Overview:

Abstract

This topic review will cover concept learning. More specifically, we discussed the Find-S Algorithm, the List-Then-Eliminate Algorithm, and the Candidate Elimination Theorem, in addition to general concept learning. We finished our discussion with an overview of inductive bias and its necessity in learning algorithms.

The Find-S Algorithm

The Find-S Algorithm was the first algorithm we addressed. This algorithm tries to find the most specific target concept. To do so, it assumes that the target concept exists in the hypothesis class and that the most specific representation will yield the best results (it is the best representation).

How does it work? We begin by initializing our hypothesis h to the most specific hypothesis in our hypothesis class H [1]. We then proceed through our training examples. If we find a negative example, we discard it, making no changes to h. If, however, we have a positive example, we then alter h so that it accepts this new example (as well as all previous training examples) but remains as specific as possible. We then continue this process until we have seen all of the training data.

There are some shortcomings to this method, however. Although it will find a hypothesis consistent with the data, there is no way to determine that this hypothesis is the only target concept consistent with the data. Further, there is no way to determine how many consistent hypotheses exist within the hypothesis class [1]. In addition, the Find-S Algorithm is very sensitive to incorrect data (or noise) that may be in the training examples. Further, the only contributors to the hypothesis are the positive examples in the training data [2].

The List-Then-Eliminate Algorithm

The List-Then-Eliminate Algorithm is another learning algorithm. This algorithm begins with a full Version Space (a list containing every hypothesis in H). Then for every training example, we remove every hypothesis (from the Version Space) that does not agree with the training example. Once we are finished with our training examples, we output the list of remaining hypotheses. The problem with this algorithm is that it takes $\Omega(|H|)$ to enumerate all the hypotheses [2].

The Candidate Elimination Algorithm

The third learning algorithm we considered was the Candidate Elimination Algorithm. This algorithm addresses several of the limitations of Find-S [1]. The goal of this algorithm is to output a set of all hypotheses that are consistent with the data. With this algorithm, we can do this without explicitly enumerating all of the members of the hypothesis set.

This method begins with a maximally general set of hypotheses as well as a maximally specific set of hypotheses. Call them G and S respectively and define them such that $G \subseteq H$;

$S \subseteq H$. Then, for each training example $d \varepsilon D$, we consider the following operations on G. If d is a positive example, we remove from G any hypothesis that is inconsistent with G. Then for any hypothesis $s \ \varepsilon \ S$ that is not consistent with d, remove s from S and add to S all minimal generalizations of h of s such that h is consistent with d, and some member of G is more general than h. In other words, we remove any hypothesis that is not consistent with G and make S more general ("minimal generalization") [2]. If d is negative, we perform a similar operation, but we make G more specific and remove from S any hypothesis that is inconsistent with d [2].

Inductive Bias

We also learned that any algorithm that does not assume anything about the structure of its target function degrades to a method of rote memorization. If this happens, the algorithm is unable to make any predictions on data that it has not seen (we always have a 50/50 split). Thus, it is impossible to generalize without some sort of bias.

Most and Least Interesting Concepts:

We feel the most interesting concept we covered was the concept of bias in learning. We found it very interesting that an algorithm that lacked bias also lacked the ability to make decisions (Essentially, it became a "Rote Learner.").

The least interesting concept was the concept was the List-Then-Eliminate Algorithm. Maybe I just need to review it a bit more to get a better understanding of it.

Questions on the Material (2-3):

Can we create a learning algorithm that can make predictions on presented data as well as use this new data as new training examples? More specifically, can we create an algorithm that is always learning or are forced to simply give it some data and then let it fend for itself, never to learn again?

Currently, I would be unable to code an algorithm capable of asking it's own questions (queries). Are we going to explore this topic in more detail?

Research Ideas (2-3):

Two research ideas that seemed interesting would be to implement both the Candidate Elimination and Find-S algorithms and do a comparative study of each one's effectiveness. The second would be to create some sort of game playing algorithm (perhaps Poker, e.g. No Limit Texas Hold 'Em – in the spirit of "Rounders"). Even though it may be a problem that has an existing solution, it is a problem that I have not personally solved.

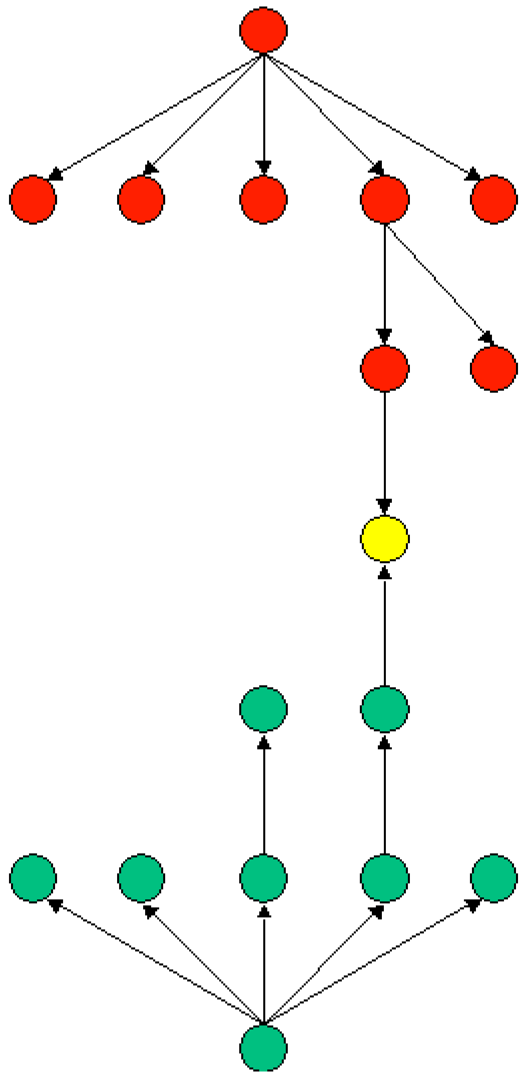
The Version Space Method (Another Way of Viewing Candidate Elimination)

Introduction

A **version space** is a hierarchical representation of knowledge that enables you to keep track of all the useful information supplied by a sequence of learning examples without remembering any of the examples [3]. The **version space method** is a concept learning process accomplished by managing multiple models within a version space [3]. The version space also represents all possible hypotheses.

The Depiction

It is important to note that the version space represents two complementary sets. One set contains all nodes (or sets) that are overly general and the other contains all nodes that are overly specific. Our bias for this representation is as follows. We assume that all training data is correct. In addition, the correct result occurs in the version space. The version space can be depicted as a tree (see Fig 1). Note that each of the nodes is a model (or hypothesis) in the version space. Each model consists of attributes that are representative of our target concept.



At the top of the tree we have the most general hypotheses.

The next row is an expanded version of the first. This row of hypotheses is slightly more specific than the root node.

As training data (positive examples) is processed, inconsistent nodes are removed from the general specification.

Eventually we converge to a solution.

Any hypothesis that is inconsistent with the training data (negative instances) is removed from the tree.

The specific hypothesis is expanded to form more nodes that are slightly more general.

This is the most specific hypothesis.

Fig 1. Tree representation of a version space [3].

“The key idea in version space learning is that specialization of the general models and generalization of the specific models may ultimately lead to just one correct model that matches all observed positive examples and does not match any negative examples” [3]. This means that each time a positive example is used to generalize a “more specific” hypothesis, the general hypotheses (nodes) are removed.

The Downside of this Method (and the Candidate Elimination Algorithm)

Even though it is quite fast “close to linear,” it still has some problems [3]. This algorithm assumes that if it has worked, it will likely work again. Therefore, it is sensitive to errors in the training data. For example, an error in the training data could cause a node to be pruned (removed) from the tree even though it should have remained. Obviously, the further up (towards the root) this occurs, the more likely it is to cause errors in the final hypothesis.

Conclusion

We have tried to present a different conceptual model of the Candidate Elimination Algorithm as well as provide a few new facts about the algorithm itself. While both methods effectively follow the same logic, it can be useful to consider each representation.

References:

[1] Tom M. Mitchell. Machine Learning. McGraw Hill. 1997.
[2] Stephen D. Scott. Lecture 2 Slides
[3] Dr. Howard Hamilton, Ergun Gurak, Leah Findlater, and Wayne Olive. Version Spaces. http://www.cs.uregina.ca/~dbd/cs831/notes/ml/vspace/3_vspace.html. 2001.