CS 2416 - Machine Learning Theory

Lecture #1: 15 September 2005

Lecturer: Toniann Pitassi

Scribe Notes by: Danny Heap

Machine Learning tackles many real-world problems and tries to develop algorithms where (we suspect) an explicit algorithm doesn't exist. For example: distinguish chairs from tables.

In Machine Learning Theory (MLT) we develop models for machine learning and evaluate ML algorithms with respect to those models. Benefits of MLT include identifying intractable problems (which need to be re-specified to have any hope of being tractable), and suggest approaches to developing new ML algorithms.

Here are some ML models we'll consider:

- Consistency model
- On-line model
- Occam
- PAC

Some concepts we'll look at:

- VC-dimension (a measure over vectors that gives an idea of the necessary sample size for ML).
- boosting (use an existing ML algorithm with error ϵ to build a better ML algorithm with error whose reciprocal is exponential in $1/\epsilon$.
- hard-core sets and distributions, from complexity theory.

We're heading towards the PAC model. But first, we'll look at some other models in (chronological?) order. Assume that we're restricted to searching for boolean-valued functions on $\{0,1\}^n$. We're given access to instance examples that may include a label from $\{-,+\}$ indicating whether they are negative or positive instances. Our instances may be provided in a batch, after which we're expected to make predictions about the labelling of subsequent instances (or output a hypothesis function that produces such a label), or our predictions may be required after each instance is supplied, in a sequence.

Consistency model

Definition Concept c is a boolean function over $\{0,1\}^n$. The domain variables are denoted by $x_1, \ldots, x_n$, A *labelled example* is a pair $\langle \alpha, \beta \rangle$ where $\alpha \in \{0,1\}^n$ and $\beta \in \{-,+\}$ indicates whether α is a positive or negative example. A *concept class* $\mathcal{C}$ is a set of concepts over $x_1, \ldots, x_n$ **usually** with an associated representation. For example, all CNF formulas over $x_1, \ldots, x_n$.

Definition Algorithm A learns $\mathcal{C}$ in the Consistency Model if given any set S of labelled examples, algorithm A produces concept $c \in \mathcal{C}$ consistent with S if such a concept exists, and outputs "there is no consistent concept" otherwise. Furthermore, the runtime of A should be polynomial in the size of S.

Note that in the above definition, we require that $c \in \mathcal{C}$, otherwise simply taking the disjunction of all the positive examples in S will suffice (and would be ugly, slow, and of no predictive value). If we don't care about run-time, devising A is simple: just try concepts from $\mathcal{C}$. For example, if $\mathcal{C}$ is the set of all monotone conjunctions, then A would simply try all 2^n possible conjunctions of variables in $\{x_1, \ldots, x_n\}$. Thus we also require that A should run in time polynomial in the size of S. (each example in S requires $n+1$ bits).

1. Example 1: monotone conjunctions over $x_1, \ldots, x_n$. Suppose the target concept (which distinguishes the positive and negative labels) is $x_3x_4x_7$, so $|\mathcal{C}| = 2^n$, and you're given labelled pairs:

$$\begin{gathered} \langle 10100110, - \rangle \\ \langle 11111010, + \rangle \\ \langle 00110010, + \rangle \\ \langle 11011111, - \rangle \\ \langle 11001100, - \rangle \end{gathered}$$

   Algorithm A simply takes the conjunction of all the x_i where all positive examples have a 1 in position i, and verifies that this conjunction is falsified by all the negative examples (otherwise output "no consistent concept"). Note that any variable that does not appear in the conjunction had to be eliminated because it was set to zero by some positive example. Thus if our conjunction is not consistent with the negative examples, then no conjunction will be.

2. Example 2: monotone disjunctions over $x_1, \ldots, x_n$. Suppose the target concept $c = x_3 \vee x_4 \vee x_7$, and you're given labelled pairs:

$$\begin{gathered} \langle 01000100, - \rangle \\ \langle 11001100, - \rangle \\ \langle 00001101, - \rangle \\ \langle 10101001, + \rangle \\ \langle 01011100, + \rangle \end{gathered}$$

Algorithm A simply takes the disjunction of all the x_i where all negative examples have a zero in position i, and verifies that this disjunction satisfies all the positive examples (otherwise output "no consistent concept").

3. Example 3: non-monotone conjunctions/disjunctions. These are taken over $\{x_1, \overline{x_1}, \ldots, x_n, \overline{x_n}\}$. We can perform a reduction to the monotone case. Define $y_i = \overline{x_i}$. Under this transformation, the target function is a monotone conjunction over this space of $2n$ variables, $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$.

4. Example 4: k-DNF formula. These are disjunctions of terms, each term being the conjunction of at most k literals. Again we can reduce this to the case of monotone disjunctions as follows. For each possible term of size at most k, introduce a new variable representing this term, and use Example 2.

5. Example 5: 2-term DNF. For example, $c = x_1\overline{x_3}x_5 \vee x_1\overline{x_4}x_5\overline{x_3}$. This **cannot** be learned in polytime unless $P = NP$. This (effective) intractability is due to the fact that A must find a hypothesis that is a 2-term DNF. See the book for a proof of intractability.

6. Example 6: DNF formula. Find a consistent DNF concept, with no restriction on the number of terms. This is trivially learnable in the consistency model, since we can take the disjunction of terms, one term for each positive example. This example raises a red flag, since we are simply memorizing the data. We don't expect that the hypothesis returned will be very similar to the true DNF that the examples came from. We will see later that for true learning (being able to predict well on future examples), there needs to be compression. That is, the hypothesis returned should be of size that is sublinear in the size of the samples, S.

Some problems with the Consistency Model:

- DNF learner is just memorizing the input (we'd like the output concept to be substantially smaller than the input).
- Insisting on the algorithm found by A being in $\mathcal{C}$ seems a bit extreme.
- If the examples aren't completely consistent with any $c \in \mathcal{C}$ there's no way to output something "close."

Online/Mistake Bound (MB) model

As we saw above, the consistency model has the problem that a positive result doesn't always imply finding a rule with good predictive power. The Mistake Bound model addresses this problem by expliitly modelling learning as an on-line process.

Predictions are interleaved with examples (as opposed to learning from a batch of examples before making a prediction). Learning takes place in stages. Each stage consists of:

1. A (the learner) is provided with an unlabelled example.
2. A makes a prediction about the label (equivalently produces a hypothesis for $h \in \mathcal{C}$ that would produce the right label).

3. A is told the correct label (the one produced by the target concept $c \in \mathcal{C}$.

Definition Let $\mathcal{C}$ be a concept class over $x_1, \ldots x_n$ together with an associated representation. A learns $\mathcal{C}$ over $x_1, \ldots, x_n$ in MB model: if for any sequence of examples consistent with some $c \in \mathcal{C}$ the total number of mistakes made by A, when run on this sequence of examples, is polynomial in n and s, where s is the description size of c. (The extra parameter s is necessary since a concept c may be exponentially long in n.) A learns $\mathcal{C}$ in polytime if it completes each stage/step in time polynomial in n and s.

Example 1: non-monotone conjunctions: The learner starts with $h = x_1\overline{x_1}x_2\overline{x_2}\cdots x_n\overline{x_n}$. For each mistake on a positive example, remove all unsatisfied literals from h (there'll be n of them removed on the first mistake, and at least 1 removed on each subsequent mistake). If h makes a mistake on a negative example, output "no consistent concept." The mistakes on the positive examples are bounded by $n+1$, since n literals are removed after the first example, and at least one literal is removed per subsequent mistake, and there are at most $2n$ literals altogether. Further, there will be no mistakes on the negative examples as long as the examples are consistent with some conjunction. This follows because we only threw out literals when we absolutely had to (to be consistent with the positive examples), so if *any* conjunction is consistent with the negative examples as well, then our current conjunction will be.