SL07. Computational Learning Theory

Introduction:

- The computational learning theory helps us:
  - Define learning problems.
  - Show that specific algorithms work/don't work for specific problems.
  - Show that some problems are fundamentally hard (No algorithm in a particular class will ever be able to solve them).
- The tools used for analyzing learning problems are similar to those used for analyzing algorithms in computing.
- Resources in machine learning:
  - Time.
  - Space.
  - Data samples.
- Some of the resources useful in analyzing learning algorithms are time and space just like normal algorithms. Furthermore, learning algorithms are measured in terms of the number of samples or data they need to be effective.

Inductive Learning:

- As discussed before, Inductive Learning is basically learning from examples.
- Properties to think of when dealing with Inductive Learning:
  - Probability of successful trainings $(1-\delta)$.
  - Number of training examples (m).
  - Complexity of hypothesis class (Complexity of H): A class of simple hypotheses might not be sufficient to truly express complex concepts. Whereas a class of complex hypotheses can potentially overfit the data.
  - Accuracy to which the target concept is approximated (ε).
  - Manner in which training examples are presented (Batch/Online).
  - Manner in which training examples are selected.

Selecting Training Examples:

- Several ways to select training examples:
  - The learner asks questions to the teacher: The learner selects x and asks about $c(x)$.
  - The teacher gives examples to help the learner: the teacher gives $\big(x, c(x)\big)$ pairs to the learner.
  - Fixed distribution: x chosen from D by nature.
  - Evil distribution: Intentionally misleading.

Learning with Constrained Queries:

- For a set of hypotheses H:
  - The teacher can help the learner get to the right answer using only one question.
  - The learner would have to ask $\log H$ questions to get to the answer on its own.
- Teacher with constrained queries:
  - The teacher has to show what's irrelevant. This can be achieved using two positive examples.
  - The teacher has to show what's relevant. This can be achieved using k negative examples, where k is the number of variables.
  - The answer can be achieved in $2 + k$ examples.
- Learner with constrained queries:
  - It would be very hard (exponential time) for the learner to come up with different cases for positive and negative examples on its own.
- Learner with mistake bounds:
  - It's very hard to learn with constrained queries, so we'll change the rules:
    1. Input arrives.
    2. Learner guesses answer.
    3. Wrong answer charged.
    4. Go to 1.
  - Total number of mistakes will be bound by a certain number.
  - The algorithm will be:
    1. Assume that each variable can be positive and negated.
    2. Given input, compute output.
    3. If wrong, sell all positive variables that were 0 to absent, negative variables that were 1 to absent.
    4. Go to 2.
  - Using these rules, we'll never make more than $k + 1$ mistakes.

Definitions:

- Computational complexity: How much computational effort the learner needs to converge to a correct hypothesis, or to the best hypothesis in the available hypotheses space.
- Sample complexity: How much training examples the learner needs to create a successful hypothesis.
- Mistake bounds: How many misclassifications a learner can make over an infinite run.
- Version space: A consistent learner is a learner that produces a hypothesis that matches the data it had seen. Version space is the space of all the consistent hypotheses.

$$VS(s) = \{h \in H \mid h(x) = c(x)\ \forall x \in S\}$$

PAC (Probably Approximately Correct) Learning:

- The goal with PAC learning is to determine which classes of target concepts can be learned from a reasonable number of randomly drawn training examples with a reasonable amount of computation.
- Training error of h: Fraction of training examples misclassified by hypothesis h.
- True error of h: Fraction of examples that would be misclassified by h on a sample drawn from distribution D.

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

- Parameters:
  - C: concept class.
  - L: Learner.
  - H: Hypothesis space.
  - $|H|$: → size of hypothesis space.
  - D: distribution over inputs
  - Error goal: $0 \leq \varepsilon \leq {}^{1}/_{2}$
  - Certainty goal: $0 \leq \delta \leq {}^{1}/_{2}$
- Since we're drawing our training sample randomly from a distribution D, it's possible to get unlucky and have a bad training set that results in higher error but it's okay because the probability of that happening is very low.
- Formal definition of PAC Learning:
  C is PAC-learnable by *L* using *H* if, and only if, learner *L* will, with probability $1 - \delta$, output a hypothesis $h \in H$ such that $error_d(h) \leq \varepsilon$ in time and samples that grow polynomially with ${}^{1}/_{\varepsilon}$, ${}^{1}/_{\delta}$ and $|H|$.
  - Given D, we can't always expect a learner to produce hypotheses with 0 error, so we have to agree that our learner L does a good job if it outputs an "approximately correct" hypothesis h such that $error_d(h) \leq \varepsilon$.
  - We can't even expect a learner L to always produce hypothesis with such low error (i.e. if the training data has misleading examples). So, we will be satisfied if our learner L produced a low-error hypothesis "most of the time"
  - We will be satisfied if our learner L is Probably (with probability $(1 - \delta)$) going to produce an Approximately Correct ($error_d(h) \leq \varepsilon$) learner. Hence the name PAC.
- In practice, people are less interested in the "time" part of the definition above, and more interested in the number of computations L might take to learn.

Epsilon Exhaustion:

- Given a hypothesis H, target concept c, instance distribution D and set of training examples S, we say the version space $VS(S)$ is $\varepsilon - exhausted$ if, and only if, every hypothesis in $VS(S)$ has true error less than ε:

$$VS(s) is\ \varepsilon - exhausted\ iff\ error_d(h) \leq \varepsilon\ \forall h \in VS(s)$$

- The question is, how many examples would it take to (probably) $\varepsilon - exhaust$ the version space? The answer is given by Haussler's Theorem.

Haussler Theorem – Bound True Error:

- Haussler's Theorem is a way of bounding True Error as a function of the number of training examples.
- If H is finite, and m is a sequence of independent randomly drawn training points, then for any $0 \leq \varepsilon \leq 1$, the probability that the version space is not $\varepsilon - exhasuted$ is bounded above by:

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

  - Suppose we bound the probability in Haussler's Theorem by some desired level δ:

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

  - Rearranging the terms to recover m:

$$m \geq \frac{1}{\varepsilon}(ln|H| + ln\frac{1}{\delta})$$

- If the target hypothesis is not present in the hypotheses space, the learner has to learn the "best" available hypothesis.
- Haussler's Theorem will break with infinite hypotheses spaces.