Computational Learning Theory
10 : Mistake-Bounded Learning

Lecturer: Varun Kanade

So far we've mainly looked at settings where there is an underlying distribution over the data and we are given access to an oracle that provides random examples from this distribution. While this framework provides a useful way to analyse the behaviour of learning algorithms, it is not always the case that one may get independent training examples in practice. In reality, the distribution from which the data is generated may change over time. In this lecture, we will look at a specific learning framework that removes the requirement that data comes from a fixed distribution as (stochastically) independent examples.

1 Online Prediction Framework

We consider the setting where the learning algorithm is interacting with an *environment* and has to make predictions at discrete time-steps. Let X be an instance space and C a class of concepts.[1] The setup is as follows:

(a) At time t, the learning algorithm is presented an instance $\mathbf{x}_t \in X$.

(b) The learning algorithm makes a prediction $\widehat{y}_t \in \{0, 1\}$.

(c) The true label y_t is revealed and the learning algorithm is said to have made a mistake if $\widehat{y}_t \neq y_t$.

The process defined above repeats indefinitely for $t = 1, 2, \ldots$. How might one measure the performance of such a learning algorithm? For a learning algorithm L and infinite sequence $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$, for time $t \in \mathbb{N}$, define,

$$\mathsf{MISTAKES}\left(t; L, ((\mathbf{x}_i, y_i))_{i=1}^{\infty}\right) = \sum_{s=1}^{t} \mathbb{1}(\widehat{y}_s \neq y_s).$$

In the process defined above, the access L gets to the data $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ is sequential, where it has to make a prediction $\widehat{y}_t$ before seeing y_t. We will only consider sequences $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ for which there exists $c \in C$, such that $y_i = c(\mathbf{x}_i)$ for all i. We will say that C is learnable with a finite mistake bound B, if there exists an online learning algorithm L, that for every sequence $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ that satisfies for some $c \in C$ that for all i, $y_i = c(\mathbf{x}_i)$, satisfies for all $t \in \mathbb{N}$, $\mathsf{MISTAKES}(t; L, ((\mathbf{x}_i, y_i))_{i=1}^{\infty}) \leq B$.

It is worth making a couple of observations at this point. If X is finite and there is no restriction on the computational resources available to L, one can always trivially get a mistake bound of $|X|$. We will typically be interested in algorithms that are efficient in their use of space and time. We will define the notion of efficiency later, but first let us consider an example. We will design an algorithm for online learning CONJUNCTIONS in the mistake-bounded setting. The Algorithm is shown given in Alg. 1.

Theorem 1. CONJUNCTIONS *can be learnt online with mistake-bound $n + 1$. Furthermore, the running time of the algorithm is polynomial in n at each time t.*

[1] We will forgo the slightly cumbersome notational overhead of writing $X = \bigcup_{n \geq 1} X_n$ and $C = \bigcup_{n \geq 1} C_n$ and implicitly assume that there is a parameter n that captures the size of instances. Likewise, we assume that there is a function size(c) that gives the representation size of concepts.

Algorithm 1 Mistake-bounded algorithm for learning conjunctions

```
1: Input: Sequence $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ provided online.
2: Let $h_1 = z_1 \wedge \bar{z}_1 \wedge z_2 \wedge \bar{z}_2 \wedge \cdots \wedge z_n \wedge \bar{z}_n$        ▷ Start with conjunction of all literals
3: for $t \leftarrow 1, 2, \ldots,$ do
4:     Receive $\mathbf{x}_t$
5:     $\widehat{y}_t = h_t(\mathbf{x}_t)$
6:     Receive $y_t$
7:     if $\widehat{y}_t \neq y_t$ then
8:         Remove all $z_i$ from $h_t$ such that $x_{t,i} = 0$
9:         Remove all $\bar{z}_i$ from $h_t$ such that $x_{t,i} = 1$
10:        Let $h_{t+1}$ be the resulting conjunction
11:    end if
12: end for
```

Proof. First observe that because we start with all literals in the hypothesis conjunction and only drop literals where we are sure that the literal can't be part of the target conjunction, the only mistakes we make are of the form $y_t = 1$ and $\widehat{y}_t = 0$.

To begin, h_1 has $2n$ literals. When the first mistake occurs, exactly n literals are removed: for each i, exactly one of z_i or $\bar{z}_i$ is dropped. At every subsequent mistake at least one literal is dropped. Thus, the number of mistakes cannot exceed $n + 1$.

Finally, note that the algorithm is only maintaining a hypothesis conjunction and using it to make the prediction $\widehat{y}_t$. So the running time of the algorithm at each time-step is $O(n)$. □

Exercise: Show that the above bound is tight for this particular algorithm. What can you say about a general mistake bound for any algorithm for online learning CONJUNCTIONS?

1.1 Resource Constraints on Online Algorithms

In the most generous setting, we can allow the algorithm L to predict $\widehat{y}_t$ using any computable function of $(\mathbf{x}_1, y_1, \mathbf{x}_2, y_2, \ldots, \mathbf{x}_{t-1}, y_{t-1}, \mathbf{x}_t)$. For computationally efficient algorithms, we may require that this function be computable in time polynomial in n, size(c) and t. However, as we observed in the algorithm for learning CONJUNCTIONS, the algorithm did not need to store the entire history of observations, but the current hypothesis h_t was sufficient as a *sketch* of the history up to that point.

We consider space-bounded algorithms, where at time t, the algorithm maintains a state S_t, such that for each t, $|S_t| \leq \text{poly}(n, \text{size}(c))$. For an efficient algorithm, we will require that there are two polynomial time computable functions f and g, such that $\widehat{y}_t = f(S_t, \mathbf{x}_t)$ and $S_{t+1} = g(S_t, \mathbf{x}_t, y_t)$. Thus, the function f is used to make a prediction $\widehat{y}_t$ at time t, and g is used to update the state.

Note that the we can define $h_t : X \to \{0, 1\}$ as $h_t(\mathbf{x}) = f(S_t, \mathbf{x})$, thus essentially this is equivalent to the algorithm maintaining a hypothesis at each time t. Furthermore, because of the requirement that $|S_t| \leq \text{poly}(n, \text{size}(c))$, the total number of possible hypothesis is at most $2^{\text{poly}(n, \text{size}(c))}$, thus the algorithm in this case is making predictions using a hypothesis that comes from a fairly restricted class of hypotheses. We will refer to such algorithms as *efficient* online algorithms.

1.2 Conservative Online Learning

Definition 2 (Conservative Online Learner)**.** *We say that an online learning algorithm is conservative, if it only changes its prediction rule after making a mistake. Equivalently it only updates its state if it makes a mistake.*

Proposition 3. *If C is learnable with a mistake bound B using an online learning algorithm A, then C is learnable with mistake bound B using a conservative online learning algorithm. The conservative online learning algorithm is efficient if A is efficient.*

Proof. The proof of this result is relatively straightforward. We design an algorithm A' as follows. A' initialises itself exactly the same way as A does. Let S'_t be the state of the A' at time t and let $m(t)$ denote the number of mistakes made by A' up to (but no including) time t. We will simulate A on a subsequence of examples on which A' makes mistakes. We will maintain the invariant that $S'_t = S_{m(t)+1}$. Note that by definition $S'_1 = S_1$.

A' behaves as follows. If there is no mistake at time t, then $S'_{t+1} = S'_t$. If on the other hand a mistake is made, then we pass the example $\mathbf{x}_t$ to the simulation of A and set $S'_{t+1} = S_{m(t+1)+1}$. Clearly A' is conservative by definition. However, the prediction rule used by A' at time t is the same as the one used by A at time $m(t)+1$; as a result every time A' makes a mistake so does A. Since A has a mistake bound of B, so does A'. $\square$

The requirement that an online learning algorithm be conservative is a natural one and the above result shows that it is not a restrictive one. This result will be useful to establish that efficient mistake-bounded online learning implies PAC learning.

2 Relationships to Other Models of Learning

In the PAC learning framework, we have access to an example oracle $\mathsf{EX}(c, D)$ that when queried returns an example $(\mathbf{x}, c(\mathbf{x}))$ where $\mathbf{x} \sim D$. Earlier in the course, we also considered two other oracles, a membership query oracle, $\mathsf{MQ}(c)$, which when queried with $\mathbf{x}$, returns $c(\mathbf{x})$, and an equivalence query oracle, $\mathsf{EQ}(c)$, which when queried with a hypothesis h, either returns that $c \equiv h$ or returns a counterexample $\mathbf{x}$, such that $h(\mathbf{x}) \neq c(\mathbf{x})$. We will now relate mistake-bounded learning to learning using some of these other oracles.

2.1 Relationship to PAC Learning

Theorem 4. *If C is efficiently learnable with a mistake bound B using an online learning algorithm A, where $B \le \mathrm{poly}(n, \mathrm{size}(c))$, then C is efficiently PAC learnable.*

Proof. Without loss of generality, let A be a conservative online learning algorithm. We generate each example $(\mathbf{x}_t, y_t)$ at time t by querying the example oracle $\mathsf{EX}(c, D)$. Let h_t denote the hypothesis used by A at time t. Since A is conservative and has a mistake-bound of B, we need to consider no more than $B + 1$ distinct hypotheses.

We either stop the simulation of A after B mistakes have been made and output the hypotheses that is then guaranteed to be equal to the target c, or we stop the simulation if we simulate $s = \frac{1}{\epsilon}\log\frac{B}{\delta}$ steps without making a mistake. The probability that a hypothesis, h, with $\mathrm{err}(h) \ge \epsilon$ will go for s steps with examples drawn i.i.d from $\mathsf{EX}(c, D)$, without making a mistake is at most $(1-\epsilon)^s \le e^{-\epsilon s} \le \delta/B$. Thus, a simple union bound suffices to show the correctness.

We remark that the condition that $B \le (n, \mathrm{size}(c))$ together with the efficiency of A suffices to conclude that the resulting PAC learning algorithm is efficient. Note that the sample complexity of the resulting algorithm is $O\left(\frac{B}{\epsilon}\log\frac{B}{\delta}\right)$. $\square$

Remark 5. *If we wanted to allow non-efficient algorithms, but insist on polynomial sample complexity and polynomial-time evaluatability of the hypothesis class, we would still require* $B \leq \text{poly}(n, \text{size}(c))$ *and would require that the online learning algorithm* A *had a polynomial time prediction rule, even if the state update could potentially require more than polynomial time.*

2.2 Relationship to Learning Using Equivalence Queries

Proposition 6. *If* C *is learnable with a mistake bound* B *using an online algorithm* A*, then* C *can be learnt using* $\mathsf{EQ}(c)$ *only with at most* $B + 1$ *equivalence queries. Furthermore, if* A *is efficient so is the algorithm that learns using* $\mathsf{EQ}(c)$*.*

Proof. Let h_t be the hypothesis used by A at time t for $t \geq 1$. We will query $\mathsf{EQ}(c)$ with h_t: if we get that $c \equiv h_t$, then we are done, otherwise we get a counterexample $\mathbf{x}_t$ which forces A to make a mistake at time t. In fact, this forces A to make mistakes at every single time-step in the simulation. Thus, after B mistakes h_{B+1} will be identical to c. We can verify this by an additional query to $\mathsf{EQ}(c)$. □

Proposition 7. *If* C *is learnable using only* $\mathsf{EQ}(c)$ *and makes at most* Q *queries to* $\mathsf{EQ}(c)$*, then* C *is learnable with a mistake bound of* Q *using an online algorithm. Furthermore, if the algorithm that learns using* $\mathsf{EQ}(c)$ *is efficient, then so is the online algorithm.*

Proof. Let L be the learning algorithm that only uses $\mathsf{EQ}(c)$. At any point in its simulation when it is about to make an equivalence query, it has a hypothesis h, we will use h to make predictions in the online setting. If we make a mistake, we have successfully simulated the oracle $\mathsf{EQ}(c)$ to get a counterexample. After Q such misakes L has a hypothesis h that is equivalent to c and will make no further mistakes. Thus, the mistake bound is Q. □

Theorem 4 also follows using Proposition 6 and an exercise previously seen that simulates an equivalence oracle using $\mathsf{EX}(c, D)$. The results in these sections show that online learning with a finite mistake bound is at least as hard as PAC learning. In fact it can be shown that it is strictly harder in that for the class of linear threshold functions in general we cannot get a finite mistake bound. We will discuss this point further in Section 4. We will study the Perceptron algorithm in Section 4 that gives a finite mistake bound for learning linear threshold functions with a margin. In Section 5, we will study an algorithm that learns sparse disjunctions with a significantly improved mistake-bound than can be achieved directly using the Perceptron algorithm.

3 The Halving Algorithm and Some Examples

In this section, we will see some *information-theoretic* bounds for mistake-bounded online learning algorithms. We will not be concerned with computational efficiency but see how these compare to other notions such as the VC dimension.

Theorem 8. *For any finite concept class* C *over an instance space* X*, the Halving algorithm (Alg. 2) has a mistake bound of* $\log |C|$*.*

Proof. The proof is immediate. If there is a mistake at time t, then $|C_{t+1}| \leq |C_t|/2$. Thus, after $\log_2 |C|$ mistakes, we can have at most one concept left, which must be the target concept. □

It is also straightforward to see that $\mathsf{VCD}(C)$ is a lower bound for any achievable mistake bound for deterministic algorithms, as we can give any learning algorithm points from a shattered

Algorithm 2 Halving algorithm for online learning C

```
**Input**: Sequence $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ provided online.
Let $C_1 = C$
for $t \leftarrow 1, 2, \ldots,$ do
    Receive $\mathbf{x}_t$
    $\widehat{y}_t = \text{majority}\{c(\mathbf{x}_t) | c \in C_t\}$
    Receive $y_t$
    if $\widehat{y}_t \neq y_t$ then
        $C_{t+1} = \{c \in C_t \mid c(\mathbf{x}_t) = y_t\}$
    end if
end for
```

set and force it to make a mistake on every one of them. We do know that for finite C, $\mathsf{VCD}(C) \leq \log |C|$. We will now see some examples where the gap between $\mathsf{VCD}(C)$ and $\log |C|$ is large and show that it is possible for the mistake bound to lie at either end of this interval.

Dictators

Let $X = \{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$ be the instance space where $\mathbf{e}_i$ is the i^{th} basis vector which has 1 in the i^{th} co-ordinate and 0 everywhere else. Let C be the class of dictator functions $C = \{c_1, \ldots, c_n\}$, where $c_i(\mathbf{x}) = x_i$, i.e. the output of c_i is simply the i^{th} bit of $\mathbf{x}$ regardless of the remaining bits.[2] In this case, the mistake bound of the Halving Algorithm is 1, as is the $\mathsf{VCD}(C)$; obviously in this case $\log |C| = \log n$. As an exercise, you can show that if $X = \{0, 1\}^n$, then the mistake bound is indeed $\log n$ rather than 1.

Binary Search

Let $X = \{1, 2, \ldots, 2^n\} \subseteq \mathbb{N}$. Let C be the class of half intervals defined as $C = \{c_i | 1 \leq i \leq 2^n\}$, where,

$$c_i(x) = \begin{cases} 1 & \text{if } x \geq i \\ 0 & \text{otherwise} \end{cases}$$

In this case it is easy to see that $\mathsf{VCD}(C) = 1$, however the mistake-bound for any algorithm must be $\Theta(\log n) = \Theta(\log |C|)$.

4 Perceptron

The Perceptron algorithm is perhaps the most famous online learning algorithm. It was designed by Rosenblatt (1958) and the first proofs of its convergence were given by Block (1962) and Novikoff (1963).

For the purpose of this section it will be convenient to treat Boolean functions as taking values in $\{-1, 1\}$ and also letting $\text{sign}(0) = 1$. We consider homogenous halfspaces, or linear threshold functions passing through the origin, characterized by $\mathbf{w} \in \mathbb{R}^n$. This defines a threshold function $f_{\mathbf{w}} : \mathbb{R}^n \to \{-1, 1\}$ as,

$$f_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

[2]This terminology comes from social choice theory. The bits 0 and 1 can represent binary preferences of a group of n individuals and a dicator rule essentially uses the preference of the i^{th} individual, ignoring the rest. Obviously, the *majority* rule also lies in this framework of social choice theory.

The Perceptron algorithm is given as Alg. 3.

Algorithm 3 The Perceptron Algorithm

```
Input: Sequence $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ provided online.
Set $\mathbf{w}_1 = \mathbf{0} \in \mathbb{R}^n$
for $t \leftarrow 1, 2, \ldots,$ do
    Receive $\mathbf{x}_t$
    $\widehat{y}_t = \mathrm{sign}(\mathbf{x}_t \cdot \mathbf{w}_t)$
    Receive $y_t$
    if $\widehat{y}_t \neq y_t$ then
        $\mathbf{w}_{t+1} = \mathbf{w}_t + y_t \mathbf{x}_t$
    end if
end for
```

Before we formally analyse the algorithm, it is worth understanding geometrically what the algorithm is doing. Let $\mathbf{w}^\star$ be the true vector that defines the labels y_t. If the algorithm makes a mistake at time t it must be the case that $\mathbf{w}^\star \cdot \mathbf{x}_t$ and $\mathbf{w}_t \cdot \mathbf{x}_t$ have opposite signs. Thus adding $y_t \mathbf{x}_t$ to $\mathbf{w}_t$ has the effect of *rotating* $\mathbf{w}_t$ in the direction of $\mathbf{w}^\star$. This is reasonably correct intuition, but it is not perfectly accurate, it only does this on *average* as will be established by Lemmas 10 and 11. Rather than analyse the angles between $\mathbf{w}_t$ and $\mathbf{w}^\star$, it will be easier to show that the inner product between $\mathbf{w}_t$ and $\mathbf{w}^\star$ increases every time a mistake is made and that the length of $\mathbf{w}_t$ doesn't increase too much. This means that the increase in inner product is at least in part caused by the decrease in the angle and not merely by the increase in the length. We state and prove the following theorem.

Theorem 9. *Suppose $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ is a sequence such that for every t, $\|\mathbf{x}_t\|_2 \leq D$ and there exists $\mathbf{w}^\star \in \mathbb{R}^n$ with $\|\mathbf{w}^\star\|_2 = 1$ and $\gamma > 0$, such that for every t, $y_t(\mathbf{w}^\star \cdot \mathbf{x}_t) \geq \gamma$. Then the total number of mistakes made by the Perceptron Algorithm (Alg. 3) with input sequence $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ is bounded by D^2/γ^2.*

Proof. Let m_t denote the number of mistakes made by the Perceptron algorithm up to but not including time t. Using Lemmas 10 and 11, we have the following,

$$\begin{aligned}
m_t \gamma &\leq \mathbf{w}^\star \cdot \mathbf{w}_t && \text{Lemma 10} \\
&\leq \|\mathbf{w}^\star\|_2 \|\mathbf{w}_t\|_2 && \text{By the Cauchy-Schwarz Inequality} \\
&\leq \sqrt{m_t} D. && \text{Lemma 11}
\end{aligned}$$

This gives that $m_t \leq D^2/\gamma^2$ for every t.

□

Lemma 10. *Let m_t denote the number of mistakes made by the Perceptron algorithm (Alg. 3) up to, but not including, time t on a sequence $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ satisfying the conditions of Theorem 9. Then,*

$$\mathbf{w}_t \cdot \mathbf{w}^\star \geq \gamma m_t.$$

Proof. We prove this by induction. When $t = 1$, $\mathbf{w}_t = \mathbf{0}$ and $m_t = 0$, so this is clearly true.

Now suppose this is true for time t. If no mistake occurs at time t, then $m_{t+1} = m_t$ and $\mathbf{w}_{t+1} = \mathbf{w}_t$, so the inequality continues to hold. On the other hand, if there is a mistake, then

$m_{t+1} = m_t + 1$, and we have,

$$\begin{aligned}
\mathbf{w}_{t+1} \cdot \mathbf{w}^\star &= (\mathbf{w}_t + y_t\mathbf{x}_t) \cdot \mathbf{w}^\star \\
&\geq m_t\gamma + y_t(\mathbf{x}_t \cdot \mathbf{w}^\star) \\
&\geq (m_t + 1)\gamma = m_{t+1}\gamma.
\end{aligned}$$

This completes the proof. □

Lemma 11. *Let m_t denote the number of mistakes made by the Perceptron algorithm (Alg. 3) up to, but not including, time t on a sequence $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ satisfying the conditions of Theorem 9. Then,*

$$\|\mathbf{w}_t\|_2^2 \leq D^2 m_t.$$

Proof. We prove this by induction. When $t = 1$, $\mathbf{w}_t = \mathbf{0}$ and $m_t = 0$, so this is clearly true.

Now suppose this is true for time t. If no mistake occurs at time t, then $m_{t+1} = m_t$ and $\mathbf{w}_{t+1} = \mathbf{w}_t$, so the inequality continues to hold. On the other hand, if there is a mistake, then $m_{t+1} = m_t + 1$, and we have,

$$\begin{aligned}
\|\mathbf{w}_{t+1}\|_2^2 &= \|\mathbf{w}_t + y_t\mathbf{x}_t\|_2^2 \\
&\leq \|\mathbf{w}_t\|_2^2 + \|\mathbf{x}_t\|_2^2 + 2y_t\mathbf{w}_t \cdot \mathbf{x}_t \\
&\leq m_t D^2 + D^2 \\
&\leq (m_t + 1)D^2 = m_{t+1}D^2.
\end{aligned}$$

Above, in the second last step, we used the fact that $\|\mathbf{x}_t\|_2 \leq D$ and that $y_t(\mathbf{w}_t \cdot \mathbf{x}_t) < 0$ as the Perceptron algorithm made a mistake at time t. This completes the proof. □

Some Comments

Notice that for convergence, we require a *margin condition*: $y(\mathbf{w}^\star \cdot \mathbf{x}) \geq \gamma > 0$ for all points $(\mathbf{x}, y)$ seen by the algorithm. Clearly, the condition implies that y has the same sign as $\mathbf{w}^\star \cdot \mathbf{x}$, but it further says that the projection of $\mathbf{x}$ in the direction of $\mathbf{w}^\star$ must have length at least γ (as $\mathbf{w}^\star$ is a unit vector). This means that there is a region of width 2γ around the hyperplane $\{\mathbf{x} \mid \mathbf{w}^\star \cdot \mathbf{x} = 0\}$ in which we do not observe any data. This is what it means to say that the linear threshold function has a *margin* γ under the distribution. On Problem 3 of Sheet 6, you will essentially show a matching lower bound on the mistakes made by the Perceptron algorithm. It is in fact possible to show that no online algorithm can achieve a finite mistake bound without a margin condition for learning threshold functions. This can already be seen in the one-dimensional case by generalizing the binary search example from Section 3. It is worth recalling that in the PAC setting we do have efficient algorithms for learning linear threshold functions. Thus, this also yields a separation between PAC learning and mistake-bounded online learning.

It is also worth comparing the updates of the Perceptron algorithm to the algorithm we developed for learning GLMs. Note that we can consider sign : $\mathbb{R} \to \mathbb{R}$ as a function that is monotone; it is however not Lipschitz. If we used the same surrogate loss approach we would get update rules that are the same as the one used by Perceptron (up to constant factors); this corresponds to an online gradient descent approach. The margin condition allows us to use a Lipschitz approximation of the sign function as there is no data in the region around the boundary.

5 The Winnow Algorithm

We will now look at an online learning problem where the target function depends on a relatively small number of features, but the total number of available features is very large. The particular example we shall look at is the class of monotone disjunctions. Let $X = \{0,1\}^n$ be the instance space, for any subset $S \subseteq [n]$, then define the monotone disjunction,

$$f_S(\mathbf{x}) = \bigvee_{i \in S} x_i.$$

The class of monotone disjunctions is the set of all such f_S, $S \subseteq [n]$. For any monotone disjunction, f_S, it can be expressed as a linear threshold function,

$$f_S(\mathbf{x}) = \operatorname{sign}\left(\sum_{i \in S} x_i - \frac{1}{2}\right),$$

where we map $\{0,1\}$ to $\{-1,1\}$ in order to ensure the equivalence. This suggests that one can use the Perceptron algorithm for online learning monotone disjunctions. There are a couple of tweaks required to make this work. First, the way we have defined the Perceptron algorithm it only works for homogeneous halfspaces. This is relatively easy to handle. We instead consider the instance space to be X_{n+1} and let the last bit of all examples always be 1. Then, any threshold function of the form $\operatorname{sign}(\mathbf{w} \cdot \mathbf{x} + b)$ where $\mathbf{w} \in \mathbb{R}^n$, $\mathbf{x} \in X_n$ and $b \in \mathbb{R}$ can be written as $\operatorname{sign}((\mathbf{w}, b) \cdot (\mathbf{x}, 1))$, where $(\mathbf{w}, b) \in \mathbb{R}^{n+1}$ and $(\mathbf{x}, 1) \in X_{n+1}$.

Second, we need to look at target vectors having norm 1 in order to apply Theorem 9. For $S \subseteq [n]$, let $\mathbf{w}^\star_S \in \mathbb{R}^{n+1}$ be such that $\mathbf{w}^\star_i = c$ if $i \in S$ and $\mathbf{w}^\star_i = 0$ for $i \notin S$ and $i \leq n$, and let $\mathbf{w}^\star_{n+1} = -c/2$. By setting $c^2 = 1/(|S| + \frac{1}{4})$ we can ensure that $\|\mathbf{w}^\star\|_2 = 1$. Note that $f_S(\mathbf{x}) = \operatorname{sign}(\mathbf{w}^\star_S \cdot (\mathbf{x}, 1))$ for every $\mathbf{x} \in X_n$. Clearly, we have that $\|(\mathbf{x}, 1)\|_2^2 \leq n + 1$. And $y(\mathbf{w}^\star_S \cdot (\mathbf{x}, 1)) \geq c/2$. Thus, we can use $\gamma = c/2$ and $D = \sqrt{n+1}$, to get a mistake bound of $4(n+1)(|S| + \frac{1}{4})$. If we knew that $|S| \leq k$, applying Theorem 4 this would give us a PAC learning algorithm with a sample complexity $O((nk/\epsilon)\log(nk/\epsilon))$. However, using Occam's Razor, we would expect a sample complexity of $O((k \log n)/\epsilon + \log(1/\delta)/\epsilon)$.

We will study an online algorithm called *Winnow* which achieves a mistake bound of $O(k \log n)$ for this problem. This will essentially give us a near optimal sample complexity after applying Theorem 4. The online algorithm itself is of independent interest as it uses multiplicative weight updates which we shall study in greater detail later. This algorithm and its analysis appeared in the work of Littlestone (1988). The algorithm in the paper can be generalized to certain types of linear threshold functions which we won't consider here.

Winnow starts by assigning a weight of 1 for every input feature. It adjusts these weights every time it makes a mistake by either doubling the weight for some feature or setting it to 0. Once a weight is set to 0 it can never become non-zero again. The predictions are made using a weighted majority rule. The main result we will prove is that for online learning monotone disjunctions, the mistake bound of Winnow is $O(k \log n)$, where k is the number of variables in the target disjunction.

Theorem 12. *Let $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ be an infinite sequence with $y_i = f_S(\mathbf{x}_i)$ for every i where $|S| = k$. Then if Winnow (Alg. 4) is given this input in an online fashion, the number of mistakes made by Winnow is at most $2k \log_2 n + 2$.*

Proof. We will prove this theorem through a sequence of claims which are proved separately. Let P be the number of times the algorithm makes mistakes of the form where $y_t = 1$ and $\widehat{y}_t = 0$. These are promotion steps, as the weights increase for some of the features. Let E be

Algorithm 4 The Winnow Algorithm

```
**Input**: Sequence $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ provided online.
Set $\mathbf{w}_1 = (1, 1, \ldots, 1) \in \mathbb{R}^n$
**for** $t \leftarrow 1, 2, \ldots,$ **do**
    Receive $\mathbf{x}_t$
    $\widehat{y}_t = \mathbb{1}\left(\mathbf{x}_t \cdot \mathbf{w}_t \geq \frac{n}{2}\right) \in \{0, 1\}$
    Receive $y_t$
    **if** $\widehat{y}_t = 1$ and $y_t = 0$ **then**
        $w_{t+1,i} = 0$ for all $i$ such that $x_{t,i} = 1$.
    **else if** $\widehat{y}_t = 0$ and $y_t = 1$ **then**
        $w_{t+1,i} = 2w_{t,i}$ for all $i$ such that $x_{t,i} = 1$.
    **end if**
**end for**
```

the number of times the algorithm makes mistakes of the form of $y_t = 0$ and $\widehat{y}_t = 1$. These are elimination steps, as some weights are set to 0.

The total number of mistakes is $P + N$. Claim 15 shows that $P \leq k \log_2 n$ and Claim 14 shows that $E \leq P + 2$. Together these yield the required result. $\square$

Claim 13. *Under the conditions of Theorem 12, if $w_{t,i}$ is the weight of the i^{th} feature at time t for the Winnow Algorithm (Alg. 4), we have $w_{i,t} \leq n$.*

Proof. Suppose there is a mistake at time t. Clearly the weights only increase when $y_t = 1$ and $\widehat{y}_t = 0$. Only those indices i for which $x_{t,i} = 1$ can have their weights changed. The fact that $\widehat{y}_t = 0$ means that $w_{t,i} < n/2$ for each i such that $x_{t,i} = 1$. Then, it follows that $w_{t+1,i} \leq n$. $\square$

Claim 14. *Under the conditions of Theorem 12, if P is the number of mistakes when $\widehat{y}_t = 0$ and E is the number of mistakes when $\widehat{y}_t = 1$, we have $E \leq P + 2$.*

Proof. We consider how the quantity $\sum_i w_{t,i}$ varies with time. Note that this quantity is always non-negative and when $t = 1$, $\sum_i w_{1,i} = n$. Every mistake where $y_t = 1$ and $\widehat{y}_t = 0$ increases the sum of weights by at most $n/2$. This is because the weights are doubled for all i such that $x_{t,i} = 1$; however, the reason $\widehat{y}_t = 0$ was that $\sum_i w_{t,i} x_{t,i} < n/2$. So doubling these weights can't increase the total weight by more than $n/2$.

On the other hand, by essentialy the same logic, every mistake where $y_t = 0$ and $\widehat{y}_t = 1$ decreases the total weight by at least $n/2$.

Thus, we have for all t,

$$0 \leq \sum_i w_{t,i} \leq n + P \cdot \frac{n}{2} - E \cdot \frac{n}{2}.$$

The conclusion then follows. $\square$

Claim 15. *Under the conditions of Theorem 12, if P is the number of mistakes when $\widehat{y}_t = 0$, we have $P \leq k \log_2 n$.*

Proof. We look at each time a mistake of the type where $y_t = 1$ and $\widehat{y}_t = 0$ occurs. There must be some $i \in S$, where S is the set of literals in the target disjunction, such that $x_{t,i} = 1$. Thus $w_{t,i}$ will be doubled. Note that for any $i \in S$, $w_{t,i}$ can never be set to 0 and every mistake that is counted in P doubles the weight of at least one of the relevant variables. The fact that $|S| \leq k$ and Claim 13 finishes the proof. $\square$

References

Hans-Dieter Block. The perceptron: A model for brain functioning. i. *Reviews of Modern Physics*, 34(1):123, 1962.

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

Albert B Novikoff. On convergence proofs for perceptrons. Technical report, Stanford Research Institute, Menlo Park, CA, 1963.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.