Machine Learning	Spring semester, 2002

Lecture 3: April 20, 2002

Lecturer: Yishay Mansour *Scribe: Yoav Giora and Omer Meshar* [1]

3.1 The PAC Model

This lecture presents the PAC model, which is the most important learning model. The PAC model is used in many theoretical studies in machine learning. PAC stands for *Probably Approximately Correct*, the goal of the model is to learn a certain concept (a member of a known concept class) so that with a high degree of confidence the prediction error will be small.

3.2 An Example - a Rectangle Learning Game

Suppose we want to learn the concept of "a typical person", where a person is represented by a data point $(height, weight)$ and its label by '+' for a "typical person" and '-' otherwise. A "typical person" is represented by the axis-aligned rectangle R: for example 170 cm $\leq height \leq$ 180 cm and 70 kg $\leq weight \leq$ 90 kg. We shall call R the *target rectangle*. The *goal* of the algorithm is to find a rectangle that approximates the target rectangle, so that with high probability, the error of the prediction will be small.

How can we find a statistical model for this problem ? Even when the probability of meeting each person is the same, the data will not be uniformly distributed because:

- The heights and weights are not uniformly distributed.
- The two quantities are highly correlated.

Finding the common distribution of heights and weights can be very difficult. Instead, we merely assume that the data is sampled using a distribution $\mathcal{D}$. We also assume that the samples are *independent*. We don't assume anything about $\mathcal{D}$, besides the fact that it remains fixed through the whole process.

Model Based approach (i.e. Bayesian learning) tries to model the distribution in order to find an optimal decision rule. PAC model bypasses the need of modelling the input distribution, because its goal is *probabilistic* and defined by means of the input distribution.

[1] Based on scribes written by Yuval Krymolowski (March 19, 1997)

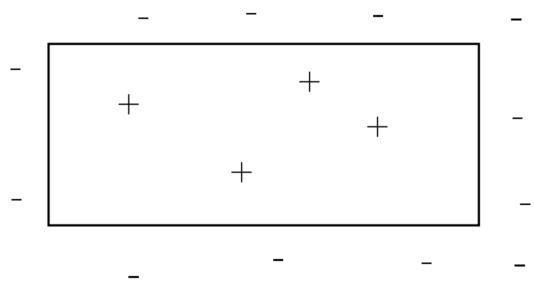


Figure 3.1: A rectangle with positive and negative examples

This learning problem can be thought of as a one-player game:

- Goal: Learn an unknown axis-aligned rectangle R, that is, a rectangle in the Euclidean plane $\Re^2$ whose sides are parallel with the coordinate axes.
- Input: Examples consist of data points and their label: $\langle (x, y), +/- \rangle$.
- Output: Using the minimal sample size, find efficiently a hypothesis rectangle R' that will be a good approximation of R.

Figure 3.1 shows the unknown rectangular region R along with a sample of positive and negative examples.

3.2.1 What is a Good Hypothesis ?

The goodness of a hypothesis is defined using the *input distribution* $\mathcal{D}$. The error of R' is defined as the probability, according to $\mathcal{D}$, of a wrong labeling. We require the error to be smaller than a specified ε. Let $R \Delta R' = (R - R') \cup (R' - R)$, then the condition for a good hypothesis is:

$$\Pr[err] = \mathcal{D}(R \Delta R') < \varepsilon \,.$$

3.2.2 Learning Strategy

The rectangle we choose will be *consistent*, that is, it has to contain all positive examples and no negative ones. This is possible due to the assumption that there exists a rectangle that labels correctly the persons *(the target rectangle R)*. We can choose any rectangle "between" the smallest and the largest one. It turns out that there is no advantage in taking any specific consistent rectangle, therefore, for our error analysis we take the smallest one.

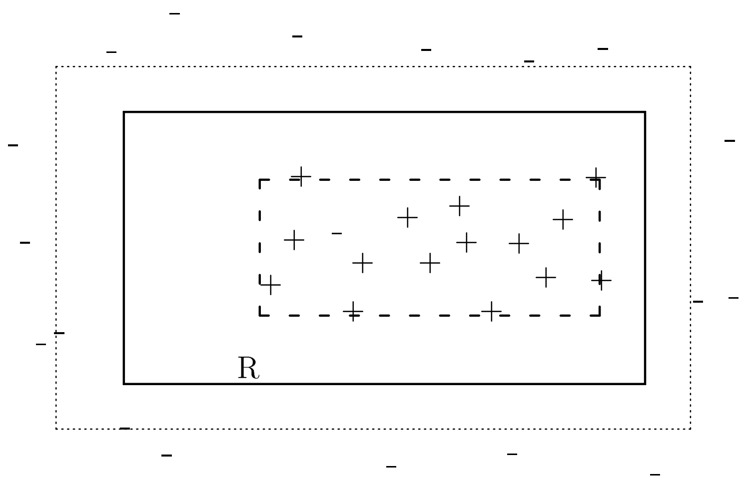


Figure 3.2: The smallest (dashed line) and largest (dotted line) rectangles border the rectangles which are consistent with the examples. The area between them represents the error region.

A simple and efficient strategy is to request a "sufficiently large" number of m random examples, then choose as the hypothesis the axis-aligned rectangle R which gives the *tightest fit* to the positive examples (that is, the rectangle with the smallest area that includes all the positive examples and none of the negative examples). If no positive examples are drawn, then $R' = \emptyset$. Figure 3.2 shows the tightest-fit rectangle defined by the sample shown in Figure 3.1.

3.2.3 The Sample Size

Now that we fixed our strategy, the remaining important parameter we have to determine is the number of sample points needed for a good approximation. We will now show that for any target rectangle R and any distribution $\mathcal{D}$, and for any small values ε and δ ($0 < \varepsilon, \delta \leq \frac{1}{2}$), for a suitably chosen value of the sample size m we can assert that with a high probability, i.e. probability at least $1 - \delta$, the tightest-fit rectangle has error at most ε with respect to R and $\mathcal{D}$.

The tightest-fit rectangle R' is always contained in the target rectangle R

$$R' \subseteq R\,.$$

We can express the difference $R - R'$ as the union of four rectangular strips, as can be seen in Figure 3.3. There is some overlap between these four rectangular strips at the corners, so if we count the overlap regions twice, we only increase our error estimate

$$R \Delta R' = R - R' = \cup T_i'\,.$$

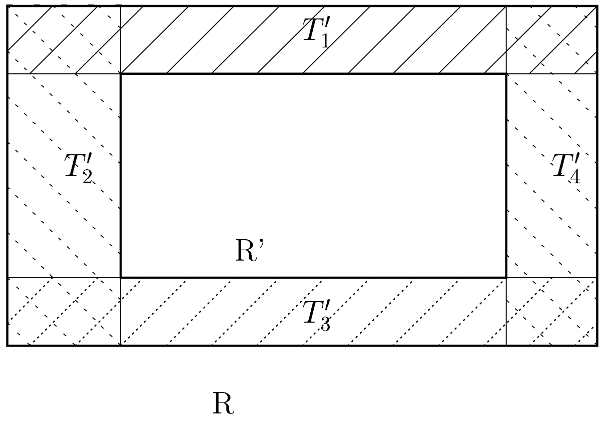


Figure 3.3: Breaking up the error into four rectangular strips

If the weight under $\mathcal{D}$ of each strip is at most $\frac{\varepsilon}{4}$

$$\mathcal{D}(T_i') \leq \frac{\varepsilon}{4},$$

then the error of R' (defined by the weight that the probability distribution gives to $R\Delta R'$) is at most

$$\mathcal{D}(R\Delta R') = \mathcal{D}(\cup T_i') \leq \sum \mathcal{D}(T_i') = 4(\frac{\varepsilon}{4}) = \varepsilon$$

which implies that $\mathcal{R}'$ is a good hypothesis.

We shall analyze the case of the upper strip T_1', (the calculation for the other strips is analogous). Define T_1 to be the upper strip of $\mathcal{R}$ for which $\mathcal{D}(T_1) = \frac{\varepsilon}{4}$. (Note that T_1 is independent of the sample, and is defined only by $\mathcal{R}$.) If $T_1' \subseteq T_1$ then $\mathcal{D}(T_1') \leq \frac{\varepsilon}{4}$, and $\mathcal{R}'$ is a good hypothesis. If at least one sampled point resides in T_1, it implies that $T_1' \subseteq T_1$, because $\mathcal{R}'$ must contain all sampled (positive) points in $\mathcal{R}$, and therefor the lower bound of T_1' is higher than the lower bound of T_1.

By definition of T_1, the probability that a single draw from the distribution $\mathcal{D}$ misses the region T_1 is exactly

$$Prob[x \notin T_1] = 1 - \frac{\varepsilon}{4}.$$

Therefore the probability that m *independent* draws from $\mathcal{D}$ all miss the region T_1 is exactly

$$Prob[x_1, x_2, \ldots, x_m \notin T_1] = (1 - \frac{\varepsilon}{4})^m.$$

The same analysis holds for the other three rectangular regions of $R - R'$: T_2', T_3' and T_4'. Therefore, for the entire region our error would not exceed the sum of probabilities for each of the strips. Hence,

$$Prob[\mathcal{D}(R\Delta R') \geq \varepsilon] \leq 4(1 - \frac{\varepsilon}{4})^m.$$

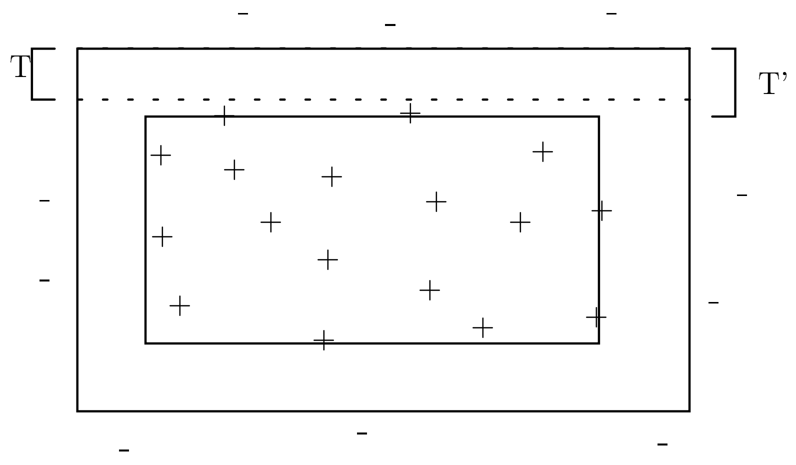


Figure 3.4: Adjusting strip size to have a weight of at most ε

If we want the error probability to be less than δ, we have to choose m to satisfy:

$$4(1 - \frac{\varepsilon}{4})^m \leq \delta\,.$$

This guarantees with probability $1 - \delta$ over the m random examples, that the weight of the error region $R - R'$ will be bounded by ε, as claimed. Using the inequality $(1 - x) \leq e^{-x}$ we get:

$$4(1 - \frac{\varepsilon}{4})^m \leq 4e^{-\frac{\varepsilon}{4}m} \leq \delta$$

also satisfies the previous condition. Taking natural logarithms of both sides along with some simple arithmetic gives:

$$m \geq \frac{4}{\varepsilon} ln \frac{4}{\delta}.$$

This is the sample size we must take. It must have a *lower* bound since the bigger the sample the better the estimate. This sample size does not depend on the distribution function $\mathcal{D}$, and this is the great power of the algorithm.

Thus, if we take a sample of at least $(\frac{4}{\varepsilon} ln \frac{4}{\delta})$ examples to form the hypothesis rectangle R', we can assert that with probability at least $1 - \delta$, R' will misclassify a new point (drawn according to the same distribution from which the sample was chosen) with probability at most ε.

3.2.4 Remarks

The analysis holds for any fixed probability distribution $\mathcal{D}$. We only needed the independence of successive data points to obtain our bound.

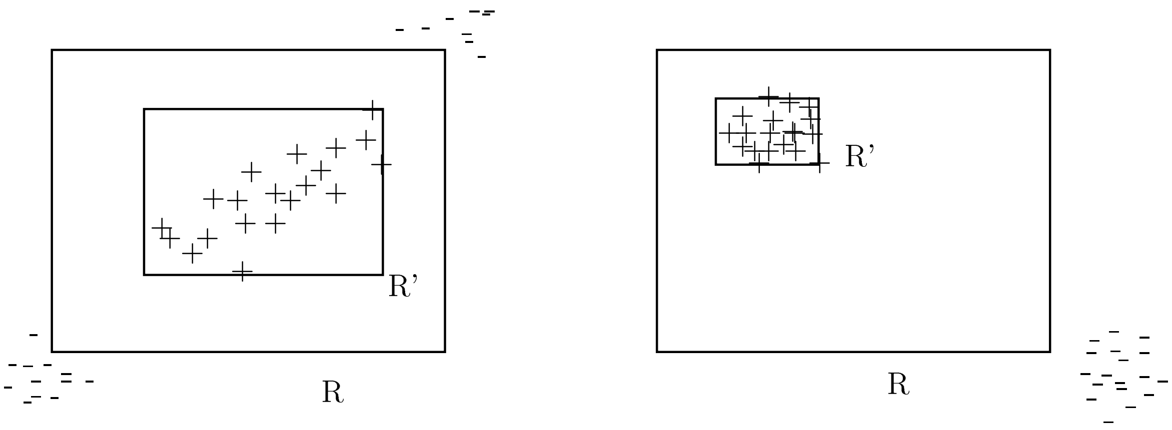


Figure 3.5: Two cases depending on the sample size

The sample size bound behaves as we might expect, in that we increase our demands on the hypothesis rectangle — that is, as we ask for greater accuracy by decreasing ε or greater confidence by decreasing δ — our algorithm requires more examples to meet those demands.

The parameter ε gives the degree of accuracy that we want to achieve. It determines which assumptions we shall classify as good; it does not depend on the distribution function. It tells us by how much the learner's assumption and the target function can differ.

The parameter δ gives the degree of confidence that we require from our sample, that is, the degree of randomness of the sample. Again, it does not depend on the distribution function.

In fact, we might have cases as shown in Figure 3.5, where the distribution function gives large weights to particular regions of the plain, creating a distorted image of the rectangle. In any case, under these conditions, since the learner is tested only through the distribution $\mathcal{D}$, and this distribution has small weight between R and R', the rectangle $\mathcal{R}'$ will be a good hypothesis.

The algorithm we have analyzed is *efficient*: the required sample size is a slowly growing function of $\frac{1}{\varepsilon}$ and $\frac{1}{\delta}$ (linear and logarithmic, respectively), and once the sample is given, the computation of the tightest-fit hypothesis can be carried out rapidly.

It seems we have done better than the Bayesian approach, since we haven't been trying to model $\mathcal{D}$ or to guess which rectangle is more likely (prior). We have separated the distribution $\mathcal{D}$ from the classifying function (rectangle R), and produced a learning model for this function.

3.3 A formal Presentation of the PAC Model

3.3.1 Preliminaries

- The goal of the learning game is to learn an unknown target concept out of a known set, for example to learn one particular rectangle out of the set of all possible rectangles. Unlike Bayes inference, no prior knowledge is needed.
- Learning occurs in a probabilistic setting. Examples from the target rectangle are drawn randomly in the plane according to a fixed, unknown probability distribution and are mutually independent.
- The hypothesis of the learner is evaluated relative to the *same* probability distribution.
- The solution should be efficient: the sample size required for obtaining small (ε) error with high ($1-\delta$) confidence is linear in $\frac{1}{\varepsilon}$ and $\ln\frac{1}{\delta}$, and 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 *input space* (for instance X could be the whole plane R^2). Let $\mathcal{D}$ be any fixed probability distribution over the instance space X (which is unknown). A *concept class* over X is a set

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

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

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

Let $EX(c_t, \mathcal{D})$ be a procedure (we will sometimes call it an *oracle*) that runs in unit time, and on each call return 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.

3.3.3 PAC Learning 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 (from class H) satisfying $error(h) \leq \varepsilon$. (The probability is over $EX(c_t, \mathcal{D})$ and the algorithm A).

3.3.4 Efficiently PAC Learnable Algorithms

If A runs in time polynomial in $\frac{1}{\varepsilon}$, $\ln \frac{1}{\delta}$, n and m when 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) we say that $\mathcal{C}$ is *efficiently PAC learnable*. We mean that it is "easier" to learn a "simpler" target function. The input parameters are:

ε : the *error parameter* — the level of accuracy that is demanded for a hypothesis to be a good approximation

δ : the *confidence parameter* — how sure we are that we've reached that level of accuracy. δ is a bound on probability that the algorithm will fail on finding a good hypothesis.

The output is a hypothesis $h \in \mathcal{H}$.

3.4 Finite Hypothesis Class

In a finite hypothesis class $\mathcal{H}$ we define a hypothesis h to be ***ε**-Bad* if $error(h) > \varepsilon$. After having processed $m(\varepsilon, \delta)$ instances, the algorithm will find an h which is consistent with the sample (i.e., classifies all the instances the same as the target concept). The algorithm will succeed if h is not ε-Bad, that is, if $error(h) < \varepsilon$.

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

Consider an ε-Bad hypothesis h. We analyze the probability that h will survive as a consistent hypothesis after learning from a sample of size m (at least one consistent hypothesis exists because $c_t \in \mathcal{H}$).

Since the examples are independent and $\varepsilon < error(h)$,

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

We bound the failure probability by the probability that there exist a concept h which is both consistent and ε-*Bad*:

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

$$\leq \quad |\mathcal{H}|(1-\varepsilon)^m < |\mathcal{H}|e^{-\varepsilon m}.$$

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

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

which implies a bound on the sample size,

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

This algorithm is general and achieves the desired results. As expected it tells us that the larger the concept 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 infinite class — the class of all rectangles.

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

In this case we need to redefine our goal, since a small error hypothesis might not even exist (in H). Let h^* be the hypothesis with the minimal error, i.e., for every $h \in \mathcal{H}$:

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

Let's assume that: $error(h^*) = \beta$. Note that h^* is not required to be consistent with the examples. We say that $error(h^*)$ is the *inherent error* of the problem. There will be an additional error component when trying to estimate h^*. We therefore measure the goodness of a hypothesis h by its difference from the error of h^*. Thus the goal is to find h such that:

$$error(h) - error(h^*) < \varepsilon.$$

or:

$$error(h) \leq error(h^*) + \varepsilon = \beta + \varepsilon.$$

Note that this is in fact a generalization of our previous system (in which $\beta = 0$).
We also need to correct the algorithm, not only the goal, since now it may be impossible to choose a consistent h. Define $abs\_error(h)$ to be the absorbed error of h on the data sample, and choose h with minimal $abs\_error(h)$.
We will now bound the sample size required for obtaining a suitable hypothesis. Consider a hypothesis h, we will denote by $\widehat{error}(h)$ the average error on the data. We want to show that for a large enough sample, we output a good hypothesis with high probability.

We want to choose a sample size m such that the difference between the true and the estimated error is small. That is the probability of failure in estimating $abs\_error(h)$, using Chernoff bounds we get:

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

Therefore, we can get a lower bound on the sample size by requiring the probability of failure over $\mathcal{H}$ to be smaller than δ:

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

hence

$$m \geq \frac{4}{\varepsilon^2} \ln \frac{|\mathcal{H}|}{\delta} .$$

If we have good error estimates, i.e., for every $h \in \mathcal{H}$ we have $|\widehat{error}(h) - abs\_error(h)| \leq \varepsilon/2$, then

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

which yields

$$error(h) \leq error(h^*) + \varepsilon .$$

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 needing a single counter-example (in which $abs\_error(h) = 0$), to needing to average over many examples to disqualify a function, by estimating the error on the data.

3.4.3 Example - Learning Boolean Disjunctions

Consider the concept class of disjunctions of n variables. more formally:
Variables: $x_1, ..., x_n$.
Literals : $x_1, \bar{x}_1, ..., x_n, \bar{x}_n$.
We need to learn an Or function over the literals. For example: $x_1 \vee \bar{x}_2 \vee x_5$.

For each variable x_i the target disjunction c_t may contain x_i, $\bar{x}_i$, or neither, therefore there are 3^n possible disjunctions, i.e., $|\mathcal{C}| = 3^n$. We use $\mathcal{H} = \mathcal{C}$, when both are finite classes.
We shall now introduce an algorithm for learning Or functions (ELIM).
During the learning process we maintain a list of literals which may appear in c_t. Initially

$$L = x_1, \bar{x}_1, \ldots x_n, \bar{x}_n .$$

Each example is represented by variable values and the result of the c_t evaluation. When an example x is encountered with $c_t(x) = 0$, all the literals whose values are positive are deleted from the list L, thereby keeping L including all the literals consistent with the data (examples which encounter $c_t(x) = 1$ are erased from the date). The reason L is consistent with the data is that we begun with a full literal list and removed only those which caused conflicts.
For instance let $n = 5$,and the goal function $f = x_1 \vee \bar{x}_3 \vee x_5$. The given example is: $(01100, 0)$. Thus $\bar{x}_1, x_2, x_3, \bar{x}_4, \bar{x}_5$ could be deleted from L since neither of them can be a part of our goal function.

The results of Section **4.4.1** show that we can PAC learn boolean disjunctions of n variables, using the learning method outlined here with a sample of size m satisfying

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

3.4.4 Example: Learning parity

Consider the concept class of xor of n variables. more formally:
Variables: $x_1, \ldots\ldots, x_n$.
Function example for $n = 9$: $x_1 \otimes x_7 \otimes x_9$.
The number of functions is 2^n since each variable could either appear in the function or not. Note that allowing literals to appear (allowing negations) will only increase the size of the class to $2 * 2^n$. We take the function which includes the same variables with no negations at all. We add $\otimes 1$ to the end of this function, in case the number of negations is odd, and in case of an even number it remains the same.

The algorithm will regard the samples as a linear equation problem (in Z_2 field) and then solve it using Gaus 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: $0 * a_1 + 1 * a_2 + 1 * a_3 + 0 * a_4 + 1 * a_5 = 1(mod2)$ (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. Section **4.4.1** results imply that we can PAC learn a xor function with a sample size m satisfying:

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

3.5 Infinite Hypothesis Class

We have already seen an example of PAC learning from an infinite concept class. The theoretical approach in this case will be briefly presented later in this lecture and in greater detail in a future talk. Here is another example:

Let X be the interval $[0, 1]$.

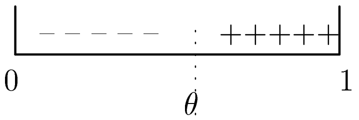
Let $\mathcal{H}$ be a concept class over X:

$$\mathcal{H} = \{c_\theta \mid 0 \leq \theta \leq 1\}$$

and

$$c_\theta = \begin{cases} 1 & x \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

Schematically a concept looks as follows,



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

Proof I

Let's choose m examples and define:

$$min\{x \mid c_t(x) = 1\} \;=\; max,$$

and

$$max\{x \mid c_t(x) = 0\} \;=\; min.$$

We shall choose a value $\theta \in [min, max]$ and return c_θ. It is enough to show that with probability $1 - \delta$ the weight of the interval $[min, max]$ under $\mathcal{D}$ is at most ε. This is sufficient because errors will accure only on this interval. Let's analyze the probability that m samples will not be taken from the interval $[min, max]$. Suppose

$$\mathcal{D}([0_{max}, 1_{min}]) \geq \varepsilon$$

The probability that we did not sample in $[min, max]$ is $(1-\varepsilon)^m$, then using the inequality $(1-x) \leq e^{-x}$ we get:

$$(1-\varepsilon)^m \leq e^{-\varepsilon m} \leq \delta$$

so it is enough that:

$$m \geq \frac{1}{\varepsilon} ln \frac{1}{\delta}.$$

This proof is wrong - why?
Note that min and max where determined only by the sample. Therefor, there may not be any point of the sample between them. Consequently we can't make a probabilistic assumption on the sample.

Proof II
We shall Define parameters which do not depend on the sample. Let's define max' and min' to be the points for whom are $\varepsilon/2$ close to θ.

$max' : D[\theta, max'] = \varepsilon/2$.
$min' : D[min', \theta] = \varepsilon/2$.
Note that θ is the target we try to learn.
The goal is to show that with high probability $(1-\delta)$: $max \in [\theta, max']$, and $min \in [min', \theta]$. In this case any value between $[min, max]$ is good and could be returned by the algorithm, since now $D[min, max] < \varepsilon$, and D is the real distribution (not just the sample).

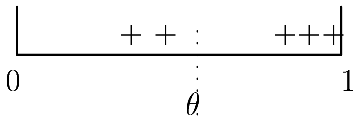
We have that

$$Prob[x_1, x_2, \ldots, x_m \notin [min', \theta]] = (1 - \frac{\varepsilon}{2})^m < e^{\frac{-m\varepsilon}{2}}$$

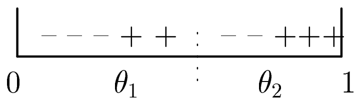
The failure probability for $[\theta, max']$ is derived similarly. Finally we get: $2e^{\frac{-m\varepsilon}{2}} < \delta$. Hence: $m > \frac{2}{\varepsilon} ln \frac{2}{\delta}$.
Remember that min', max' where chosen with no dependence on the sample.

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

This case corresponds to noisy data or a more complicated labeling function than those represented by $\{c_\theta\}$.



In this case we can not find some c^*_θ which we would like to approximate because functions with a similar error may use values of θ very far from each other, as θ_1 and θ_2 in the figure below.



For a sample size m, there are $m+1$ possible hypotheses according to the intervals between samples (the values of θ within an interval are equivalent in the sense that they produce the same learning error.) We can thus choose the hypothesis which will minimize the error on the examples.

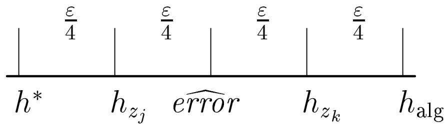
For a given distribution $\mathcal{D}$, let $0 = z_0 < z_1 \cdots < z_k = 1$ such that $\mathcal{D}([z_i, z_{i+1}]) = \frac{\varepsilon}{4}$ (this is called a $\frac{\varepsilon}{4}$ net). Let h_{alg} be the output of the learning algorithm, h^* an optimal hypothesis, and h_{z_i} the hypothesis with $\theta = z_i$. Our definition of the $\{z_i\}$ implies that there exist z_j and z_k such that:

$$error(h_{z_j}) - error(h^*) \leq \frac{\varepsilon}{4},$$

and

$$|error(h_{z_k}) - error(h_{\mathrm{alg}})| \leq \frac{\varepsilon}{4},$$

We will show that when the sample size is large enough, for every z_i, with a high probability, $|error(h_{z_i}) - obs\_error(h_{z_i})| \leq \frac{\varepsilon}{4}$. That implies $error(h_{\mathrm{alg}}) - error(h^*) \leq \varepsilon$ as the figure demonstrates:



Let

$$q_i = Pr_{\mathcal{D}}[c_t(x) \neq h_{z_i}(x)],$$

and $\hat{q}_i$ be the estimate of q_i calculated from the sample. Using Chernoff bounds we get

$$Pr[\ |\hat{q}_i - q_i| \geq \frac{\varepsilon}{4}] \leq e^{-(\frac{\varepsilon}{4})^2 m}\,.$$

Since the number of z_i's is $\frac{4}{\varepsilon}$, the probability of an estimation error larger than $\frac{\varepsilon}{4}$ may be bounded by

$$Pr[\ |error(h_{z_i}) - obs\_error(h_{z_i})| \geq \frac{\varepsilon}{4}] \leq \frac{4}{\varepsilon} \cdot e^{-(\frac{\varepsilon}{4})^2 m} < \delta\,.$$

Therefore, we have to choose a sample whose size is:

$$m \geq \frac{16}{\varepsilon^2}[\ln\frac{4}{\varepsilon} + \ln\frac{1}{\delta}]\,.$$

3.5.3 General ε-Net approach

An ε-net is a set $\mathcal{G} \subset \mathcal{H}$ such that for every $h \in \mathcal{H}$ there exists $g \in \mathcal{G}$ such that

$$Prob_{\mathcal{D}}[g(x) \neq h(x)] \leq \varepsilon\,.$$

The algorithm will first find an $\frac{\varepsilon}{4}$-net for $\mathcal{H}$ which will be represented by the set $\{h_{z_i}\}$, and returns the optimal h_i of the net. In general, the sample size required for a confidence of $1-\delta$ when an $\frac{\varepsilon}{4}$ of $\mathcal{G}$ is known is

$$m \geq \frac{16}{\varepsilon^2} \ln\frac{|\mathcal{G}|}{\delta}\,.$$

The derivation of that bound is similar to that done in the preceding section.

3.5.4 The VC Dimension

The question of determining the sample size required for achieving a given confidence when the hypothesis class is infinite is generally addressed by the *VC dimension* (Vapnik-Chervonenkis). The VC dimension of H is the maximal number of points, d, which can be labeled in all the 2^d possible combinations by concepts from $\mathcal{H}$. The VC dimension is a substitute for $\ln|\mathcal{H}|$ in the expressions for sample size bounds.

The topic of VC dimension will be covered in a future talk.

3.6 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 algorithm's parameters.

3.6.1 Occam Algorithm and Compression

An Occam algorithm gives a way to compress the sample. The n bits σ_i can be replaced by the $n^\alpha m^\beta$ bits that represent h. Thus, suppose A wants to transmit σ_i to B, and both share the x_i, it is enough for A to transmit h, becasue B would be able to compute the values of σ_i by evaluating $h(x_i) = \sigma_i$. Therefore we have the same information in a more compact form.

3.6.2 Occam Algorithm and PAC Learnability

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. Let D be the distribution according to which the samples, X, are drawn. Let $c_t \in C$ be the target concept, and n be its size. For any $0 \leq \epsilon, \delta \leq 1$, if A gets as an input a sample, S, which has m examples drawn from $EX(c_t, D)$, where*

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

then the probability that the output, $h \in H$, satisfies : $error(h) \leq \epsilon$, is at least $1 - \delta$.

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: Since A returns a hypothesis h, satisfying: $size(h) \leq n^\alpha m^\beta$, and assuming the hypotheses under consideration are given by binary strings, the size of the hypothesis space is no greater than $2^{n^\alpha m^\beta}$. We showed that if:

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

then any consistent $h \in H$ satisfies $error(h) \leq \epsilon$ with probability at least $1 - \delta$.

Since $|H| \leq 2^{n^\alpha m^\beta}$ we get the same conclusion for:

$$m \geq \frac{1}{\epsilon} \ln \frac{2^{n^\alpha m^\beta}}{\delta}$$

We can conclude that

$$m \geq \frac{n^\alpha m^\beta}{\epsilon} \ln \frac{2}{\delta}$$

We need to solve for m,

$$m^{\beta-1} \geq \frac{n^\alpha}{\epsilon} \ln \frac{2}{\delta}$$

therefore,

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

which completes the proof. ■

3.7 Example - Boolean Disjunctions with k Literals

We will now assume that our target concept is a disjunction of k literals. We will learn this class of functions with a sample size polynomial in k and $\log n$, where n is the total number of variables (for fixed ϵ and δ).

We will show an OCCAM algorithm for this class, and show that the number of examples needed for it is $O(k \log n \ln m)$, when m is the size of the sample.

A main tool in our algorithm is a reduction of the problem to Set-Cover, and finding an approximate set cover.

3.7.1 Set-Cover

The problem: Given a collection $S_1, \ldots, S_t$ of subsets of $U = \{1, \ldots, m\}$ find if there exists k subsets $S_{i_1}, \ldots, S_{i_k}$, which covers the set U, i.e. $\bigcup S_{ij} = U$.

Although the problem is NP-hard there is an approximation algorithm that finds a cover with at most $k \ln n$ subsets (where k is the optimal). This is done by the following GREEDY algorithm:

1. $U_0 \leftarrow U$, $j \leftarrow 0$, $S \leftarrow \phi$
2. while $U_j \neq \phi$
   (a) Choose a group S_i so that $|U_j \bigcap S_i|$ is maximal.
   (b) Add S_i to the cover: $S \leftarrow S \bigcup \{i\}$

(c) $U_{j+1} \leftarrow U_j \setminus S_i$.

(d) $j \leftarrow j + 1$

Analysis: Clearly, at the end S is a cover. Suppose the optimal cover is $S_{opt} = S_{i_1}, \cdots, S_{i_k}$. Then, S_{opt} is in particular cover for U_j. Since the optimal cover, S_{opt}, consists of only k sets, there exists a set in S_{opt} that covers at least U_j/k elements.

In other words, for every U_j there exists $S_{i_l} \in S_{opt}$ such that:

$$|U_j \bigcap S_{i_l}| \geq \frac{|U_j|}{k}$$

The greedy algorithm chooses at each step, the S_i, such that $|U_j \bigcap S_i|$ is maximum. Therefore the S_i that GREEDY chooses has,

$$|U_j \bigcap S_i| \geq \frac{|U_j|}{k}$$

Since $U_{j+1} \leftarrow U_j \setminus S_i$, then

$$|U_{j+1}| \leq |U_j| - \frac{|U_j|}{k}.$$

Therefore,

$$|U_{j+1}| \leq (1 - \frac{1}{k})|U_j|,$$

and hence,

$$|U_{j+1}| \leq (1 - \frac{1}{k})^{j+1}|U_0| = (1 - \frac{1}{k})^{j+1}m.$$

Therefore, after $j = k \ln m + 1$ steps we get $|U_j| < 1$ and the algorithm GREEDY stops. The number of iteration of the algorithm GREEDY is exactly the number of sets it has in the cover it builds. Therefore we have a cover with at most $k \ln m + 1$ sets.

3.7.2 Occam Algorithm

We can now define the Occam algorithm for this class. Given a sample S of size m it first uses the ELIMINATION algorithm on the literals. This algorithm as we analyzed gives a hypothesis h which is consistent with all the examples. Also recall that h consists of a set of literals (denoted LIT) that always contains the literals of the target function. The problem is, that there can still be $\theta(n)$ literals in LIT.

$$LIT = \{l : l \textit{ is a literal, consistent with the sample } S\}$$

Note that if we take from h some of its literals (no matter which) this does not affect its consistency with the negative examples (we just make an hypothesis negative in more places).

Our goal is to take as few literals as possible in a way that the consistency with the positive examples in the sample, denoted T, will not be affected as well.

$$T = \{x :< x, + >\in S\}$$

For every literal $z \in LIT$, let T_z be the set of positive examples that satisfy z.

$$T_z = \{x : x \in T, x \ satisfies \ z\}$$

Note that generating these sets from the sample S is easy.

We know that there exists a cover of T with k sets T_{z_i} - those that correspond to the literals of the target function $c_t = z_{i_1} \vee \cdots \vee z_{i_k}$. Therefore, we can apply GREEDY to get $k \log m$ sets that cover all T.

Assume that the output of GREEDY is $T_{z_{j_1}} \cdots T_{z_{j_l}}$, where $l = k \ln m$. The learning algorithm returns the hypothesis $h' = z_{j_1} \vee \cdots \vee z_{j_l}$. The hypothesis h' is consistent with the sample and has size of $(k \ln m) \log n$ bits, since there are $k \ln m$ literals, and each literal can be encoded using $\log n$ bits to describe it.

Since : $size(c_t) = k \log n$, and $\forall \beta > 0 \ \ln m < m^\beta$ (for m which is large enough):

$$size(h') = k \ln m \log n \leq m^\beta k \log n = m^\beta size(c_t)$$

Hence we have an Occam algorithm.

This implies, using the general theorem about Occam algorithms, that we can PAC learn this class with a sample of size $(\frac{k \log n}{\epsilon} \ln \frac{2}{\delta})^{1+\beta}$, where $\beta > 0$ can be arbitrary small.

3.8 Bibliographic Notes

The PAC model was introduced by Valiant [1]. The presentation of this class follows closely the book of Kearns and Vazirani [2].

1. Leslie G. Valiant, "A theory of the Learnable", Communication of the ACM, pp. 1134-1142, Nov. 1984.

2. Michael Kearns and Umesh Vazirani, "An introduction to computational Learning Theory", 1994.