INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)

www.ijprems.com
editor@ijprems.com

Vol. 03, Issue 12, December 2023, pp : 309-312

e-ISSN : 2583-1062

Impact Factor : 5.725

AN ANALYSIS OF THE CANDIDATE-ELIMINATION ALGORITHM'S COMPUTATIONAL COMPLEXITY

Janani A. S[1]

[1]M.Sc., Department of Computer Science, Fatima College, Madurai, India.

ABSTRACT

The candidate elimination procedure uses a hypothesis space, H, and a set of examples, E, to systematically create the version space. One way to gradually reduce the version space is to introduce instances one by one, eliminating hypotheses that don't fit each specific example. Unlike the Find-S method and List-then-Eliminate algorithm, the Candidate Elimination algorithm finds hypotheses that match all the training instances that are presented. It considers both positive and negative instances, eliminating any theories that don't hold true for every example. This paper focuses on the computational complexity of Candidate Elimination Algorithm so that this algorithm can be used for real time application development.

Keywords: Concept Learning, version space, Specific hypothesis, General Hypothesis.

1. INTRODUCTION

Machine learning is the process of teaching a machine to maximize a performance criterion by utilizing sample data or historical information. Our model is specified up to a point, and learning is the process of running a computer program to maximize the model's parameters based on training data or prior knowledge. The definition of learning in accordance with machine learning is "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks T, as measured by P, improves with experience E."

Example:

Handwriting recognition learning problem.

- Task T: Recognizing and classifying handwritten words within images.
- Performance P: Percent of words correctly classified.
- Training experience E: A dataset of handwritten words with given classifications.

2. STEPS INVOLVED IN CANDIDATE ELIMINATION ALGORITHM

Candidate Elimination Algorithm is an example of supervised learning where the hypothesis from version space is compared with the training dataset and the hypotheses that are inconsistent with the training dataset are removed from the version space.

Terms:

- **Concept learning:** Concept learning is basically the learning task of the machine (Learn by Train data)
- **General Hypothesis:** Not Specifying features to learn the machine.
- **G = {'?', '?','?','?'…}:** Number of attributes
- **Specific Hypothesis:** Specifying features to learn machine (Specific feature)
- **S= {'pi','pi','pi'…}:** The number of pi depends on several attributes.
- **Version Space:** It is an intermediate of general hypothesis and Specific hypothesis. It not only |ust writes one hypothesis, but a set of all possible hypotheses based on training data-set.

Steps involved:

Step1: Load Data set

Step2: Initialize General Hypothesis and Specific Hypothesis.

Step3: For each training example

Step4: If example is positive example

if attribute value == hypothesis value: Do nothing.

else: replace attribute value with '?' (Basically generalizing it)

Step5: If example is Negative example.

Make generalize hypothesis more specific.

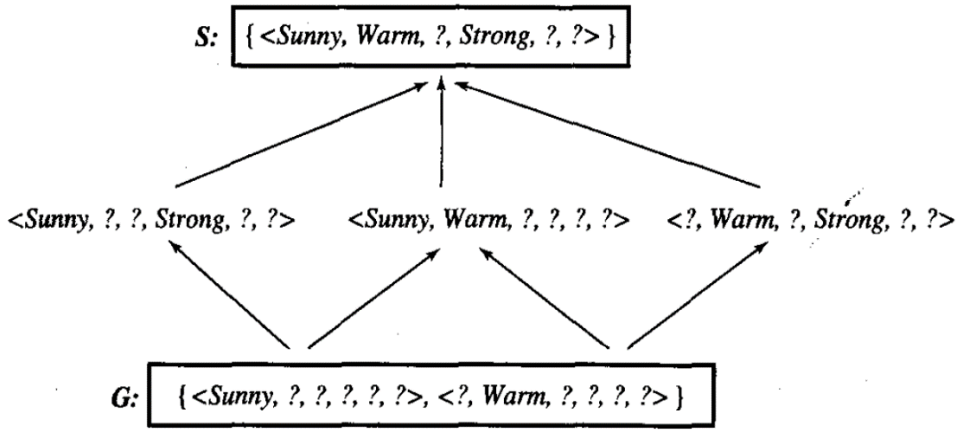
INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)

www.ijprems.com
editor@ijprems.com

Vol. 03, Issue 12, December 2023, pp : 309-312

e-ISSN : 2583-1062

Impact Factor : 5.725



3. COMPUTATIONAL COMPLEXITY FOR SINGLE UPDATE

The boundary sets for the current version space are updated by the candidate-elimination process to consider each new instance that is obtained.

The candidate-elimination algorithm requires three basic computations. The first determines which description is the most generic. This calculation is used to determine if a concept definition covers an instance and to compare the relative generality of two concept definitions.

Finding the minimal generalizations of a description that includes an example is the second calculation that is required. There may be more than one of these minimum generalizations because the description language is so much less restrictive; f will indicate the maximum number of such generalizations. To exclude an instance yet stay above another description, the third and final computation required is to compute the minimum specializations of a description. This step's computational complexity will be designated as Cspec. Once more, there may be several specialties of this kind, and B will decide how many of these specialties are allowed. Whether an instance is positive or negative determines how it is handled by the candidate-elimination method. For positive data, it takes O (|G| *Cabove*?) time to remove components of a G set that do not cover an instance since each G-set element needs to be evaluated to see if it covers the given instance. Since each S-set member needs to be generalized to cover the description, generalizing items from the S set to include the new instance requires O (|S| Cgen). Since each element in the S-set may be generalized in a maximum of m ways, the resultant S set will have at most |S| m items.To prune nonminimal items, each pair of these elements needs to be compared, which requires O ((|S| m) 2 Cabove?) time. Lastly, each element needs to be compared to every G-set element, which adds to the cost of O (|S| |G| m Cabove?). Because the first term may be deleted because it is always less than or equal to the last term, the total time is therefore O (|G| Cabove? + |S| Cgen + (|S| m)2 Cabove? + |S| |G| m Cabove?), or O (|S| Cgen + (|S| f) 2 Cabove? + |S| |G| m Cabove?). The analysis for negative data is the same. Each piece in the G set needs to be specifically designed to fit the present instance. S-set elements covering an instance must be removed in O (|S| Cabove?) time.

G-set elements specialization requires O (|G| Cspec). The final G set will include a maximum of |G| b elements.

Using pairwise comparisons to prune nonmaximal items takes O ((|G| b) 2 Cabove?) time. The last step is to compare each element to each S-set element, which adds an extra expense O (|S| |G| b Cabove?).

Therefore, the total time is O (|G| Cspec + (|G| b) 2 Cabove? + |S| |G| b Cabove?).

4. COMPUTATIONAL COMPLEXITY FOR MULTIPLE INSTANCES

The preceding section examined how difficult it is to handle a single instance. This section considers the changes in computational complexity over several cases. The assumption that Cabove? Cabove?, and Cabove? are all constant-time operations leads to the complexity of version spaces being $O(sg(p + n) + s^2 p + g^2 n)$, where s and g are the highest sizes achieved by the S and G sets, and p and n are the number of positive and negative examples handled. Every S-set member is compared to every G-set element for each positive and negative instance to get the sg(p + n) term. In order to exclude non-minimal items, the S^2p emerges for positive data by comparing each member of a new S set to the other new S-set elements. Similarly, by comparing pairs of G-set members, the g^2n is derived from negative data. If the analysis of the preceding section were to be applied to it, then all instances of Cabove?, Cgen, and Cspec might be eliminated. The analysis assumed that the costs of comparisons, generalization, and specialization were constant across time. This indicates that the complexity of the positive data may be reduced to $O((|S| m)^2 + |S| |G| m)$ and that of the negative data to $O((|G|b)^2 + |S| |G| b)$. Additionally, the words |S| m and |G| b may be put by the limits s and g, respectively, because they are constrained by the maximum size of the new S and G sets after processing a single

example. This results in a complexity of O (s^2 + s |G|) for positive data and O (g^2 + |S| g) for negative data. And lastly, because |S| and |G| are also always constrained by s and g, they may also be substituted in a similar fashion to get O (s^2 + sg) for positive data and O (g^2 + sg) for negative data. The overall complexity will thus be O (p [s^2 + sg] + n [g^2 + sg]) = O (sg (p+n) + s^2 p+g^2n) if there are p positive examples and n negative cases.

5. RANGES AND HIERARCHIES OF GENERALIZATION

The preceding sections did not impose limits on the description languages; instead, a more thorough consideration of computational complexity can only be done within the framework of a specific idea description language. This section applies the analyses of Section 3 to two types of feature hierarchies for con|unction languages: tree-structured (a generalization hierarchy example is provided in Figure 1), and ranges of the form a<x<b, where a and b are required to be selected from a set of prespecified values (e.g., x may range over reals between 0 and1000, and a and b over integers between 0 and1000). K will be used throughout this research to indicate how many characteristics there are.

As shown in the following complexity analysis, these forms' features exhibit two qualities that support learning. First, they both have the same characteristic, which is that any two feature values have a single minimum generalization (i.e., they are upper semi-lattices). As a result, f = 1, and if the S set starts off as singleton, it always stays singleton (i.e., |S| = 1). The reason for b = k is that there is only ever one minimum specialization of a feature that may remain above a value without excluding a second. The complexity analyses of Section 3 are simplified to O (cgen+|G| Cabove?) for positive data and

O (|G|cspec + (|G|k) ^{2}cabove?) for negative data through these.

Moreover, more accurate specification of cabove?, cgen, and cspec is possible for both tree-structured and range-valued features. The comparison of two feature values for relative generality in a tree-structured feature requires O(d) time, where d is the maximum depth of any feature's hierarchy. Therefore, the caboose? function takes O(kd). If the supplied value is within the relevant range of the concept definition, then all that has to be checked for each feature for ranges is whether it falls within that range. If this takes constant time, the answer is O(k). The computation of the simplest generalization of a description to encompass an instance, or cgen, takes O(k) time for ranges and kd time for tree-structured features. Ultimately, computing the minimal specialization of a description to exclude an instance while remaining above another description, or cspec, takes O(kd) time for tree-structured features and O(k) time for ranges.

Because of these more sophisticated complexity evaluations, which account for tree-structured aspects, providing a favorable example necessitates Processing a negative example needs O (|G|kd + (|G|k) ^{2} kd) = O (|G|^{2}k^{3}d) time, whereas O (kd + |G|kd) = O (|G|kd) time. In ranges, positive data needs two

O (k + |G|k) time and negative data equals O (|G|k). The time is O (|G|k + (|G|k) ^{2} k) = O (|G| ^{2} k ^{3}).

6. CONCLUSION

The conclusion we are coming to after the study are as follows:

- For the two popular languages examined here (tree-structured hierarchies and ranges), the computational complexity is largely dependent on the size of G sets. A single update will be costly for both positive and (much more so) negative data if the G set is huge.
- When dealing with tree-structured hierarchies, the computational complexity varies based on the number of features. It is only linear for positive data and cubically for negative data, where negative data indicates poorer behavior.
- In general, the number of minimum generalizations of two feature values affects the complexity of conjunctive languages most severely. If this number is more than one, the number of features k will determine the value of f exponentially. The influence of the number of maximal specializations of a feature to exclude a value that is maintained above another, on which b relies linearly, may be contrasted with this. Therefore, in feature hierarchies that are not upper semi-lattices, learning might rely exponentially on the amount of features; also, processing positive data would exhibit this exponential complexity, in contrast to the preceding two situations.

The main emphasis of this study is the intricacy of a single step in the candidate-elimination method. The intricacy of a particular group of cases is only discussed within the framework of Mitchell's initial analysis. The primary cause of this is that the behavior of version spaces across several instances is already well-known; specifically, Haussler (1988) demonstrated using an example set of data that the size of the G set can grow grows exponentially in the number of cases, even for basic Boolean features. The topic of whether it is feasible to handle data generally so that the G set never grows exponentially in size if the final G set's size is polynomial in the number of examples at the conclusion of learning remains unanswered. The INBF approach may be extended even further, as more recent work (Hirsh, 1992) demonstrates. Specifically, the list of fall negative data can replace the usage of the G set for most desired version-space operations and produce assured polynomial-time learning.

7. REFERENCES

[1] T.G. Dietterich, G. Dromey, B. London, and K. Clarkson. Education and inferential reasoning. Volume III of The Handbook of Artificial Intelligence, edited by P. Cohen and E.A. Feigenbaum. Los Altos, CA / William Kaufmann, 1982.

[2] D. Haussler, second.AI Learning Algorithms and Valiant's Learning Framework for Quantifying Inductive Bias. Artificial Intelligence, 26(2):177–221, September 1988.

[3] Hirsch, H. Learning in polynomial time with version spaces. In July 1992, San Jose, California, Proceedings of the National Conference on Artificial Intelligence.

[4] Thomas M. Mitchell. Stanford University, PhD thesis, "Version Spaces: An Approach to Concept Learning," December 1978.

[5] Mitchell, T.M. "Generalization as research." Artificial Intelligence, 18(2), 203–226, March 1982.

[6] P.S. Rosenbloom and B.D. Smith. Gradual non-reversal concentrating on exponentially bounded generalization algorithms for inverse space conversions. In August 1990, Boston, MA, Proceedings of the National Conference on Artificial Intelligence, pp 848–853.