COMS 6253: Advanced Computational Learning Theory **Spring 2012**

PAC-learning, Occam Algortihm and Compression

Lecturer: Rocco Servedio *Scribe: Mengqi Zong*

1 Introduction

The distribution-independent model of concept learning (PAC-learning, for "probably approximately correct learning") was introduced by Valiant and has been widely used to investigate the phenomenon of learning from examples.

In COMS 6253, we spent most of our time studying PAC-learning model. It's worthy for us to spend some time to take a closer look at this model.

This lecture will talk about the relationship between PAC-learning model and Occam algorithm. Later, we will show that for many classes, PAC-learnability is equivalent to data compression.

2 PAC-learning and Occam algorithm

2.1 Notation and Definitions

We describe a context for representing concepts over X. We define a class of representations to be a four-tuple $\mathbf{R} = (R, \Gamma, c, \Sigma)$. Σ and Γ are sets of characters. Strings composed of characters in Σ are used to describe elements of X, and strings of characters in Γ are used to describe concepts. $R \subseteq \Gamma^*$ is the set of strings that are concept descriptions or representations into concepts over Σ. R may be thought of as a collection of names of concepts, and for any $r \in R$, $c(r) \subseteq \Sigma^*$ is the concepts named by r.

We let $R^{[s]}$ denote the set $\{r \in R : |r| \leq s\}$; that is, the set of all representations from R of length at most s. If $\mathbf{R} = (R, \Gamma, c, \Sigma)$ is a class of representations, $r \in R$, and D is a probability distribution on Σ^*, then $EXAMPLE(D, r)$ is an oracle that, when called, randomly chooses an $x \in \Sigma^*$ according to distribution D and returns the pair $(x, r(x))$.

A randomized algorithm is an algorithm that behaves like a deterministic one with the additional property that, at one or more steps during its execution, the algorithm can flip a fair two-sided coin and use the result o the coin flip in its ensuing computation.

Definition 1. *The representation class* $\mathbf{R} = (R, \Gamma, c, \Sigma)$ *is PAC-learnable if there exists a (possibly randomized) algorithm L and a polynomial* p_L *such that for all* $s, n \geq 1$*, for all* ϵ *and* δ*,* $0 < \epsilon, \delta < 1$*, for all* $r \in R^{[s]}$*, and for all probability distributions* $\mathcal{D}$ *on* $\Sigma^{[n]}$*, if L is given as input the parameters* s, ϵ*, and* δ*, and may access the oracle EXAMPLE(*$\mathcal{D}$*, r), then L halts in time* $p_L(n, s, 1/\epsilon, 1/\delta)$ *and, with probability at least* $1 - \delta$*, outputs a representation* $r' \in R$ *such that* $\mathcal{D}(r' \oplus r) \leq \epsilon$*. Such an algorithm L is a polynomial-time learning algorithm for* $\mathbf{R}$*.*

Definition 2. *A randomized polynomial-time (length-based) Occam algorithm for a class of representations* $\mathbf{R} = (R, \Gamma, c, \Sigma)$ *is a (possibly randomized) algorithm O such that there exists a constant* $\alpha < 1$ *and a polynomial* p_O*, and such that for all* $m, n, s \geq 1$ *and* $r \in R^{[s]}$*, if O is given as input any sample* $M \subseteq S_{m,n,r}$ *and* $1/\gamma$*, and, with probability at least* $1 - \gamma$*, outputs a representation* $r' \in R$ *that is consistent with M, and such that* $|r'| \leq p_O(n, s, 1/\gamma)m^\alpha$*.*

2.2 Example: Learning Boolean conjunctions

In the book "An Introduction to Computational Learning Theory", learning Boolean conjunctions has been given as an example to compare PAC-learning and Occam algorithm.

For PAC-learning, the regular approach is to use inequality and union bounds to calculate a probability of m examples satisfy the needs. And the number of examples from PAC-learning is

$$m_{PAC} \geq (2n/\epsilon)(\ln(2n) + \ln(1/\delta))$$

As for Occam's algorithm, the approach is simply to take a look at the representation class H_n of Boolean conjunctions and see what we can get there. The number of examples from Occam' algorithm is

$$m_{Occam} \geq \frac{1}{\epsilon}\log\frac{1}{\delta} + n/\epsilon$$

Note that Occam's algorithm is an improvement by a logarithmic factor over the bound m_{PAC}. But the improvement doesn't happen always. For this example, the reason why Occam's algorithm gives a tighter bound is probably due to the union bounds used in the PAC-learning.

As we can see, Occam's algorithm is a naive approach for learning. And on most cases, the bound given by Occam's algorithm is a general complexity bound for PAC-learning.

2.3 The sufficient condition

We now show that the existence of an Occam algorithm for a class of concepts is a sufficient condition for the PAC-learnability of that class.

Theorem 3. *Let* $\mathbf{R} = (R, \Gamma, c, \Sigma)$ *be a class of representations with* γ *finite. If there exists a randomized polynomial-time (length-based) Occam algorithm for* $\mathbf{R}$*, then* $\mathbf{R}$ *is PAC-learnable.*

This theorem shows that the existence of an Occam algorithm for a class of concepts is a sufficient condition for the PAC-learnability of that class.

However, it was left as an **open problem** whether PAC-learnability is equivalent to the existence of Occam algorithms; i.e. whether the existence of an Occam algorithm is also a necessary condition for PAC-learnability.

2.4 Exception list

Definition 4. *A class* $\mathbf{R} = (R, \Gamma, c, \Sigma)$ *is polynomially closed under exception lists if there exists an algorithm EXLIST and a polynomial* p_{EX} *such that for all* $n \geq 1$*, on input of any* $r \in R$ *and any finite set* $E \subset \Sigma^{[n]}$*, EXLIST halts in time* $p_{EX}(n, |r|, |E|)$ *and outputs a representation* $EXLIST(r, E) = r_E \in R$ *such that* $c(r_E) = c(r) \oplus E$*. Note that the polynomial running time of EXLIST implies that* $|r_E| \leq p_1(n, |r|, \log|E|) + p_2(n, \log|r|, \log|E|)|E|$ *is satisfied, then we say that* $\mathbf{R}$ *is strongly polynomially closed under exception lists.*

Clearly, any representation class that is strongly polynomially closed is also polynomially closed. The definition of polynomial closure is above is easily understood - it asserts that the representation r_E that incorporates exceptions E into the representation r has size at most polynomially larger than the size of r and the total size of E, the latter of which is at most $n|E|$.

2.5 Results for finite representation alphabets

We consider the case in which the alphabet Γ (over which the representations of concepts are described) is finite. This typically occurs when concepts are defined over discrete domains (e.g. Boolean formulas, automata, etc.).

Theorem 5. *If* $\mathbf{R} = (R, \Gamma, c, \Sigma)$ *is strongly polynomially closed under exception lists and* $\mathbf{R}$ *is PAC-learnable, then there exists a randomized polynomial-time (length-based) Occam algorithm for* $\mathbf{R}$*.*

Corollary 6. *Let* Γ *be a finite alphabet. If* $\mathbf{R} = (R, \Gamma, c, \Sigma)$ *is strongly polynomially closed under exception lists, then* $\mathbf{R}$ *is PAC-learnable if and only if there exists a randomized polynomial-time (length-based) Occam algorithm for* $\mathbf{R}$*.*

3 PAC-learning and Compression

3.1 Learning versus Prediction

There are two common variants on the standard definition of PAC-learning. Under these alternative definitions, the hypothesis output by the learning is not required to be of the same form as the target concept description.

The notion of learning one representation class $\mathbf{R} = (R, \Gamma, c, \Sigma)$ in terms of another representation class $\mathbf{R}' = (R', \Gamma', c', \Sigma')$ was introduced. Under this definition, a learning algorithm for $\mathbf{R}$ is required to output hypotheses in R' rather than in R. A representation class $\mathbf{R}$ is polynomially predictable if there exists a representation class R' with a uniform polynomial-time evaluation procedure procedures such that $\mathbf{R}$ is PAC-learnable.

Example 7. *Fixed encoding and Huffman encoding.*

3.2 Data compression

Now we talk about the relationship between learning and data compression. It has been shown that, if any sample can be compressed-that is, represented by a prediction rule significantly smaller than the original sample-then this compression algorithm can be converted into a PAC-learning algorithm.

Suppose C_n is a learnable concept class and that we have been given m examples $(v_1, c(v_1)), (v_2, c(v_2)), ..., (v_m, c(v_m))$ where each $v_i \in X_n$ and c is a concept in C_n of size s. These examples need not have been chosen at random. The data compression problem is to find a small representation for the data, that is, an hypothesis h that is significantly smaller than the original data set with the property that $h(v_i) = c(v_i)$for each v_i. An hypothesis with this last property is said to be consistent with the sample.

Theorem 8. *Let C be a learnable concept class. Then there exists an efficient algorithm that, given $0 < \delta \leq 1$ and m (distinct) examples of a concept $c \in C_n$ of size s, outputs with probability at least $1 - \delta$ a deterministic hypothesis consistent with the sample and of size polynomial in n, s, and $\log m$.*

3.3 Future Research

- Motivation: it is useful.

  Machine learning techniques have been widely used in lots of fields, including data compression. And it seems quite hard to do the analysis from a theoretical perspective. We need a theoretical model to help us do the analysis.

- PAC-learning and lossy data compression: graphics.

  The symmetric difference of c and h: $c\Delta h = (c \backslash h) \cup (h \backslash c)$. We could drop the part we don't care-the ϵ part.

- Exact-learning and lossless data compression: r-junta.

  The general description length of an r-junta

  $$DL_{general} = n \cdot \log n + 2^n$$

  The probably best description length of an r-junta

  $$DL_{best} = r \cdot \log r + 2^r$$

  As we can see, by using DL_{best}, we achieve a compression rate of

  $$r \approx= 1 - \frac{2^r}{2^n} = 1 - 2^{r-n}$$

  That is, we saved about $1 - 2^{r-n}\%$ of the space.

- The compression model is already complicated enough.

  This is a long and winding road.

4 References

- Raymond Board, Leonard Pitt (1990). *On the necessity of Occam algorithms.*
- Robert E. Schapire (1990). *The Strength of Weak Learnability.*
- Micheal J. Kearns, Umesh V. Vazirani. *An introduction to computational learning theory.*
- Elchanan Mossel, Ryan O'Donnell, Rocco A. Servedio (2003). *Learning Juntas.*