\} + \log N.$$

Proof. We will need to shoehorn the problem into the log-loss problem where the Bayes algorithm was used.

First assume that there exists some C such that $p_t(i) \in [0, C]$ for all t and i. The domain of our experts will be $\tilde{X} = \{0, 1\}$. The N experts are defined by:

$$\tilde{p}_{t,i}(1) = \frac{p_t(i)}{C} \text{ and } \tilde{p}_{t,i}(0) = 1 - \frac{p_t(i)}{C}.$$

On every round, we give $\tilde{x}_t = 1$ to the Bayes algorithm and receive weights $\tilde{w}_{t,i}$, which we will use as weights $w_t(i) = \tilde{w}_{t,i}$. Thus, we have fully specified how to convert this problem into the log-loss problem.

For the regret bound, we need to calculate $\tilde{q}(\tilde{x}_t)$ (learner's loss suffered):

$$\tilde{q}_t(\tilde{x}_t) = \tilde{q}(1) = \sum_{i=1} \tilde{w}_{t,i}\tilde{p}_{t,i}(1) = \frac{w_t \cdot p_t}{C}.$$

Therefore, plugging into the previously shown regret bound for the Bayes algorithm and canceling the C's on both sides, we get the desired regret bound.

□

This bound is actually somewhat trivial, because it is simply the bound we would get by dividing our wealth equally and following a "buy and hold" strategy. To see this, note that the bound is of the form:

$$-\log(\text{wealth of learner}) \leq \log(\text{wealth of best stock}) + \log N.$$

Raising e by both sides we get:

$$(\text{wealth of learner}) \geq \frac{1}{N}(\text{wealth of best stock}),$$

which is what we would get with the "buy and hold" strategy. In fact, it is possible to show that the Bayes algorithm is precisely equivalent to "buy and hold" (we will not do so here).

17.2 The Universal Portfolio Algorithm

Our trivial bound motivates us to try to find something better. In investing, it is common to use a "rebalancing" strategy, where at the end of the day, the stock holdings are shifted to keep the *proportion* of their wealth invested in each stock constant. This is called the **constant rebalanced portfolio** or CRP. When the proportion of each stock is equal, it is called the **uniform CRP**. The intuition is that rebalancing causes us to "buy low and sell high".

We will define a CRP by a vector $b \in \mathbb{R}^N$, where $b_i \geq 0$ and $\sum_{i=1}^N b_i = 1$. The set of CRPs forms an $(N-1)$-dimensional simplex. We would like an algorithm which has low regret compared to the best CRP; even though there are infinitely many CRPs this is still possible using the **universal portfolio** (UP) algorithm.

The idea is to invest an infinitesimally small portion of our total wealth according to each CRP. We will need to calculate $w_t(i)$ to serve as the weights in UP algorithm. Let us define Δ to be the set of all CRPs:

$$\Delta \stackrel{\text{def}}{=} \{\text{all CRPs}\} = \{b \in [0,1]^N : \sum_{i=1}^N b_i = 1\}.$$

Let $\delta\mu(b)$ denote the portion of our wealth invested according to b. For such a fixed b, observe that the amount of wealth invested according to b after $t-1$ days is $\prod_{s=1}^{t}(b \cdot p_s) \cdot d\mu(b)$. The fraction of stock i we hold due to b at the beginning of day t is $b_i \prod_{s=1}^{t}(b \cdot p_s) \cdot d\mu(b)$. So the weight vector can be written as:

$$w_t(i) = \frac{\int_{b\in\Delta} b_i \prod_{s=1}^{t}(b \cdot p_s) \cdot d\mu(b)}{\int_{b\in\Delta} \prod_{s=1}^{t}(b \cdot p_s) \cdot d\mu(b)},$$

where the integrals are taken over the space of all possible CRPs. The numerator is the total wealth invested in stock i on day t, and the denominator is the total wealth of the algorithm at the start of day t.

We have the following regret guarantee for the UP algorithm:

Theorem 43. *After T rounds:*

$$(\textit{wealth of UP}) \geq \frac{1}{(T+1)^{N-1}} \times (\textit{wealth of best CRP}).$$

Proof. We will actually prove a weaker version of this bound with the constant $1/e$ on the RHS.

Let $b^\star$ be the best CRP in hindsight. We will first argue that the CRPs b which are close to $b^\star$ make almost as much money as $b^\star$, and then we will argue that a decent amount of money is invested in strategies that lie within that neighborhood.

Step 1. Define $\mathcal{N}(b^\star)$ to be the neighborhood of $b^\star$ with parameter $\alpha \in [0,1]$:

$$\mathcal{N}(b^\star) = \{(1-\alpha)b^\star + \alpha z : z \in \Delta\}.$$

Consider some $b \in \mathcal{N}(b^\star)$. Then:

$$\begin{aligned} b &= (1-\alpha)b^\star + \alpha z \\ \Rightarrow b \cdot p_t &= (1-\alpha)b^\star p_t + \alpha z \cdot p_t \\ &\geq (1-\alpha)b^\star p_t \\ \Rightarrow \prod_{t=1}^{T}(b \cdot p_t) &\geq (1-\alpha)^T \prod_{t=1}^{T}(b^\star \cdot p_t). \end{aligned}$$

In other words:

$$(\text{wealth of } b) \geq (1-\alpha)^T (\text{wealth of } b^\star).$$

Step 2. Now we measure the proportion of wealth invested in $\mathcal{N}(b^\star)$ by relating the volume:

$$\text{Vol}(\mathcal{N}(b^\star)) = \alpha^{N-1}\text{Vol}(\Delta)$$

(this follows because the neighborhood is scaled by α).

Therefore, because we initially uniformly invest, then the initial fraction invested in $\mathcal{N}(b^\star)$ is α^{N-1}.

Step 3. The total wealth of our algorithm is lower bounded by the wealth generated by only strategies in $\mathcal{N}(b^\star)$. Combining the previous two steps, we see that:

$$\begin{aligned}(\text{wealth of UP}) &\geq \alpha^{N-1}(1-\alpha)^T(\text{wealth of best CRP } b^\star) \\ &\geq \frac{1}{e(T+1)^{N-1}}(\text{wealth of best CRP } b^\star) \qquad \text{taking } \alpha = (T+1)^{-1}.\end{aligned}$$

Thus, we have shown the regret guarantee. □

18 Learning and Game Theory

We will briefly describe the setting of game theory. Define a game $\mathcal{G}$ as follows. The game has m players; each player $i \in [m]$ is endowed with an action set $\mathcal{A}_i$ with $n_i := |\mathcal{A}_i|$ actions. Let $\mathcal{A} := \prod_{i\in[m]} \mathcal{A}_i$ denote the joint action space. Each player $i \in [m]$ also has a loss function $\mathcal{L}_i : \mathcal{A} \to [0,1]$ which has the following interpretation: if the action profile (actions of all the players) are $a = (a_1, a_2, \ldots, a_m)$, then player i suffers loss $\mathcal{L}_i(a)$. In general, we allow for players to play *distributions* over actions, and we denote $x_i \in \Delta(\mathcal{A}_i)$ to be a probability distribution. We will denote $x = (x_1, \ldots, x_m)$ to be the strategy profile when each player plays a distribution x_i. It is also convenient to introduce the notation: for $x = (x_1, \ldots, x_m)$, we denote x_{-i} to be the strategy profile after removing the ith strategy x_i (a_{-i}) is defined similarly). The goal of each player is to pick a strategy x_i to minimize their expected loss.

18.1 2-player zero-sum games

First, we will discuss 2-player zero-sum games.

Definition 24 (2-player zero-sum game)**.** In the 2-player zero-sum game, there are two players, ROW and COL. The losses have the property that for any $a_1 \in \mathcal{A}_1, a_2 \in \mathcal{A}_2$, we have $\mathcal{L}_1(a_1, a_2) = -\mathcal{L}_2(a_1, a_2)$.

For two player games, it will be convenient to think of ROW as minimizing expected loss and COL as maximizing expected payoff.

For simplicity, we denote $L(a_1, a_2)$ to be the loss that ROW gets for action profile (a_1, a_2); likewise, $L(x_1, x_2)$ denotes the expected loss that ROW gets for strategy profile (x_1, x_2). We can write out the loss in a simple form; let $M \in [0,1]^{n_1 \times n_2}$ be defined as: $M_{ij} := L(a_1 = i, a_2 = j)$. Then we have $L(x_1, x_2) = x_1^\top M x_2$.

First, we make the basic observation that:

$$\max_{x_2} \min_{x_1} L(x_1, x_2) \leq \min_{x_1} \max_{x_2} L(x_1, x_2).$$

This is because it is always better to play second (once you have observed what the other player has done).

Now we discuss a notion of optimal strategy:

Definition 25 (Minimax-optimal strategy)**.** A minimax-optimal strategy is a (randomized) strategy that has the best guarantee on its expected loss over the choices of the opponent.

Theorem 44 (von Neumann 1928)**.** *Every 2-player zero-sum game has a unique value V. The minimax optimal strategy for* ROW *guarantees* ROW*'s expected loss is at most V, and the minimax optimal strategy for* COL *guarantees* COL*'s expected gain is at least V:*

$$\max_{x_2} \min_{x_1} L(x_1, x_2) = \min_{x_1} \max_{x_2} L(x_1, x_2) = V.$$

Now we will give an algorithmic proof of the minimax theorem.

Proof. Suppose the minimax theorem was false; that is, for some game G we have

$$\min_{x_1} \max_{x_2} L(x_1, x_2) =: V_R > \max_{x_2} \min_{x_1} L(x_1, x_2) =: V_C$$

This means that:

- If ROW commits first, the COL player can make ROW get at least V_R loss.
- If COL commits first, the ROW player can get at most V_C loss.

Define $\delta = V_R - V_C > 0$.

Consider playing the RWM algorithm (or any regret minimizing algorithm) as ROW, against COL who plays optimally against ROW's distribution. Recall that in the RWM algorithm, in T steps, we have:

$$\mathbb{E}\text{ RWM's loss} \leq \text{best row in hindsight} + 2\sqrt{T \log n}.$$

However, also note that:

$$\begin{aligned} \text{best row in hindsight} &\leq T \cdot V_C \\ \mathbb{E}\text{ RWMA's loss} &\geq T \cdot V_R. \end{aligned}$$

Therefore, when $T > 4\log(n)/\delta^2$, we have a contradiction. □

What happens if two regret minimizers play each other? In this case, their time-average strategies must approach minimax optimality. Suppose ROW's time-average was far from minimax optimal (say, getting $V + \Delta$), then COL in hindsight has a strategy that substantially beats the value of the game. So by COL's no-regret guarantee, in actuality they must be substantially beating the value of the game. In other words, ROW is substantially doing worse than the value. This contradicts the no-regret guarantee for ROW.

18.2 Nash Equilibrium

Now we will discuss general-sum games. For simplicity, we will continue to work with 2 player games, but these ideas extend to m-player games. In general-sum games, we can get "win-win" and "lose-lose" situations i.e., ROW and COL's payoff doesn't have to sum up to 0 for every pair of actions that ROW and COL play. For example, considering driving on the left/right: if we have left/right, then they will have a crash and both receive negative payoff. But if they drive left/left or right/right, they will receive a positive payoff. include picture

Definition 26 (Nash Equilibrium, 2-player)**.** A *Nash Equilibrium* (NE) is a pair of strategies (x_1, x_2), such that neither player has incentive to deviate on their own.

We say that a Nash Equilibrium is **pure** if each strategy is deterministic, and **mixed** otherwise.

What does this mean in equations? Let $\pi = x_1 \times x_2$. Then π is a Nash equilibrium if for all $i \in \{1, 2\}$ and any $a_i' \in \mathcal{A}_i$:

$$\mathop{\mathbb{E}}_{a\sim\pi}\left[\mathcal{L}_i(a)\right] \leq \mathop{\mathbb{E}}_{a_{-i}\sim\pi_{-i}}\left[\mathcal{L}_i(a_i', a_{-i})\right]. \tag{5}$$

In our driving car example, there are 3 NE: left/left, right/right, and both players 50-50. The first two are pure NE, while the last one is a mixed NE.

Theorem 45 (Nash 1950)**.** *Any general-sum game must have at least one such equilibrium.* (*This equilibrium might be mixed*).

Corollary 46. *This yields the minimax theorem for 2-player zero-sum games as a corollary.*

Proof. Pick some Nash equilibria and let V be the value to the row player in that equilibrium. Since it's a Nash equilibria, neither player can do better even knowing the randomized strategy their opponent is playing, so they are both playing minimax optimal. □

Computing Nash can be hard. We have previously seen that in 2-player zero-sum games, computing an (approximate) NE is easy: we can just run no-regret learning algorithms for each player and take the product distribution of the time-average distributions for each player.

Another way to do so is via *fictitious play*, where instead of each player running a no-regret algorithm, at each round they pick the action which has the best loss in hindsight (up til time t). In learning theory, this is equivalent to the "follow the leader" (FTL) algorithm. It is known that fictitious play also converges to an NE, but the rate of convergence may be exponentially slow [DP14].

More generally, we would like to leverage tools we have developed in *online learning* to compute NE efficiently. Let us formalize this framework:

for $t = 1, 2, \ldots T$:

- Every player $i \in [m]$ plays a mixed strategy $x_i^{(t)}$.
- Every player $i \in [m]$ receives a loss vector $\ell_i^{(t)}$ such that $\ell_i^{(t)})[j] := \mathbb{E}_{a_{-i} \sim x_{-i}^{(t)}} [\mathcal{L}_i(j, a_{-i})]$.

The previous stated results on using no-regret learning algorithms and fictitious play can be stated as ways to pick the mixed strategies $x_i^{(t)}$ in the 2-player zero-sum setting in order to converge to NE.

So now we ask: can we efficiently compute Nash equilibrium in general sum games (using any algorithm)? Unfortunately the answer is no: [CDT09] showed that even for 2-player general sum games, computing approximate NE is PPAD-complete.

We will build some intuition for why regret minimization can fail even for 2-player general sum games. Consider the augmented Shapley game [Zin04]:

1. The first 3 rows/cols are the Shapley game (rock/paper/scissors but if both do the same action the both lose).
2. The 4th action is "play foosball" which is slightly negative if the other player is doing rock/paper/scissors but positive if other player does 4th action too.
3. The RWM algorithm will cycle among first 3 and have no regret, but do worse than only Nash Equilibrium of both playing foosball.

Furthermore, [BCM12] showed that regret minimization can fail to converge even in rank-1 games, which is interesting because one can find equilibria in such games efficiently.

18.3 A Hierarchy of Equilibrium

Since computing Nash equilibrium is hard, we might want to settle for computing other notions of equilibrium. Here, we will introduce two such notions: correlated equilibrium (CE) and coarse correlated equilibrium (CCE). The concept of correlated equilibrium is due to the work of Robert Aumann.

The key observation is that Nash equilibrium are always product distributions, i.e., each player acts independently of others. Aumann noted that we can potentially reach better equilibrium by introducing a trusted third-party. Think of the third-party as a traffic signal: the third-party draws an action profile from some distribution $\pi \in \Delta(\mathcal{A})$ and proposes each action a_i to the ith player. The players do not get to see the other actions a_{-i}, but must choose whether to follow the action or pick some other distribution. π need not be a product distribution; hence the name "correlated".

Definition 27 (Correlated Equilibrium, 2-player)**.** We say $\pi \in \Delta(\mathcal{A}_1 \times \mathcal{A}_2)$ is a correlated equilibrium such that if the trusted party picks one at random and tells each player their part, they have no incentive to deviate.

In equations, this means that for each player $i \in \{1, 2\}$, any $a_i \in \mathcal{A}_i$, $a_i' \in \mathcal{A}_i$:

$$\mathop{\mathbb{E}}_{a \sim \pi} [\mathcal{L}_i(a) \mid a_i] \leq \mathop{\mathbb{E}}_{a \sim \pi} [\mathcal{L}_i(a_i', a_{-i}) \mid a_i] . \tag{6}$$

Going back to the driving example, we see that there are additional correlated equilibrium which are not Nash equilibrium. For example, we can let π be evenly distributed over (L, L) and (R, R).

Now we introduce coarse correlated equilibrium, which is even less restrictive.

Definition 28 (Coarse Correlated Equilibrium, 2-player)**.** We say $\pi \in \Delta(\mathcal{A}_1 \times \mathcal{A}_2)$ is a correlated equilibrium such that if the trusted party picks one at random, and players have the choice to either (1) do what the trusted party tells them to do; or (2) not see the advice at all, each player would prefer the former.

In equations, this is the following. For each player $i \in \{1, 2\}$ and any $a' \in \mathcal{A}_i$:

$$\mathop{\mathbb{E}}_{a\sim\pi}\left[\mathcal{L}_i(a)\right] \leq \mathop{\mathbb{E}}_{a\sim\pi}\left[\mathcal{L}_i(a'_i, a_{-i})\right].$$

CE and CCE can be efficiently computed. Now we state results that show that CE and CCE can be efficiently computed by the online learning framework. They can also be computed via linear programming, but we will not discuss this in detail.

First, we reexamine our notions of regret:

1. *best expert* or *external* regret: We have already seen this. Given n strategies, compete with the best of them in hindsight.
2. *sleeping expert* or *regret with time-intervals*: Given n strategies, k properties. Let S_i be the set of days satisfying property i (which might overlap). We want to simultaneously achieve low regret over each S_i.
3. *internal* or *swap* regret: Like (2), except that $S_i =$ the set of days in which we chose strategy i. Formally for internal regret, you are allowed to change one item i to j, but in swap regret you define an optimal "rewiring" function $f : [n] \to [n]$ such that every time you played action j, it plays $f(j)$.

Proposition 47. If each player runs a low swap regret algorithm, the the empirical distribution of play is an approximate correlated equilibrium.

We will not prove this fact. However, we note that many low swap regret algorithms exist. In particular, [BM07] show how to convert low external regret algorithms to low swap regret algorithms.

Proposition 48. If each player runs a low external regret algorithm, the the empirical distribution of play is an approximate coarse correlated equilibrium.

Proof. This is more or less straightforward from the definition. Using the no-regret property, for each i, a'_i:

$$\frac{1}{T}\sum_{t=1}^{T}\mathcal{L}_i(a^{(t)}) \leq \frac{1}{T}\sum_{t=1}^{T}\mathcal{L}_i(a'_i, a^{(t)}_{-i}) + \frac{\varepsilon(T)}{T}.$$

Now let π_T be the uniform distribution over the action profiles $\{a^{(1)}, \ldots, a^{(T)}\}$. We immediately see that π_T has the properties to be a CCE. Note that π_T may not actually converge

to a single distribution, however all limit points of π_T must be CCE, since the simplex is closed and bounded. □

The relationship between NE, CE, and CCE. We have the following fact:

$$\mathrm{NE} \subset \mathrm{CE} \subset \mathrm{CCE}.$$

References

[BCM12] Maria-Florina Balcan, Florin Constantin, and Ruta Mehta. The weighted majority algorithm does not converge in nearly zero-sum games. 2012.

[BM07] Avrim Blum and Yishay Mansour. From external to internal regret. *Journal of Machine Learning Research*, 8(6), 2007.

[CDT09] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player nash equilibria. *Journal of the ACM (JACM)*, 56(3):1–57, 2009.

[DP14] Constantinos Daskalakis and Qinxuan Pan. A counter-example to karlin's strong conjecture for fictitious play. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 11–20. IEEE, 2014.

[Zin04] Martin Zinkevich. *Theoretical guarantees for algorithms in multi-agent settings*. Carnegie Mellon University, 2004.