Computational Learning Theory

Introduction

The PAC Learning Framework

Finite Hypothesis Spaces

Examples of PAC Learnable Concepts

B. Yanikoglu
Dec 2012

Introduction

- Computational learning theory:
  - Provides a theoretical analysis of learning
  - Shows when a learning algorithm can be expected to succeed
  - Shows when learning may be impossible
  - …

Introduction

- Some fundamental problems addressed by Computational Learning Theory:
  - Sample Complexity: How many examples we need to find a good hypothesis?
  - Computational Complexity: How much computational power we need to find a good hypothesis?
  - Mistake Bound: How many mistakes we will make before finding a good hypothesis?

Computational Learning Theory

Introduction

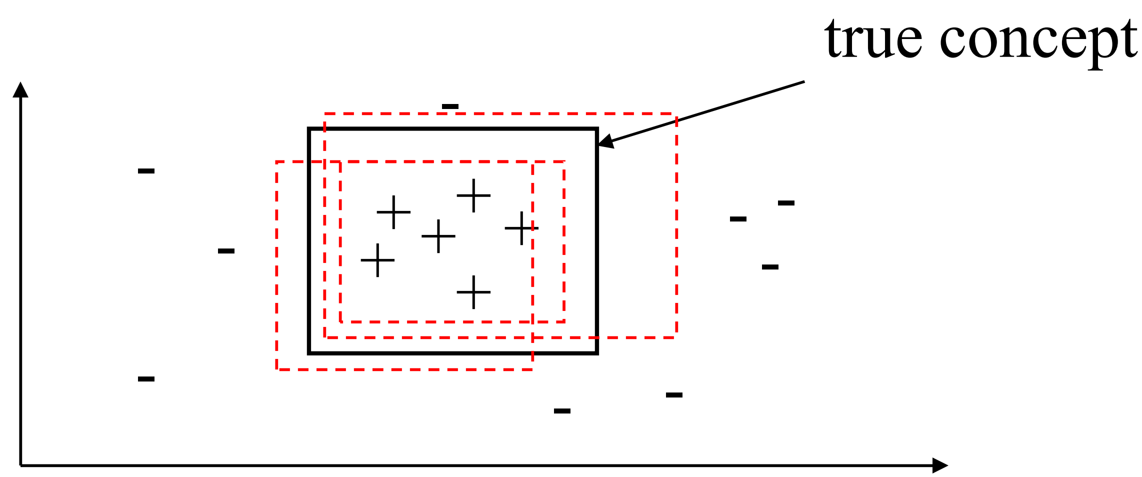
The PAC Learning Framework

Finite Hypothesis Spaces

Examples of PAC Learnable Concepts

The PAC Learning Framework

- Let's start with a simple problem: Assume a two dimensional space with positive and negative examples. Our goal is to find a rectangle that includes the positive examples but not the negatives (input space is R^2):



Definitions

- *Class of Concepts* **C.** Let C be a class of concepts that we wish to learn. In our example **C** is the family of all rectangles in R^2.
- *Distribution* **D**. Assume instances are generated at random from a distribution **D**.
- *Class of Hypotheses* **H**. The hypotheses our algorithm considers while learning the target concept.
- *True error of a hypothesis* **h**
  - $\text{error}_D(h) = \Pr_D[c(x) \neq h(x)]$

True Error of A Hypothesis

- Two Notions of Error
  - Training error of hypothesis *h* with respect to target concept *c* :

    How often $h(x) \neq c(x)$ **over training instances**

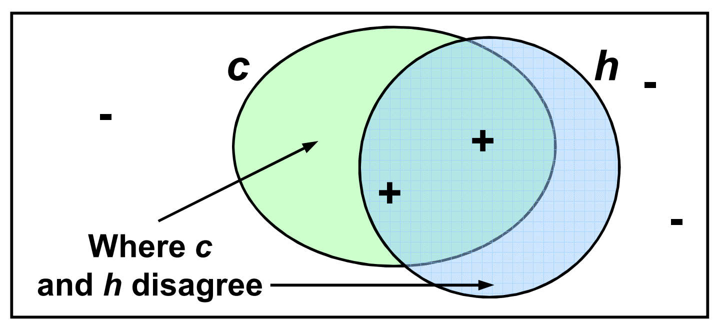
  - True error of hypothesis *h* with respect to target concept *c* :

    How often $h(x) \neq c(x)$ **over random instances drawn from distribution D**

- Definition
  - The true error (denoted $error_D(h)$) of hypothesis *h* with respect to target concept *c* and distribution D is the probability that *h* will misclassify an instance drawn at random according to D.

$$error_D(h) \equiv \Pr_{x \in D}[c(x) \neq h(x)]$$

Instance Space *X*



Two Notions of Error

Training error of hypothesis h with respect to target concept c

- How often $h(x) \neq c(x)$ over training instances D

$$error_{\mathrm{D}}(h) \equiv \Pr_{x \in \mathrm{D}} [c(x) \neq h(x)]$$

Set of training examples

True error of hypothesis h with respect to c

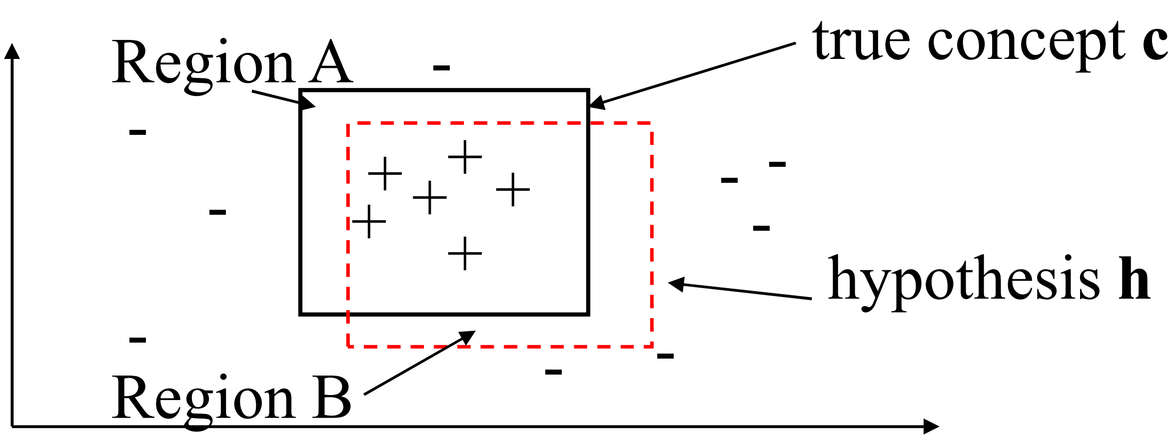
- How often $h(x) \neq c(x)$ over future instances drawn at random from $\mathcal{D}$

$$error_{\mathcal{D}}(h) \equiv \Pr_{x \in \mathcal{D}} [c(x) \neq h(x)]$$

Probability distribution P(x)

True Error

Region A : false negatives
Region B : false positives
True error: probability in regions A and B.



Probably Approximately Correct learning [Valiant84]

typically $\{0,1\}^n$ or $\mathbf{R}^n$

- *Concept class* $\mathcal{C}$ of Boolean functions over domain X
- Unknown target concept $f \in \mathcal{C}$ to be learned from examples
- **Unknown** and **arbitrary** distribution $\mathcal{D}$ over X

Learner has access to i.i.d. draws from $\mathcal{D}$, labeled according to f :

$$(x^1, f(x^1))$$
$$(x^2, f(x^2))$$
$$(x^3, f(x^3))$$
$$\ldots$$

each $x^1, x^2, \ldots$ belongs to X, i.i.d. drawn from $\mathcal{D}$

PAC learning concept class $\mathcal{C}$

Learner's goal:
Efficiently come up with hypothesis that will have high accuracy on future examples.

- For any target function $f \in \mathcal{C}$,
- for any distribution $\mathcal{D}$ over X,
- with probability $1 - \delta$, learner outputs hypothesis $h : X \to \{0, 1\}$ that is ϵ-accurate w.r.t. $\mathcal{D}$:

$$\Pr_{x \sim \mathcal{D}}[h(x) \neq f(x)] \leq \epsilon.$$

Algorithm must be computationally efficient: should run in time

$$\text{poly}(n, \tfrac{1}{\epsilon}, \tfrac{1}{\delta}, \text{size}(f)).$$

Considerations

- We don’t need a hypothesis with zero error. There might be some error as long as it is small (bounded by a constant ε).
- We don’t need to always produce such a good enough hypothesis. The probability of failure should be bounded by a constant δ.
- Goal: With probability $1-\delta$, output a hypothesis h which satisfies $error_D(h) < \varepsilon$.

$$\text{Probability}[error_D(h) > \varepsilon] < \delta$$

- A learner finds a hypothesis **h** that is **consistent** with the training data
  - $\text{Error}_{\text{Train}}(h) = 0$
- The probability that **h** has more than ε true error
  - $\text{Error}_{\text{true}}(h) \geq \varepsilon$

- Hypothesis **h** that is consistent with training data means it got **m** i.i.d points right

- **h** is consistent but "bad": it gets all training data right but has high true error

- Prob. **h** with $\text{Error}_{true}(\mathbf{h}) \geq \varepsilon$ gets one data point right:
  - P(h gets one point right) $\leq$ 1- ε
  - P(h gets m iid points right) $\leq (1- \varepsilon)^m$
  - We want this to be less than δ . So lets set:

    $(1- \varepsilon)^m <= \delta$

  - Since $(1-x) <= e^{-x}$ we have that

    $e^{-\varepsilon m} <= \delta$ or equivalently (taking ln of each side)

    $m >= (1/ \varepsilon) \ln (1/ \delta)$

  - The result grows linearly in 1/ **ε** and logarithmically **1/ δ**

Valiant's "PAC" model [Val84]

"PAC" = Probably Approximately Correct:

- learning problem is identified with a "concept class" $\mathcal{C}$, which is a set of functions ("concepts") $f : \{0,1\}^n \rightarrow \{0,1\}$

- nature/adversary chooses one particular $f \in \mathcal{C}$ *and* a *probability distribution* on inputs $\mathcal{D}$

- the *learning algorithm* now takes as inputs ε and δ, and also gets random examples $\langle x, f(x)\rangle$, x drawn from $\mathcal{D}$

- goal: with probability $1-\delta$, output a hypothesis h which satisfies $\Pr_{x \leftarrow \mathcal{D}} [h(x) \neq f(x)] < \varepsilon$

- efficiency: running time of algorithm, counting time 1 for each example; hopefully $\mathrm{poly}(n, 1/\varepsilon, 1/\delta)$

Example – learning conjunctions

As an example, we present an algorithm ([Val84]) for learning the concept class of conjunctions – i.e., *C* is the set of all AND functions.

- start with the hypothesis $h = x_1 \wedge x_2 \wedge \cdots \wedge x_n$
- draw $O((n / \varepsilon) \log(1/\delta))$ examples:
  – whenever you see a *positive* example; e.g.,

    $\langle 11010110, 1 \rangle$,

    you know that the zero coordinates (in this case, x_3, x_5, x_8) can't be in the target AND; delete them from the hypothesis

It takes a little reasoning to show this works, but it does.

Learning DNF formulas

Probably the most important concept class we would like to learn is **DNF formulas**: e.g., the set of all functions like

$$f = (x_1 \wedge \overline{x_2} \wedge \overline{x_6}) \vee (\overline{x_1} \wedge x_3) \vee (x_4 \wedge x_5 \wedge \overline{x_7} \wedge x_8).$$

(We actually mean poly-sized DNF: the number of *terms* should be $n^{O(1)}$, where n is the number of variables.)

Why so important?

- natural form of knowledge representation for people
- historical reasons: considered by Valiant, who called the problem "tantalizing" and "apparently [simple]"
- yet has proved a great challenge over the last 20 years

ADVANCED

The original PAC model

The trouble with this model is that, despite Valiant's initial optimism, PAC-learning DNF formulas appears to be very hard.

The fastest known algorithm is due to Klivans and Servedio [KS01], and runs in time $\mathbf{exp(n^{1/3} \log^2 n)}$.

Technique: They show that for any DNF formula, there is a polynomial in $x_1, \ldots, x_n$ of degree at most $n^{1/3} \log n$ which is *positive* whenever the DNF is true and *negative* whenever the DNF is false. *Linear programming* can be used to find a hypothesis consistent with every example in time $\exp(n^{1/3} \log^2 n)$.

Note: Consider the model, more difficult than PAC, in which the learner is forced to output a hypothesis which itself is a DNF. In this case, the problem is NP-hard.

REST IS ADVANCED

Computational Learning Theory

Introduction

The PAC Learning Framework

Finite Hypothesis Spaces

Examples of PAC Learnable Concepts

Sample Complexity for Finite Hypothesis Spaces

- Definition: A *consistent learner* outputs the hypothesis **h** in **H** that perfectly fits the training examples (if possible).

- How many examples do we need to be approximately correct in finding a hypothesis output by a consistent learner that has low error?

Version Space ε-exhausted

- This is the same as asking how many examples we need to make the Version Space contain no hypothesis with error greater than **ε.**
- When a Version Space **VS** is such that no hypothesis has error greater than **ε**, we say the version space is **ε**-exhausted.
- How many examples do we need to make a version space VS be ε-exhausted?

Probability of Version Space being ε-exhausted

- The probability that the version space is not **ε**-exhausted after seeing **m** examples is the same as asking the probability than no hypothesis in **VS** has error greater than ε.

- Since the size of the **VS** is less than the size of the whole hypothesis space **H**, then that probability is clearly less than

$$|H| e^{-\boldsymbol{\varepsilon} m}$$

- If we make this less than **δ** , then we have that

$$\boxed{m >= 1/ \boldsymbol{\varepsilon} \; (\ln |H| + \ln (1/ \boldsymbol{\delta}))}$$

Haussler, 1988

- **Theorem**: Hypothesis space H is finite, dataset D with m i.i.d samples, 0 < e < 1: for any learned hypothesis h that is consistent on the training data:

$$P(\text{error}_{\text{true}}(h) > \varepsilon) \leq |H|e^{-m\varepsilon}$$

- **Limitations of Haussler '88:**
  - Consistent classifier
  - Size of hypothesis space

Agnostic Learning

- What happens if our hypothesis space **H** does not contain the target concept **c**?;

- Then clearly we can never find a hypothesis **h** with zero error.

- Here we want an algorithm that simply outputs the hypothesis with minimum training error.

Using PAC bound

- Pick ε and δ, gives you m
- Pick m and δ, gives you ε

Computational Learning Theory

Introduction

The PAC Learning Framework

Finite Hypothesis Spaces

Examples of PAC Learnable Concepts

Examples of PAC-Learnable Concepts

- Can we say that concepts described by conjunctions of Boolean literals are PAC learnable?

- First, how large is the hypothesis space when we have **n** Boolean attributes?
  - Answer: $|H| = 3^n$

Examples of PAC-Learnable Concepts

- If we substitute this in our analysis of sample complexity for finite hypothesis spaces we have that:

$$m >= 1/ \boldsymbol{\varepsilon} \, (n \ln 3 \; + \ln (1/ \boldsymbol{\delta}) \,)$$

- Thus the set of conjunctions of Boolean literals is
- PAC learnable.

K-Term DNF not PAC Learnable

- Consider now the class of functions of k-term DNF expressions. These are expressions of the form

$$T_1 \vee T_2 \vee \ldots \vee T_k$$

  - where V stands for disjunction each of the k terms and
  - and T_i is a conjunction of Boolean attributes.
  - E.g. A 3-term DNF:
    $(x_1 \wedge \neg x_2) \vee (x_6 \wedge x_7) \vee (x_9)$

K-Term DNF not PAC Learnable

- The size of |H| is $k3^n$
- Using the equation for the sample complexity of finite hypothesis spaces:

$$m \geq 1/\boldsymbol{\varepsilon}\ (n \ln 3 + \ln (1/\boldsymbol{\delta}) + \ln k)$$

- Although the sample complexity is polynomial in the main parameters, this problem is known to be NP-complete.

K-Term CNF is PAC Learnable

- But it is interesting to see that a larger family of functions, the class of k-CNF expressions is PAC learnable.
  - k-CNF: Conjunction of disjunctiones where each disjunct has ≤ k literals

- This is interesting because the class of k-CNF expressions is strictly larger than the class of k-term DNF expressions.
  - Can convert k-term DNF into k-CNF by distributivity laws.

k-CNF Expressions

Definition: expressions $T_1 \wedge \cdots \wedge T_j$ of arbitrary length j with each term T_i a disjunction of at most k boolean attributes.

Algorithm: reduce problem to that of learning conjunctions of boolean literals. New variables:

$$a_i(X_1) \vee \cdots \vee a_i(X_n) \rightarrow Y_{a_i(X_1),\ldots,a_i(X_n)}.$$

- the transformation is a bijection;

k-Term DNF Terms and k-CNF Expressions

- Observation: any k-term DNF formula can be written as a k-CNF expression. By associativity,

$$\bigvee_{i=1}^{k} a_i(X_1) \wedge \cdots \wedge a_i(X_n) = \bigwedge_{i_1,\ldots,i_k=1}^{n} a_1(X_{i_1}) \vee \cdots \vee a_k(X_{i_k}).$$

  - Example: $(u_1 \wedge u_2 \wedge u_3) \vee (v_1 \wedge v_2 \wedge v_3) = \bigwedge_{i,j=1}^{3}(u_i \vee v_j)$.
  - But, the number of new variables is exponential in k: $O(n^k)$.