Alternative Approach for Learning from Examples

Ludmil Dakovski, Zekie Shevked

Abstract: *This paper introduces a new effective algorithm for learning from examples. Our approach is based on representing sets of positive and negative training instances as logical functions in sum-of-minterms form. The main goal is to find a more compact representation of classification function and use it to further prediction of unknown cases. We propose an algorithm that finds this functions prime implicants using a fast strategy to minimization. The found prime implicants correspond to positive examples and does not correspond to negative ones, so they can successfully classify new unknown instances.*

Key words: *Learning from Examples, Classification, Prime Implicant, Function Minimization.*

INTRODUCTION

Concept learning from examples is a problem with great actuality and wide application area. There are a lot of domains where automated extraction of more general description from available positive and negative examples is required. Science and industry needs employed machine learning researchers to improve for practical use existing techniques or to develop new faster methods.

Nowadays one of the most popular research areas in Machine Learning is learning from Examples. The main idea is having correctly classified training instances to learn a classifier that describes the examples in a more compact way and that can also be used to classify new (unknown) cases [4]. Because of high importance and wide application area of this problem a lot of work has done in this field, there are many publications and practical tools for learning from examples and classification. Fundamental theoretical value in this area has the version space approach of Mitchell [10]. It searches for conjunctive classification rule but some important shortcomings make this method impractical. Other researchers consider improvements by extending conjunctive to disjunctive concepts. It is common opinion that conjunction does not fully represent possible concepts and both disjunction and conjunction should be used.

The problem of finding minimal descriptions for a class using examples as input is NP-complete. It is time and resource consuming and in some cases even impossible to examine all possible examples. That is why our approach considers only available training instances. We use sum-of-minterms form (disjunction of conjunctions) to represent two logical functions - one that corresponds to all positive examples and other that corresponds to all negative examples. The main goal of the algorithm is to find minimal representation that corresponds to all positive and probably some of unknown cases but does not correspond to any negative sample. To do this we apply an effective and fast minimization algorithm. As a result we get prime implicants list and use it for further classification. If a new unclassified instance is corresponded to implicants list it belongs to the learned concept, i.e. it is classified positive; otherwise it is classified negative. We are going to publish a detailed description of this approach with a proof.

The paper is organized as follows: Section 1 discusses in brief some fundamental methods for learning from examples, section 2 presents the new approach and section 3 mentions experimental results from applying our method. Conclusion follows and future tasks are discussed at the end.

APPROACH FOR LEARNING FROM EXAMPLES

1. State of problem

1.1.Version space. Version spaces are an approach to concept learning [10, 11, and 12]. They are defined as sets of descriptions in concept languages that correctly classify training instances. When concept languages are partially ordered a version space is delimited by two boundary sets – the sets of most specific and most general hypotheses. Boundary sets are sets of minimal and maximal descriptions in the version space. It was proven that they correctly represent the version space [10, 16]. This approach is used in Mitchell's Candidate-Elimination (CE) algorithm. It incrementally learns concepts from positive and negative instances performing a bidirectional search through the space of hypotheses described by the concept language [1]. Hypothesizes in this space are considered as conjunction of attribute-value pairs. The algorithm represents and updates the version space by maintaining the set S containing the maximally specific consistent concepts and the set G containing the maximally general consistent concepts. Each example leads to changes in the version space. A positive example prunes concepts in G which do not correspond to it and causes all concepts in S which do not correspond to the example to be generalized just enough to correspond to it. A negative example prunes concepts in S that correspond to it and causes all concepts in G that correspond to the example to be specialized just enough to exclude it. As more examples are examined, the version space shrinks, possibly converging to a single target concept [1]. This approach has important shortcomings. It is shown that CE algorithm has limited expressive power [8]. That is why since practical applications of the CE algorithm requiring a restricted concept language, it may be unable to induce consistent concept [1]. In standard version spaces learning is a search for conjunctive hypothesis, so learned concept is restricted to be conjunctive. To overcome this problem alternative version-space representations were introduced in [6, 7, 8, 13, 14, 15, 16, 17, and 18]. Another negative side is that the size of the general boundary can grow exponentially in the number of training examples [5]. Furthermore the CE algorithm can not handle noisy training examples. If there is any incorrectly classified instance, it leads to completely different solution.

1.2.Attribute-value languages. Mitchell distinguishes between an instance language and a concept language. However, many approaches to concept learning employ in fact the same language for describing instances and concepts, a strategy that has been referred to as the Single Representation Trick [3]. A widespread representation is so called attribute-value language. It is a propositional language in which propositions are attribute-value pairs. Each attribute has a designated set of allowed values. Attribute-value pairs may be combined into conjunctive expressions for describing instances or concepts. The attribute-value pair can be considered as a predicate (statement which can have a truth value) and the set of these pairs – as a conjunction of the corresponding predicates. Usually an example has values for all of the attributes but a concept may have not a value for one or more attribute. This is indicated by "?" instead of value and means that this attribute is "don't care" attribute, i.e. it can have anyone of his possible values. The basic advantage of the attribute-value language is that it allows a straightforward definition of derivability relation. A concept is said to correspond to an example if the attribute values of this example are a super-set of the premises of the concept. In other words a concept corresponds to an example if for each attribute-value pair it has the same value as the example. Only allowed differences are when in the concept description the attribute, whose value differs from the corresponding one in the example, is "don't care". By analogy a concept is defined to correspond to (to be more general than) another concept when the attribute values of the first concept is a super-set of the premises of the second one.

2. Novel algorithm for learning from examples

We propose a novel effective algorithm for learning from examples. Our approach aims at finding a more compact classification function that approximates target concept by corresponding to all positive but no one negative examples. The algorithm is based on logical function minimization. Attribute-value language is used to represent examples as predicates. Each example is a conjunction of values corresponding to attributes. We consider target concept as a logical function with number of arguments equivalent to the number of attributes. For each combination of input variables target function F (classification function) takes value "true" or "false" indicating whether this combination belongs to the learned class or not. But not all of input combinations have a specified output. We know the output only for training instances and in practical domains they are a paltry part of all possible cases. What we have is a partly defined function and the goal is to find a good approximation that is correct for all training instances and would successfully predict the class of unknown cases. Such an approximation function should be more compact in order to make further classification easier.

From two-valued logic we know that prime implicants of a function are its shortest description. In comparison with two-valued logic where an attribute can take one of two possible values (0, 1), in our approach every attribute can take one of several predefined values and in common case their number is different for particular attributes. But yet it is a matter of logical function with two output values depending on input combinations. By analogy we define the concept prime implicant as a covering to whose number of restricted attributes is minimal, i.e. it would become incorrect (inconsistent) if any of its specified attributes were allowed to take all of possible values ("don't care" attribute). If so, such an attribute will cause the concept to correspond to one or more negative training instances and it can not be used for classification.

The process of finding prime implicants is called minimization. This way the goal of the algorithm is specified to minimize function in order to get its prime implicants. We stand on a minimization approach for partly defined logical functions described in [2]. There it is proved that the method works correctly in two-valued logic. In [2] there are considered two functions – F1 and F0. F1 is a disjunction of all positive training examples represented as conjunctions. It is an approximation of target classification function that interprets all undefined states as negative. F0 is a disjunction of all negative training examples represented as conjunctions. It is an approximation of target function that considers all undefined states as positive. Although these two functions correctly classify training instances, they would be a bad choice for solution because they are very restricted and do not generalize examples any. But they can be used successfully in order to get something better in between.

It is proven that we can get all prime implicants that correspond to the minterm U from function F and are a result of all possible reductions of the set of F1s minterms, by inverting sum-of-minterms form for F0 and removing from the got expression all of the arguments (attributes) that are not in the same form in U and adducing the expression to disjunction of conjunctions. Inverting F0 results in ^F0 a conjunction of disjunctions of attribute values. After applying the rules of Boolean algebra we get a disjunctive expression of conjunctions. According to [2] this expression contains all prime implicants of function that corresponds to all positive and all undefined instances (F1). But we do not need such a restricted function – it would classify every example that is not in the set of negative examples as belonging to the class. Right here we put in action the positive examples. The goal is from got list of prime implicants to remove these that do not correspond to any positive example. For each positive example we should remove all prime implicants that does not correspond to exactly this positive example. According to definition the corresponding expresion can not have a value that differs from the corresponding value in the corresponded instance. This is a criterion for removing unnecessary prime implicants from the resulting list.

But we can do this still earlier. If from inversion of F0 are removed all attributes whose values differ from those in the current positive example, we guaranty not to get unnecessary implicants and this improves the performance. This way from the whole list of all prime implicants of ^F0 we will get only these that correspond to exactly this positive instance and some undefined ones. When this is done for all positive examples we have got a list of prime implicants whose disjunction corresponds to all positive and some of undefined but no one of negative instances. This disjunction of conjunctions represents a learned concept; it describes the class in a more compact way because of the properties of prime implicants.

Let us show this on the classical example [19]. We are trying to learn under which circumstances a patient gets an allergic reaction after meals. We have the following observations about having a reaction:

Table 1. Training data

RESTAURANT	WEEK DAY	MEAL	PRICE	REACTION
Sam	Friday	breakfast	cheap	Yes
Rubin	Friday	lunch	expensive	No
Sam	Saturday	lunch	cheap	Yes
Sara	Sunday	breakfast	cheap	No
Sam	Sunday	breakfast	expensive	No

F1 = Sam * Friday * breakfast * cheap V Sam * Saturday * lunch * cheap
F0 = Rubin * Friday * lunch * expensive V Sara * Sunday * breakfast * cheap V
Sam * Sunday * breakfast * expensive

^F0 = (^Rubin V ^Saturday V ^lunch V ^expensive) * (^Sara V ^Sunday V
^breakfast V ^cheap) * (^Sam V ^Sunday V ^breakfast V ^expensive)

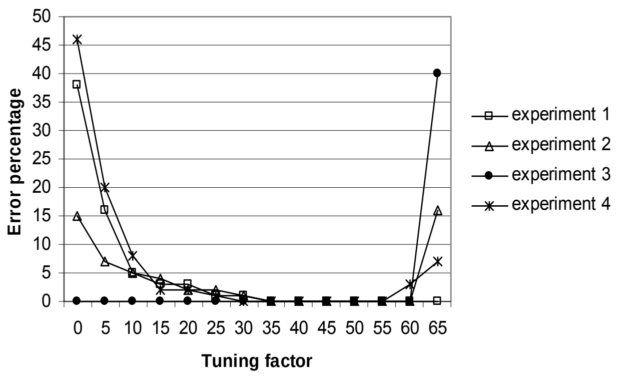
For each one positive instance we find all prime implicants that correspond to it but do not correspond to negative instances:

F0,1 = (Sam V breakfast V cheap) * (Sam V Friday) * (Friday V cheap) =
= Sam * Friday V Sam * cheap V breakfast * cheap V Friday * cheap
F0,2 = …
Finally we have got these prime implicants that correctly classify input examples.

3. **Experimental results**

In order to perform experiments the datasets from the UCI machine learning repository are used. The algorithm was applied to the Tic-Tac-Toe Endgame dataset donated by David W. Aha [9]. The Tic-Tac-Toe Endgame dataset encodes the complete set of possible board configurations at the end of tic-tac-toe games, where ”x” is assumed to have played first. The target concept is ”win for x” (i.e., true when ”x” has one of 8 possible ways to create a ”three-in-a-row”). The dataset contains 958 instances without missing values, each with 9 attributes, corresponding to tic-tac-toe squares and taking on 1 of 3 possible values: ”x”, ”o”, and ”empty”.

The data set was randomly divided into training (90% of data) and evaluation (remaining 10% of data) sets. The algorithm was applied to the training set. The classifier was evaluated for predictive classification accuracy against the corresponding evaluation set. This process was repeated 8 times for the data set. In the program implementation of the algorithm we have used a tuning factor n which is the number of implicants that must cover the tested instance in order to classify it positive. This factor is brined in because in some cases the whole list of prime implicants may cover so many of the possible instances as the classifier decides that almost every testing sample is positive. For that purpose we increase the needed number of covering implicants and this way improve the correctness of the classifier. With this new possibility we have gained in experiments enviable accuracy of 0% error after tuning the factor n. Of course more testing is required with different data sets. But in practical domains this technique will be very useful to divide several times all available instances randomly 90/10 and find the most appropriate factor n that can be used afterwards for real classifying tasks on unknown cases.



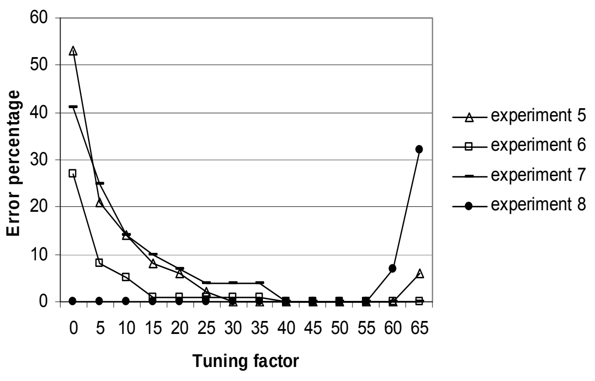


Figure 1. Experimental results

Essential advantage of the algorithm is its short time of performing. It finds prime implicants for tic-tac-toe data set in approximately 2.23min on a PC machine with 256 RAM and processor 846 MHz. Naturally the needed time highly depends on the particular data set.

CONCLUSIONS AND FUTURE WORK

In the paper a new effective algorithm for learning from examples is introduced. The approach aims at finding a more compact classification function that approximates target concept by corresponding to all positive but no one negative examples. The algorithm is based on fast strategy logical function minimization. Attribute-value language is used to represent examples as predicates. Each example is a conjunction of values corresponding to attributes. We consider target concept as a logical function with number of arguments equivalent to the number of attributes. The found prime implicants correspond to positive examples and does not correspond to negative ones, so they can successfully classify new unknown instances.

The presented approach to learning from examples solves some problems of existing methods. It has a larger concept space because of disjunctive extension. As we see from experimental results this method is efficient and fast.

.

Further work includes applying the approach to numerical and structured types of attributes. Another future task is to consider representation of got concepts in a decision tree and generating rules from it. Also extending the classifier to work for more than one class is possible, but this would increase the methods complexity. We aim at extending this machine learning concept to data mining.

REFERENCES

[1] Carpineto, C. Trading off consistency and efficiency in version-space induction. *Proceedings of Ninth International Machine Learning Conference,* Aberdeen, Scotland, pp. 43-48, 1992.

[2] Даковски, Л. Опростяване на превключвателни функции, зададени с голям брой неопределени набори на аргументите. *сп. Автоматика и изчислителна техника*, стр. 42-47. 1969.

[3] Dietterich, T., B. London, K. Clarkson, G. Dromey. Learning and inductive inference. *The Handbook of Artificial Intelligence, Volume III*. William Kaufmann, Los Altos, CA, 1982.

[4]Fensel, D., M. Wiese. From JoJo to Frog: Extending a bi-directional search strategy to a more flexible three- directional search. *In: Beiträge zum 7. Fachgruppentreffen Maschinelles Lernen, Kaiserslautern* 1994.

[5] Haussler, D.. Quantifying inductive bias: Ai learning algorithms and valiant's learning framework. *Artificial Intelligence*, 36:177–221, 1988.

[6] Hirsh, H.. Polynomial-time learning with version spaces. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 117–122, Menlo Park, CA, 1992.

[7] Hirsh, H., N. Mishra, and L. Pitt. Version spaces without boundary sets. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 491–496, Menlo Park, CA, 1997.

[8] Idemstam-Almquist. Demand networks: an alternative representation of version spaces. Master's thesis, Department of Computer Science and Systems Sciences, The Royal Institute of Technology and Stockholm University, Stockholm, Sweden, 1990.

[9] Merz, C., P. Murphy : UCI Repository of Machine Learning Databases [http://www.ics.uci.edu/ ~mlearn/MLRepository.html]. Department of Information and Computer Science, University of California, Irvine, CA (1998)

[10] Mitchell, T.. *Version spaces: an approach to concept learning*. PhD thesis, Electrical Engineering Dept.,Stanford University, Stanford, CA, 1978.

[11] Mitchell, T.. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.

[12] Mitchell, T.. *Machine learning*. McGraw-Hill, New York, NY, 1997.

[13] Sablon, G.. *Iterative versionspaces with an application in inductive logic programming*. PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium, 1995.

[14] Sebag, M., C. Rouveirol. Tractable induction and classification in first order logic via stochastic matching. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 888–893, San Francisco, CA, 1997.

[15] Sebag, M., C. Rouveirol. Resource-bounded relational reasoning: induction and deduction through stochastic matching. *Machine Learning*, 38(1-2):41–62, 2000.

[16] Smirnov, E.. *Conjunctive and disjunctive version spaces with instance-based boundary sets*. PhD thesis, Department of Computer Science, Maastricht Univeristy, Maastricht, The Netherlands, 2001.

[17] Smirnov, E., P.J. Braspenning. Version space learning with instance-based boundary sets. In *Proceedings of the Thirteenth European Conference on Artificial Intelligence (ECAI-98)*, pages 460–464, Chichester,UK, 1998.

[18] Smith, B., P.S. Rosenbloom. Incremental non-backtracking focusing: a polynomially bounded generalization algorithm for version spaces. In *Proceedings of the Eight National Conference on Artificial Intelligence (AAAI-90)*, pages 848–853. 1990.

[19] Winston, P.. *Artificial Intelligence*. Addison-Wesley, 3rd edition. 1992.

ABOUT THE AUTHORS

Ludmil Dakovski, D.S., Department of Computer Systems and Technologies, Technical University - Sofia, Phone: +359 02 9652414, E-mail: seven-in@bulinfo.net.

Zekie Shevked, mag.student at Computer Systems and Technologies, Technical University – Sofia, branch Plovdiv, E-mail: zekie_shevked@yahoo.com