Computational Learning Theory **Fall Semester, 2012**

Lecture 3: November 4

Lecturer: Yishay Mansour *Scribe: Tal Saiag and Daniel Labenski* [1]

3.1 The PAC Model

In this lecture we will talk about the PAC model. The PAC learning model is one of the important and famous learning model. PAC stands for *Probably Approximately Correct*, our goal is to learn a hypothesis from a hypothesis class such that in high confidence (probably) we will have a small error rate (approximately correct). We start the lecture with an intuitive example to explain the idea behind the PAC model and the differences between the PAC model and the Bayesian learning that we studied last week. Then, we continue to formal the PAC model, review Occam's Razor principle and give a few examples.

3.2 An intuitive example

Suppose we want to predict what is a 'typical person'. Our input is a sample of different descriptions of people based on their height and weight and their label: whether it is a typical person ('+') or not ('-'). Let H be our hypothesis class. In our example, H will be the set of all possible rectangles on a two-dimensional plane which are axis-aligned (not rotated). One example of hypothesis $h \in H$ can be: mark a person which denotes as (height, weight) to be a typical one ('+') if its description is in the range of:

$$1.60 \leq height \leq 1.90, 60 \leq weight \leq 90$$

Assuming the real target function is a rectangle R (This assumption will be referred later). Our goal is to find the best rectangle R' that approximates the real target rectangle R. In general, we will try to learn an accurate predictor which will optimize our performance.

This learning model is different than the Bayesian inference where we assumed that the underlying distribution has a specific form and our goal is to estimate this distribution. In PAC, we don't know the underlying probability of the samples. Here, the typical people distribution is unknown and finding the joint distribution of heights and weight is rather complex.

[1]Based on scribes written by Yoav Giora and Omer Meshar (April 20, 2002) and Ilan Cohen, Yaron Margalit, Alex Zhicharevich (October 2010)

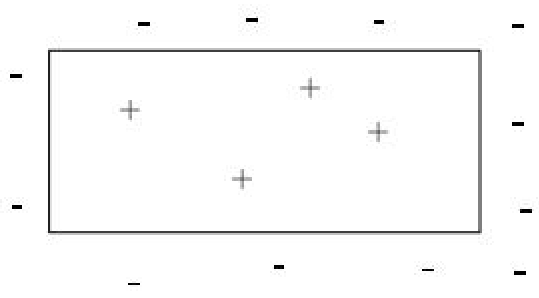

Figure 3.1: A rectangle with positive and negative examples

Generally, in the PAC model, we do not impose any assumptions on the underlying distribution of the data other than that such a distribution exists and the samples are independently and identically distributed (i.i.d) according to the same distribution. If the test samples were taken from different distribution than the training samples then we wouldn't have any reason to assume that we will be successful. This assumption gives us hope that what we learn, based on the training set, gives good results that are close to the real target function on that distribution.

The learning problem we described can be viewed as follows:

- Goal: Learn rectangle R.
- Input: Examples based on data set and their label: $\langle (x, y), +/- \rangle$.
- Output: R', a good approximation rectangle of the real target rectangle R. (An example of R', see Figure 3.1)

3.2.1 A Good Hypothesis

Our goal is to find a hypothesis that will have a small error rate, smaller than a parameter ε. Let $R \Delta R'$ be the error of R' in respect to the real target rectangle R. This can be defined by two separate areas: $(R - R') \cup (R' - R)$ (where $(R - R')$ are false negative and $(R' - R)$ are false positive) as shown in Figure 3.2.

Then, our goal can be defined as finding R' such that with probability of at least $1 - \delta$ (confidence):

$$\Pr[error] = \mathcal{D}(R \Delta R') \leq \varepsilon$$

Assuming that $R \in H$ is the real target function.

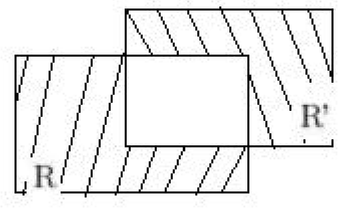

Figure 3.2: R and R' areas including two different error spaces. In our example, since $R' \subseteq R$ there exists only error of $(R - R')$.

3.2.2 Learning Strategy

Let $S = \{\langle(x_1, y_1), b_1\rangle, ..., \langle(x_m, y_m), b_m\rangle\}$ be the sample data. Intuitive, we would like a rectangle that will be *consistent*, i.e., no error on the sample data. This is possible since we assumed that there exists a rectangle that labels correctly the data (the target rectangle R). We can choose any rectangle from the minimal size (R_{min}) up to the maximize size (R_{max}) because they are all *consistent* with the sample data, as shown in Figure 3.3. We will show later, that it doesn't matter which one of these rectangles the algorithm returns.

The strategy A we will try is to request a "sufficiently large" number of m examples, and to choose the hypothesis rectangle R' that is the tightest fit to the positive examples ($R' = R_{min}$). That can be done by using four positive points indicating the left-most,right-most,down-most,up-most points in the sample data that are positive.

3.2.3 Sample Size

In this section, we try to find what is a "sufficiently large" number of examples that is needed to learn a good hypothesis. For that, we will fix our accuracy and confidence parameters (ε, δ), and the strategy A. We will show that for any distribution D, we can assert sample size m that with high confidence (that is, probability at least $1 - \delta$), the output rectangle R' from strategy A (i.e., the tightest fit rectangle) has an error of at most ε.

We will be aware of the fact that $R' \subseteq R$ and we construct $T'_1, .., T'_4$ areas that will indicate the error area, as defined in Figure 3.4. That is, $R\Delta R' = \cup T'_i$.

If the distribution D sych that each $T'_1, ..., T'_4$ is $D(T'_i) \leq \frac{1}{4}\varepsilon$, then the error rate of R' is at most: $\Pr[error] = \mathcal{D}(R\Delta R') = \mathcal{D}(\cup T'_i) \leq \sum_{i=1}^{4} \frac{\varepsilon}{4} = \varepsilon$. This approach is problematic because it depends on the strip T'_i that is constructed by the output R' from the strategy A after seeing the sample. Because the sample has already been seen, the event $D(T'_i) < \frac{\varepsilon}{4}$ lost its randomness and in order to talk about it (its probability), we need to create an event which do not depend on the sample and use it in the proof.

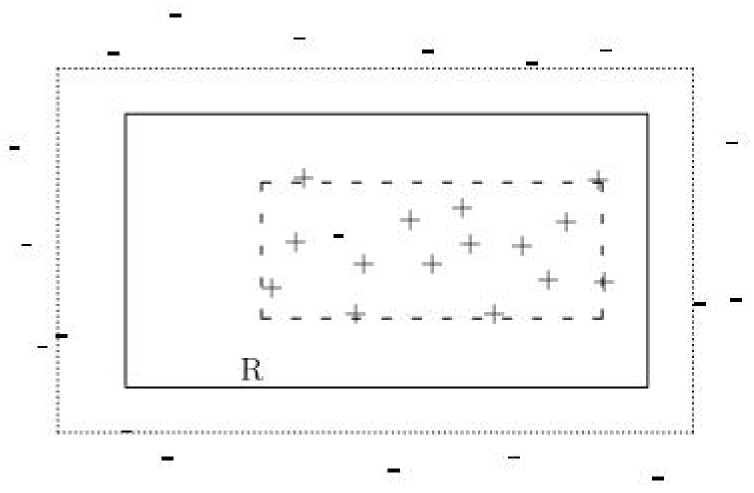

Figure 3.3: The smallest R_{min} (dashed line) and largest R_{max} (dotted line) rectangles border the rectangles which are consistent with the examples. The area between them represents the error region.

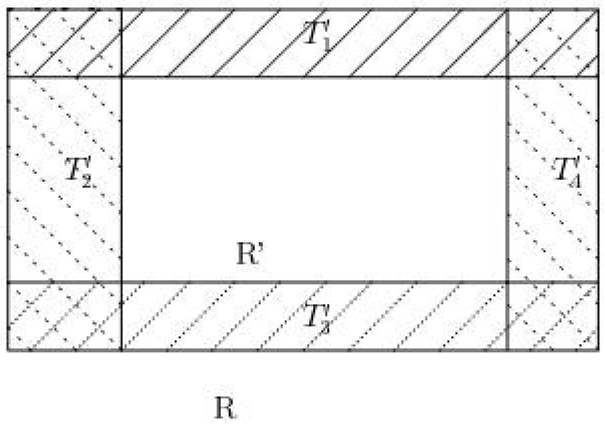

Figure 3.4: Breaking up the error into four rectangular strips $T_1', ..., T_4'$

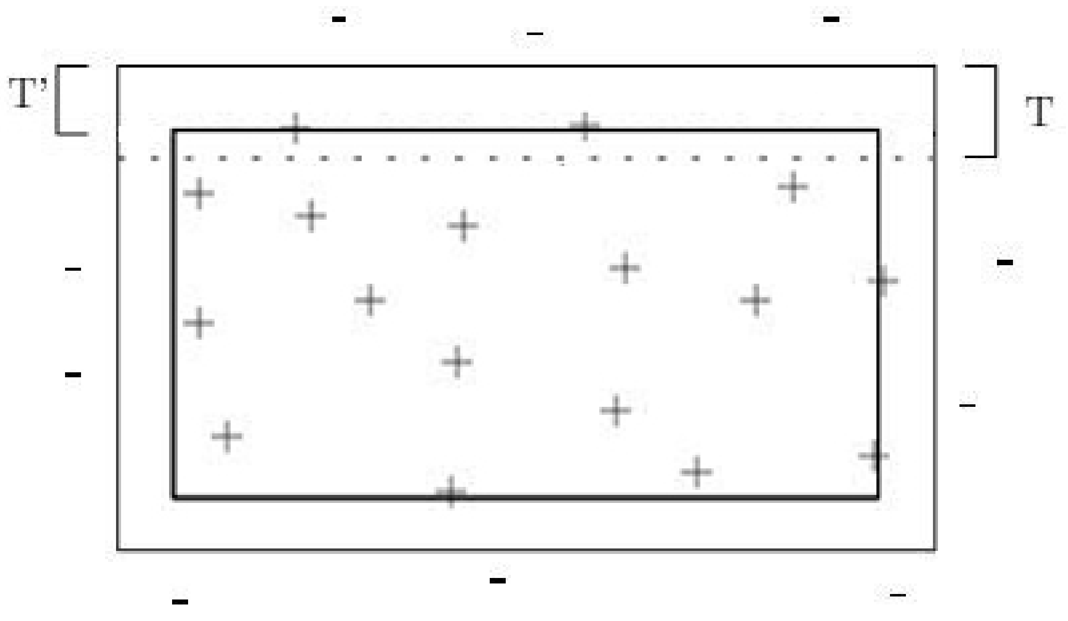

Figure 3.5: Adjusting strip size to have a weight of at most ε according to the real target function R. The strips $T'_i \subseteq T_i$ surrounding the inner rectangle $R' = R_{min}$ apart from the outer rectangle R.

A better construction will depend only on the real target rectangle R. We will construct strips $T_1, .., T_4$ such that $\forall i\ D(T_i) \leq \frac{1}{4}\varepsilon$ (Figure 3.5). From the construction we can see that T_i is independent of the sample and of R'. Note that we cannot find the T_i using the sample but we can be sure that such T_i exists.

We want to have $\forall i\ T'_i \subseteq T_i$. If that is the case, we obtained our requirement since

$$\Pr[error] = \mathcal{D}(R\Delta R') = \mathcal{D}(\cup T'_i) \leq \mathcal{D}(\cup T_i) \leq \varepsilon$$

From the construction, if there is at least one sampled point that resides in T_i it implies that $T'_i \subseteq T_i$. This is true, since the rectangle from strategy A must include all sampled positive points in R. To achieve that we can ask: what is the probability of having a bad event, that is, what is the probability that we didn't receive points from the sample data that are located on the constructive strips, i.e., T_i. Formally,

$$\Pr[\text{error} > \varepsilon] \leq \Pr[\exists i = 1..4\ \forall (x,y) \in S, (x,y) \notin T_i]$$

By definition of T_i,

$$\Pr[x \notin T_i] = 1 - \frac{\varepsilon}{4}$$

Since our sample data is i.i.d from distribution D:

$$\Pr[\forall(x,y) \in S, (x,y) \notin T_1] = (1-\frac{\varepsilon}{4})^m$$

The same analysis holds on each T_i strip. Hence, we get that on the entire region the error would not exceed the sum of probabilities for each of the strips. That is,

$$\Pr[\text{error} > \varepsilon] \leq 4(1-\frac{\varepsilon}{4})^m$$

Using the inequality $(1-x) \leq e^{-x}$, we obtain:

$$\Pr[\text{error} > \varepsilon] \leq 4(1-\frac{\varepsilon}{4})^m \leq 4e^{-\frac{\varepsilon}{4}m} < \delta$$

That is, if we want to have accuracy ε and confidence of at least $1-\delta$, we have to choose the sample size m to satisfy:

$$4e^{-\frac{\varepsilon}{4}m} < \delta <=> m > \frac{4}{\varepsilon} ln \frac{4}{\delta}$$

For this strategy A, and for any small (ε, δ) we like, we got the sample size $m(\varepsilon, \delta)$ that is needed for having a good learner.

3.2.4 Remarks

1. The analysis holds for any fixed probability distribution $\mathcal{D}$, we only required that the sample points are i.i.d from distribution $\mathcal{D}$ to obtain our bound.

2. The lower bound of the sample size $m(\varepsilon, \delta)$ behaves as we might expect. One might want to have better accuracy by decreasing ε or greater confidence by decreasing δ — our algorithm requires more examples to meet those requirements. There is a stronger dependence in ε.

3. The parameter ε gives the degree of accuracy that we want to achieve. It determines what is a good hypothesis for achieving a good approximation in respect to the target function. In our example, the accuracy determines which of our hypothesis rectangles are good enough in respect to the real rectangle target. We pay attention that the accuracy does not depend on the data distribution.

4. The parameter δ gives the degree of confidence on having a good learner. Meaning, how sure are we that we've reached that level of accuracy. This can depend on, how well the given sample data reflects the true distribution. Again, it does not depend on the data distribution.

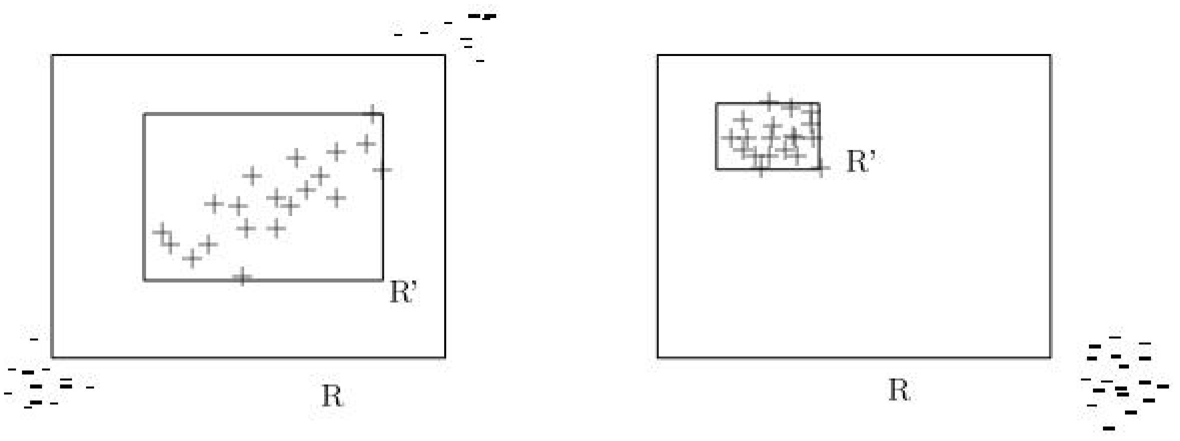

Figure 3.6: Two cases depending on the sample size

5. An example of learning, we might have cases as shown in Figure 3.6, where the distribution $\mathcal{D}$ gives large weights to particular regions of the plain, creating a distorted image of the rectangle. In any case, under those conditions, since the learner is tested on the same distribution $\mathcal{D}$, and this distribution has small error between R and R', the rectangle $\mathcal{R}'$ will be a good hypothesis (in respect to ε, δ).

6. The strategy A that we defined is efficient: In computational view, the only need is to search for the max and min points that defines our tightest-fit rectangle. In sample data size view, the number of examples that is required for achieving accuracy ε with confidence $1 - \delta$ is polynomial in $\frac{1}{\varepsilon}$ and $ln\frac{1}{\delta}$.

7. In this example, in contrast to the Bayesian approach, we are not trying to model $\mathcal{D}$ or to guess which rectangle is more likely (a-prior). We have separated the distribution $\mathcal{D}$ from the target function (rectangle R), and directly try to predict hypothesis for the target function.

3.3 A formal Presentation of the PAC Model

3.3.1 Preliminaries

- The goal of the learning algorithm is to learn an unknown target function concept out of a known hypothesis class, for example to learn one particular rectangle out of the set of all possible rectangles. Unlike the Bayesian approach, no prior knowledge is needed.

- Learning occurs in a stochastic setting. Examples from the target rectangle are drawn randomly according to a fixed, unknown probability distribution and are i.i.d.
- We assume that the sample and test data are generated by the same unknown probability distribution.
- The solution should be efficient: the sample size required for obtaining small (ε) error with high ($1-\delta$) confidence is a function of $\frac{1}{\varepsilon}$ and $\ln\frac{1}{\delta}$. Also, we can process the sample within a time polynomial in the sample size.

3.3.2 Definition of the PAC Model

Let the set X be the *instance space* or *example space.* Let $\mathcal{D}$ be any fixed probability distribution over the instance space X. A *concept class* over X is a set

$$\mathcal{C} \subseteq \big\{c \mid c : X \to \{0,1\}\big\}.$$

Let c_t be the *target concept*, $c_t \in C$, and h a *hypothesis*, h may belong to a different concept class than $\mathcal{C}$, the *hypothesis* $\mathcal{H}$. We will define the *error* of h with respect to the distribution $\mathcal{D}$ and the target concept c_t as follows:

$$error(h) = \Pr_{\mathcal{D}}[h(x) \neq c_t(x)] = D(h \Delta c_t(x))$$

Let $EX(c_t, \mathcal{D})$ be a procedure (we will sometimes call it an *oracle*) that runs in a unit time, and on each call returns a labeled example $\langle x, c_t(x)\rangle$, where x is drawn independently from $\mathcal{D}$. In the PAC model, the oracle is the *only* source of examples for the learning algorithm.

Definition Let $\mathcal{C}$ and H be concept classes over X. We say that C is *PAC learnable by* H if there exists an algorithm A with the following property: for every concept $c_t \in \mathcal{C}$, for every distribution $\mathcal{D}$ on X, and for all $0 < \varepsilon, \delta < \frac{1}{2}$, if A is given access to $EX(c_t, \mathcal{D})$ and inputs ε and δ, then with probability at least $1 - \delta$, A outputs a hypothesis concept $h \in H$ such that:
If $c_t \in H$ (Realizable case), then h is satisfying

$$error(h) \leq \varepsilon$$

If $c_t \notin H$ (Unrealizable), then h is satisfying

$$error(h) \leq \varepsilon + \min_{h' \in H} error(h')$$

We say that $\mathcal{C}$ is *efficiently PAC learnable*, if A runs in time polynomial in $\frac{1}{\varepsilon}$, $\ln\frac{1}{\delta}$, n and m, where n is the size of the input and m the size of the target function (for example, the number of bits that are needed to characterize it). Implicit, we mean that it is "easier" to learn a "simpler" target function. (this idea is further explained later in the lecture).

3.4 Finite Hypothesis Class

In this section, we will show how to learn a good hypothesis from a finite hypothesis class $\mathcal{H}$. The proof "trick" will use the idea of analyzing a bad hypothesis: we define a hypothesis h to be ***ε*-*Bad*** if $error(h) > \varepsilon$.

3.4.1 The Case $c_t \in \mathcal{H}$

Generally, after having processed $m(\varepsilon, \delta)$ instances, we will request from our algorithm A to find an h which is *consistent* with the sample S, i.e., classifies all the instances the same as the target concept (formally, $\forall x \in S,\ h(x) = c_t(x)$). The algorithm will succeed to learn if h is not ε-Bad, that is, if $error(h) \leq \varepsilon$. We note that at least one *consistent* hypothesis exists because $c_t \in \mathcal{H}$.

Now, we will try to bound the probability of algorithm A to return h that is ε-Bad. We note that if the probability for a ε-bad is smaller that δ then every consistent hypothesis is good. From our defined algorithm A, having an ε-Bad hypothesis is bound by the probability that there exists a concept h which is both consistent and ε-*Bad*. First, we will look at a fixed h ε-Bad:

$$\Pr[h \;\; is \;\; \varepsilon\text{-Bad} \;\&\; h(x_i) = c_t(x_i) \; for \; 1 \leq i \leq m] \leq (1-\varepsilon)^m < e^{-\varepsilon m}$$

First inequality derived directly from the fact that h is ε-bad, hence, $\Pr[h(x) = c_t(x)] \leq 1-\varepsilon$ and the fact that the examples are sampled i.i.d from distribution $\mathcal{D}$.

Now, we bound the failure probability of algorithm A:
Pr [A returns an ε-Bad hypothesis] $\leq$

$$\begin{aligned}
&\Pr[\exists h \in \varepsilon\text{-Bad} \;\&\; h(x_i) = c_t(x_i) \; for \; 1 \leq i \leq m] \\
\leq\;& \sum_{h \in \varepsilon\text{-Bad}} \Pr[h(x_i) = c_t(x_i) \; for \; 1 \leq i \leq m] \\
\leq\;& |\{h \;:\; h \; is \; \varepsilon\text{-Bad} \;\&\; h \in H\}|(1-\varepsilon)^m \\
\leq\;& |\mathcal{H}|(1-\varepsilon)^m < |\mathcal{H}|e^{-\varepsilon m}.
\end{aligned}$$

The first inequality comes directly from the union bound. The second inequality is true by the previous analysis for an ε-Bad hypothesis, and the third inequality is derived from the fact:

$$|\{h \ : \ h\ is\ \varepsilon\text{-Bad} \ \&\ h \in H\}| \subseteq H,$$

In order to satisfy the condition for PAC learning, we bound the failure probability by δ:

$$|\mathcal{H}|e^{-\varepsilon m} \leq \delta,$$

which implies a bound on the sample size,

$$m \geq \frac{1}{\varepsilon} \ln \frac{|\mathcal{H}|}{\delta}\,.$$

We note that for a finite class of hypothesis, every consistent algorithm A, results a hypothesis $h \in H$ that in probability $1-\delta$ is *not* ε-bad. As expected it tells us that the larger the hypothesis class, the larger the sample we require. However it is not clear how to implement it efficiently, and in general it will not be efficient. Also, it only works for finite classes and even in this lecture we saw an example that doesn't apply for this analysis — the class of all rectangles.

3.4.2 The Case $c_t \notin \mathcal{H}$

In this case, we need to relax our goal, since a small error hypothesis might not even exist (in H). As we defined before, our goal is to learn according to the "best" hypothesis in the hypothesis class (i.e. with the minimal error). Let h^* be the hypothesis with the minimal error, i.e., for every $h \in \mathcal{H}$:

$$0 < error(h^*) \leq error(h)\,.$$

Let β to be $\beta = error(h^*) = \min_{h \in H} error(h)$. Our goal is to find hypothesis h that will achieve:

$$error(h) \leq \beta + \varepsilon$$

Note that this is in fact a generalization of our previous system (in which $\beta = 0$).

After having processed $m(\varepsilon, \delta)$ instances, we define $\widehat{error}(h)$ to be the empirical error of h on sample S. Formally:

$$\widehat{error}(h) = \frac{1}{m} \sum_{i=1}^{m} I(h(x_i) \neq c_t(x_i)),$$

where I is the indicator function. We also need to correct the algorithm from the realizable case, since now it may be impossible to choose a consistent h. Algorithm A will return a hypothesis $\bar{h}$ that will have the minimal empirical error. That is, $\bar{h} = \underset{h \in H}{\operatorname{argmin}}\ \widehat{error}(h)$ (If there exists more than one hypothesis, the algorithm will pick one of them). This algorithm is also called ERM, empirical Risk Minimization.

We will now bound the sample size required for obtaining a good hypothesis from algorithm A. We want to choose a sample size m such that the difference between the true and the estimated error is small for every hypothesis. That is, with probability at least $1 - \delta$:

$$\forall h \in H, |\widehat{error}(h) - error(h)| \leq \frac{\varepsilon}{2}.$$

If we have a good error estimation, we can derive easily that using the hypothesis $\bar{h}$ from algorithm A, we obtain:

$$error(\bar{h}) \leq \widehat{error}(\bar{h}) + \frac{\varepsilon}{2} \leq \widehat{error}(h^*) + \frac{\varepsilon}{2} \leq error(h^*) + \varepsilon$$

The first and third inequality holds since the difference between the true and the estimated error is small (up to $\frac{\varepsilon}{2}$). The second inequality holds since the empirical error of the hypothesis obtained from algorithm A is the minimal one.

To conclude our lower bound for the sample size we need to find the probability of failure in estimating $error(h)$, we will use Chernoff bounds that we introduced in the first lecture:

$$\Pr[\ |\widehat{error}(h) - error(h)| \geq \frac{\varepsilon}{2}] \leq 2e^{-2(\frac{\varepsilon}{2})^2 m}$$

We can use Chernoff bounds since the training sample of size m is sampled i.i.d from the same distribution $\mathcal{D}$. We can get a lower bound on the sample size by requiring the probability of failure over $\mathcal{H}$ to be smaller than δ:

$$\Pr[\exists h \in \mathcal{H}\ :\ |\widehat{error}(h) - error(h)| \geq \frac{\varepsilon}{2}] \leq 2|\mathcal{H}|e^{-2(\frac{\varepsilon}{2})^2 m} \leq \delta\,,$$

hence

$$m(\varepsilon, \delta) \geq \frac{2}{\varepsilon^2} \ln \frac{2|\mathcal{H}|}{\delta}.$$

We have thus found a lower bound on the sample size for PAC learning when $c_t \notin \mathcal{H}$. Note that the sample size depends on $\frac{1}{\varepsilon^2}$ instead of $\frac{1}{\varepsilon}$ when we had $c_t \in \mathcal{H}$. This results from the difference between requiring a single counter-example (in the case that $error(h) = 0$), to the need of having the average over many examples be significantly different to disqualify a function, by estimating the error on the data.

3.4.3 Example - Learning Boolean Disjunctions

To demonstrate the PAC model we will look at the example of learning boolean disjunction functions. The problem is defined as follows: Given a set of boolean variables $T = \{x_1, ..., x_n\}$ and a set of literals $L = x_1, \bar{x}_1, \ldots x_n, \bar{x}_n$. we need to learn an Or function over the literals, for example: $x_1 \vee \bar{x}_3 \vee x_5$. C will be the set of all possible disjunctions. We can notice that$|C| = 3^n$, since for each variable x_i the target disjunction c_t may contain x_i, $\bar{x}_i$, or neither.
We will use H=C.

ELIM algorithm for learning boolean disjunctions

We can notice that by receiving a negative example we can eliminate all literals that give 1 to the variables in the sample. For example if there was a sample 001 (assume 0 was the value of x_1 and x_2 and 1 was the value of x_3) which was negative we can eliminate the literals $\bar{x}_1, \bar{x}_2, x_3$ because if one of them was in the target disjunction it would give a positive value. The algorithm uses that fact and eliminates the inconsistent literals. We also notice that the algorithm classifies the positive samples correct because the target disjunction is also a sub-goal $c_t \subseteq L_{\text{final}}$.
So the algorithm initializes a set $L = x_1, \bar{x}_1, \ldots x_n, \bar{x}_n$ and for each negative sample Z it assigns $L = L - \{\bar{z}_i | z_i \in Z\}$

From previous sections we can see that the algorithm learns when the sample size m is at least:

$$m > \frac{1}{\varepsilon} \ln \frac{|\mathcal{H}|}{\delta} = \frac{1}{\varepsilon} \ln \frac{3^n}{\delta} = \frac{n \ln 3}{\varepsilon} + \frac{1}{\varepsilon} \ln \frac{1}{\delta}$$

3.4.4 Example - Learning parity

Consider the concept class of xor of n variables. More formally: A set of variables : $x_1, ..., x_n$. The concepts class is the set of all XOR functions over the variables. Example for a function in C is : $x_1 \oplus x_7 \oplus x_9$.
We notice that $|C| = 2^n$ and allowing negative will not increase it since $\bar{x_1} = x_1 \oplus 1$, and we need only to add the literal 1. The algorithm will regard the samples as a linear equation problem (in Z_2 field) and then solve it using Gaus elimination method. Each sample consists of a bit vector (satisfying the value of the variable) and the classification c_t of this example. This creates a linear equation, for example: $(01101, 1)$ represents the equation: $0 * a_1 + 1 * a_2 + 1 * a_3 + 0 * a_4 + 1 * a_5 = 1 (mod 2)$ (deduced directly from the properties of xor). Each a_i is a binary indicator of x_i in the formula. A solution to all the examples exists (since we want to learn a xor function). Thus, solving the linear equation problem will

produce a solution which is consistent with the whole sample The needed sample size m is

$$m > \frac{1}{\varepsilon} \ln \frac{|\mathcal{H}|}{\delta} = \frac{n}{\varepsilon} \ln 2 + \frac{1}{\varepsilon} \ln \frac{1}{\delta} .$$

3.5 Occam Razor

"Entities should not be multiplied unnecessarily"

(William of Occam, c. 1320)

The above quote is better known as Occam's Razor or the principle of Ontological Parsimony. Over the centuries, people from different fields have given it different interpretations. One interpretation used by experimental scientists is: given two explanations of the data, the simpler explanation is preferable. This principle is consistent with the goal of machine learning: to discover the simplest hypothesis that is consistent with the sample data. However, the question still remains: why should one assume that the simplest hypothesis based on past examples will perform well on future examples. After all, the real value of a hypothesis is in predicting the examples that it has not yet seen. It will be shown that, under very general assumptions, Occam's Algorithm produces hypotheses that with high probability will be predictive of future observations. As a consequence, when hypotheses of minimum or near minimum complexity can be produced in time which is polynomial in the size of the sample data, it leads to a polynomial PAC-learning algorithm. Here, a simple hypothesis, means a short coded hypothesis. We saw that we can achieve a polynomial PAC-learning algorithm, for a finite class of hypotheses, H, if we choose the number of input examples m, to be:

$$m \geq \frac{1}{\epsilon} \ln \frac{|H|}{\delta}$$

We can allow H to grow with m, providing that it grows slower than m.

Definition: An (α, β) *Occam-algorithm* for a functions class C, using a hypotheses class H, is an algorithm which has two parameters: a constant $\alpha \geq 0$ and a compression factor $0 \leq \beta < 1$. Given a sample S of size m, which is labeled according to c_t, i.e., $\forall\ i \in \{1, \cdots, m\}\ \ c_t(x_i) = \sigma_i$, the algorithm outputs an hypothesis $h \in H$, such that:

1. The hypothesis h is consistent with the sample, i.e., $\forall\ \ i \in \{1, \cdots, m\}\ \ h(x_i) = \sigma_i$.

2. The size of h is at most $n^\alpha m^\beta$, where n is the size of c_t, m is the sample size, and α and β are the Occam algorithm's parameters.

3.5.1 Occam Algorithm and PAC

We now show that the existence of an Occam-algorithm for $\mathcal{C}$ implies polynomial PAC learnability.

Theorem 3.1 *Let A be an (α, β) Occam Algorithm for C, using H.Then A is PAC with*

$$m \geq (\frac{n^\alpha}{\epsilon} \ln 2)^{1/(1-\beta)} + \frac{2}{\varepsilon} \ln \frac{1}{\delta}$$

Remark: Notice, that when $\beta \to 1$, then $m \to \infty$. This makes sense, since the closer β is to 1, the poorer the compression we get, and hence we have less "information" about the function.

Proof: Fix m and n. Algorithm A returns an hypothesis h s.t. $size(h) \leq n^\alpha m^\beta$. The number of hypothesis of this size is $2^{n^\alpha m^\beta}$.
We have shown that for PAC we need $m \geq \frac{1}{\varepsilon} \ln \frac{|H|}{\delta}$. Since $|H| \leq 2^{n^\alpha m^\beta}$, we get $m \geq \frac{1}{\varepsilon} n^\alpha m^\beta \ln 2 + \frac{1}{\varepsilon} \ln \frac{1}{\delta}$. The following sequence of inequalities derrives the bound
$m \geq \max\{\frac{2}{\varepsilon} n^\alpha m^\beta \ln 2, \frac{2}{\varepsilon} \ln \frac{1}{\delta}\} \Rightarrow m \geq \frac{2}{\varepsilon} n^\alpha m^\beta \ln 2 \Rightarrow m \geq (\frac{2}{\varepsilon} n^\alpha \ln 2)^{\frac{1}{1-\beta}}$

□

3.5.2 Example: Boolean OR with k Literals

As before we want to learn boolean disjunctions for n variables, but we know that the expression has at most k literals, i.e., the hypothesis C size is $n^k (\ll 3^n)$. We will show Occam algorithm that creates hypothesis h with size $O(k \ln n \ln m)$ we will use the greedy algorithm for set cover.

```
Algorithm 1: Greedy Algorithm for Set Cover
Input: $S_1, S_2 ... S_t \subseteq V = \{1 \dots m\}$
Output: $S_{i_1}, ..., S_{i_k}, \Rightarrow \bigcup S_{i_j} = V$
SetCoverGreedy($V$)
(1)  $S \leftarrow \emptyset, j \leftarrow 0, V_0 \leftarrow V$
(2)  while $V_j \neq \emptyset$
(3)      Choose $S_i = \underset{S_r}{\operatorname{argmax}}\{|S_r \cap V_j|\}$
(4)      $S \leftarrow S \cup \{i\}$
(5)      $V_{j+1} \leftarrow V_j - S_i$
(6)      $j \leftarrow j + 1$
(7)  return $S$
```

Algorithm analysis

Let S_{opt} coverage for V with k sets, then the greedy algorithm return a cover with at most $1 + k \ln m$. The cover S_{opt} is also cover for any V_j , and has just k sets, hence

$$\forall j : \exists t \in S_{opt} \Rightarrow |V_j \cap S_t| \geq \frac{|V_j|}{k}$$

We upper bound how fast the sets V_j shrinks:
$|V_{j+1}| \leq |V_j| - \frac{|V_j|}{k} = (1 - \frac{1}{k})V_j$ (We choose the maximum set)
$|V_{j+1}| \leq (1 - \frac{1}{k})^{(j+1)}|V_0| \leq e^{-\frac{j+1}{k}}m,$

when the bound is less the 1 then $V_j = \emptyset$. Therefore, after $1 + k \ln m$ steps the algorithm will stop.

Algorithm Boolean OR with k Literals

Input S examples Run algorithm ELIM the output is L.
Notice $|L| = O(n)$, and we need $m = o(n)$
So our objective is to choose small group of literals that satisfy S (the positive subset).
Reduction to set cover:
$T = \{x :< x, + >\in S\}$ (All the positive examples)
$T_i = \{x : x \in T, x_i \in x\}$ (all the examples that literal x_i satisfy)
We know that there exists a group of k literals that satisfy the examples, so if we run greedy set cover, it guaranteed that we will get $O(k \ln m)$ literals that satisfy all the examples.
There exists $2n$ literals that can be encoded with $O(\log 2n)$, hence
$size(h) \simeq k \ln n \ln m$, since $\ln m \leq m^\beta$ and $\ln n \leq n^\alpha$ for any $\alpha, \beta \geq 0$, so we got (α, β) Occam algorithm where α, β are arbitrary small.
A tighter bound for the sample data size (m) can be computed directly:

$$m > \frac{1}{\varepsilon}(k \ln m \ln m) + \frac{1}{\varepsilon} \ln \frac{1}{\delta}$$

$$m > \frac{4}{\varepsilon} k \ln^2 n + \frac{1}{\varepsilon} \ln \frac{1}{\delta}$$