1 Topic Overview

Lecture 2 started out with the concept of learning by example. We began by discussing how exactly to represent a learning concept, specifically talking about representing a hypothesis that we want to be able to predict with. We then moved on to talking about the *more-general-than* relation, a partial ordering relation that lets us order hypotheses. This relation made use of the most general hypothesis, which returns a positive answer to any query, and the most specific hypothesis, which returns a positive answer to no query, as well as a spectrum of hypotheses in between.

Using the *more-general-than* relation allowed us to define the FIND-S Algorithm, an algorithm that attempts to find a maximally specific hypothesis. This algorithm begins by setting its initial hypothesis to the most specific hypothesis, and then slowly generalizes as new training examples come in. This works fairly well for simple examples that play nicely, but can easily fall apart with more complex real-world type examples.

After examining the troubles with the FIND-S algorithm, we talked briefly about the LIST-THEN-ELIMINATE algorithm. This algorithm is in some ways the opposite of the FIND-S algorithm, as it starts with all possible hypotheses for a given problem and slowly drops hypotheses that aren't consistent with training examples. Of course, this requires a lot more time and space than the FIND-S algorithm.

The last learning algorithm we talked about was the CANDIDATE-ELIMINATION algorithm. This one starts out with both the most general and most specific hypotheses, and makes them more specific or general as needed as negative and positive training examples come in. The whole time, however, it is careful to make sure it keeps all hypotheses consistent with all data it has gathered.

Finally we spent some time talking about biased learning, concluded that no generalization can be done without a bias. This lead us to a definition of the *inductive bias* of an algorithm as the set of assumptions that allows a hypothesis to predict even when it is not 100% sure due to lack of information from training examples.

2 Results

2.1 Most Interesting

Prior to this set of lectures, the only method of "learning" with respect to computers I had been exposed to involved explicitly searching a sample space for good answers. Thus, I was quite interested in the learning techniques we just talked about that didn't involve a real search. Instead, they involved cutting down the sample space without a traditional search algorithm. And that brings me to my first question: How do the methods we talked about relate to such methods as arc consistency used in AI and constraint satisfaction? Both the methods we talked about and arc consistency seem very similar in the way they pare the sample space down, but they do it in fairly different manners. Is there a correlation between the way they work, or do they just coincidently do somewhat of the same thing?

2.2 Least Interesting

The results from this lecture that disappointed me most were the number of restrictions put on the various algorithms we talked about. For example, the FIND-S algorithm was looking great for a while as we discussed it, but when we really examined it more closely it became obvious that it had some major holes. Two examples we talked about are how it can't handle errors well at all and that it can only come up with one consistent hypothesis. However, talking about inductive bias helped put that into perspective a little better. It made it more obvious that even my own learning has such holes, but I am just more able to cope with them by making assumptions and holding biases than these algorithms are.

3 Research Ideas

As we discussed this material during class, a few ideas popped into my head that I thought would be interesting to pursue more deeply. I don't know if they are really research ideas as much as mental diversions, but I think they are close enough.

Real-life learning I think it would be fairly interesting to write up a simple implementation of the FIND-S or CANDIDATE-ELIMINATION algorithms and feed them real world data. For example, I would think simple criteria for whether or not somebody will like a movie could be defined: movie genre, number of big-named actors, producing company, etc could be part of it. Since most people have seen a large number of movies, there would be a fair amount of training data that could be fed into the algorithm, meaning it could learn relatively quickly. It would then be easy to use it to make predictions on movies you haven't seen yet, and then rate how well it did. Of course, the same problems we talked about during class would exist in such an implementation, but if one was careful and didn't take the results too seriously I think this could be an interesting project to play with.

Two algorithms together At one point in the lecture we discussed how to choose good queries and training examples to help our machine learn faster and more usefully. After thinking about that a bit, I began wondering if it would be possible to have to learners work together on that task: one to do the learning, and the other to pick good queries and training examples. In some ways it seems fairly possible to do, as the original learner would just have to tell the query learner whether or not it thought the query or example was useful. However, it would also be a bit of the blind leading the blind, with one learner blindly trying to make another one understand what it thinks might be good information. So, in the end I am not sure how useful it would be. But it might be fun to play with and find out.

4 In Depth Coverage - The FIND-S Algorithm

During lecture we spoke of the *more-general-than* partial ordering relation, which gives us a way to order boolean functions according to their generality and specificity. This relation is used in the FIND-S algorithm to find the maximally specific hypothesis that is still consistent with the training examples presented to it.

```
1. Initialize $h$ to the most specific hypothesis in $H$
2. For each positive training instance $x$
    • For each attribute constraint $a_i$ in $h$
        – If the constraint $a_i$ is satisfied by $x$
        – Then do nothing
        – Else replace $a_i$ in $h$ by the next most general constraint that is
          satisfied by $x$
3. Output hypothesis $h$
```

Figure 1: The FIND-S algorithm[1]

As can be seen in the algorithm in Figure 1, the FIND-S algorithm starts with the most specific hypothesis, the hypothesis which predicts "no" under all circumstances. With no training examples, this hypothesis will therefore be maximally specific. The algorithm then iterates over a list of training examples, skipping any negatives ones. Each positive example is checked for consistency with the current knowledge, and the hypothesis is made more general as needed. Since the hypothesis is only made more general when a new training example proves it wrong, and it is only made general enough to accommodate that new example, the hypothesis stays maximally specific. Of course, only positive examples are needed for this algorithm because we start with the most specific hypothesis, which effectively takes into account all possible negative hypotheses.

Unfortunately, the simple FIND-S algorithm has some shortcomings. One big problem with it is that, if there are more than one consistent hypotheses, it can miss the target hypothesis and return a less-than-optimal one. This can be helped by incorporating some backtracking measures into the algorithm, but our simple method has to guard against such a situation. The fact that the algorithm always returns the most specific hypothesis can also be a problem, as that isn't always what is needed. There are times in real life that the most general hypothesis is desired, but this algorithm has no way of returning anything but the most specific hypothesis. But perhaps worst of all is the need for all of the training examples being fed into FIND-S to be consistent. The algorithm has an implicit assumption that the training examples are all flawless, but in real life this is not always the case. Just one flawed example can cause the FIND-S algorithm to greatly over generalize, making this algorithm fairly useless for all but the most perfect of data sets.

Despite its shortcomings, FIND-S is a good starter algorithm whose ideas can be adapted to make more robust and noise-tolerant algorithms such as the CANDIDATE-ELIMINATION algorithm. It also is a good starting point for our discussion in machine learning, as it points out some of the problems that an idea that looks fine on paper can have in real life.

References

[1] MITCHELL, T. M. *Machine Learning*. McGraw-Hill, 1997.