The Iterated Version Space Algorithm

Howard J. Hamilton
Dept. of Computer Science
University of Regina
Regina, Saskatchewan, Canada, S4S 0A2
hamilton@cs.uregina.ca

Jian Zhang
Dept. of Computer Science
University of Regina
Regina, Saskatchewan, Canada, S4S 0A2
jian@cs.uregina.ca

Abstract

We present the Iterated Version Space Algorithm (IVSA), which retains many advantages of the Version Space Algorithm while handling disjunctive concepts and noise. IVSA repeatedly generates hypotheses for regions of the concept space and combines these regional hypotheses to produce an overall concept hypothesis. Experiments were conducted to learn English pronunciation rules from word-pronunciation pairs. The rules generated by IVSA produce highly accurate predictions for unseen words.

1 Introduction

We present the Iterated Version Space Algorithm (IVSA), which retains many advantages of the Version Space Algorithm (VSA) [Mit78] while handling disjunctive concepts and noise.

Learning a disjunctive concept is similar to assembling a jigsaw puzzle from a large set of possible pieces. The target concept can be viewed as the puzzle and an ordered list of disjuncts (called regional hypotheses (RH)), can be viewed as pieces of the puzzle. One method of solving this problem is to repeatedly generate candidate pieces and try adding them to the puzzle until it is complete. With IVSA, as shown in Figure 1, the Hypothesis Generator produces candidate RHs (the puzzle pieces) using VSA. Each RH covers some of input instances (part of the concept). The Hypothesis Assembler repeatedly picks the most promising RH (according to a simple ranking heuristic) and tries this RH in each position in a list of accepted RHs.

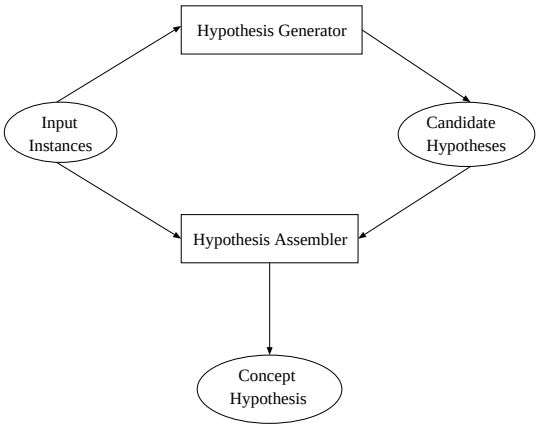

Figure 1: Overview of IVSA

If adding the RH increases the coverage of the concept, it is placed in the position that causes the greatest increase; otherwise the RH is discarded. After all candidate RHs have been examined, all accepted RHs in the list are examined to see if any can be removed without reducing accuracy. Then the process of hypothesis generation and assembly is repeated using, as input, the set of instances that are not yet covered. This process is repeated until further iterations cause no change in accuracy. The final list of accepted RHs is the concept description.

The remainder of this paper is organized as follows. In Section 2, we briefly describe version spaces and the VSA algorithm. We explain the IVSA algorithm in Section 3, and we give a descriptive example showing how IVSA discovers English pronunciation rules for graphemes in Section 4. In Section 5, we present the results of applying IVSA to learn pronunciation rules for one-syllable words. In Section 6, we discuss related research and in Section 7, we conclude.

2 Version Spaces

A *version space* [Mit82] is a representation of a concept which contains two sets: the G set, which contains the most general hypotheses consistent with all

examined instances, and the S set, which contains the most specific hypotheses consistent with all examined instances. Based on a series of positive and negative instances, the *version space algorithm* (VSA) constructs a version space. The initial general hypothesis matches everything, and the initial specific hypothesis matches only the first positive instance. If a complete and consistent set of instances of a single concept is provided, VSA converges to a single hypothesis. If the instance set is incomplete, S and G sets of hypotheses are produced. An accessible description of the VSA algorithm is given in [Win92].

VSA cannot handle *noisy* input instances [CF82], i.e., those not consistent with the existence of a single concept. Either the G set becomes overly specific or the S set becomes overly generalized. In either case, the affected set becomes empty, which indicates that no hypothesis exists that matches all instances. In practical problems, a few exceptions may be intermixed with many mutually consistent instances. In English pronunciation, the pronunciations of a grapheme in a few words contradict the majority pronunciation of the grapheme in similar contexts. Generally, a *grapheme* is a sequence of one or more letters pronounced as a single sound. For example, if the grapheme <a> is in a primarily stressed syllable which ends with a consonant, it is usually pronounced with the /æ/ sound, as in accent, access, and accident. However, in the words 'swan' and 'dwarf,' where the only syllable is primarily stressed and ends with a consonant, the grapheme <a> is pronounced with the /a/ and /open_o/ sounds, respectively. Even one such exception prevents VSA from identifying a concept.

VSA cannot discover disjunctive concepts. Continuing the example above, suppose a version space is constructed such that the G set contains only hypotheses where the conditions of being primarily stressed and ending with a consonant are present. If the next instance is <a> in accede, which is pronounced with the /æ/ sound but which is not in a primarily stressed syllable, the version space will be destroyed. This instance does not contradict the hypothesis that <a> is pronounced as /æ/ in such syllables; it contradicts the assumption that there is a single non-disjunctive hypothesis that describes when <a> is pronounced /æ/.

3 The IVSA Algorithm

In Figure 2, we present the IVSA algorithm. The main loop of IVSA is based on the overall classification accuracy of the result; if all instances are covered (i.e., correctly classified) or the accuracy does not change for max iterations, IVSA halts. First, us-

Input: I, a set of input instances
Output: H, an ordered list of regional hypotheses
$I' := I$, $H := \langle\ \rangle$, $Acc_0 := 0$
repeat for $i = 1, 2, \ldots$
  $CH :=$ Generator(I')
  $(H, Acc_i) :=$ Assembler(I, H, Acc_{i-1}, CH)
  $I' :=$ Incorrect(I, H)
until $I' = \phi$ or $(i > max$ **and** $Acc_i = Acc_{i-max})$

Figure 2: The IVSA Algorithm

Input: I', a set of uncovered input instances
Output: CH, a set of candidate regional hypotheses
$CH := \phi$
repeat for each possible decision value
  $X :=$ instances from I' classified neg or pos
  repeat until $X = \phi$
    $(S, G, x_m) :=$ VSA(X)
    $X := X - \{x_1, \ldots, x_{m-1}\}$
    $CH := CH \cup S \cup G$
  end
end

Figure 3: The Generator Algorithm

Input: I, a set of input instances;
    H, the initial list of accepted RHs;
    Acc, the initial classification accuracy;
    CH, a set of candidate RHs
Output: updated H and Acc
for each candidate hypothesis $h_i \in CH$
  $R_i := (|P_C| + |N_C|)\ /\ |I|$
end
for $h \in CH$, ordered by decreasing R values
  $BestAcc' := 0$
  for $j = 1, 2, \ldots |H| + 1$
    $H' := H$ with h inserted before position j
    $Acc' := |Correct(I, H')|\ /\ |I|$
    if $Acc' > BestAcc'$ **then**
      $BestH' := H'$, $BestAcc' := Acc'$
    end if
  end for
  if $BestAcc' > Acc$ **then**
    $H := BestH'$, $Acc := BestAcc'$
  end if
end for
/* Remove unnecessary hypotheses from H */
for $h \in H$
  $H' := H - \{h\}$
  $Acc' := |Correct(I, H')|\ /\ |I|$
  if $Acc' \geq Acc$ **then**
    $H := H'$, $Acc := Acc'$
  end if
end for

Figure 4: The Assembler Algorithm

ing the Generator algorithm, a set CH of candidate RHs is generated from the set I' of input instances that are not yet covered. Then, using the Assembler algorithm, the candidate RHs in CH are ranked and the best ones are inserted in a list H of accepted hypotheses. Finally, the input set is updated using the function called Incorrect, which returns the set of instances from I that are not yet covered by H.

The Generator algorithm (Figure 3) handles multiple values of the decision attribute. VSA requires that instances be classified as positive and negative. For each decision value, we create a set X of appropriate instances by marking all instances with this decision value as positive and all others as negative. (In our implementation, we actually use a variant of VSA called MVSA [HZ94], which obviates the need for this translation.) Each iteration of the inner loop generates multiple candidate RHs for a single decision value. Each RH is consistent with a series of instances. If an instance x_m is encountered that (according to VSA) would destroy the version space for X, the Generator instead stores the existing S and G sets and then begins constructing a new RH, using the instances beginning with x_m as input. Thus, each RH is constructed from mutually consistent instances. Finding inconsistent instances of a concept provides a natural way of separating regions of a concept to give a disjunctive concept hypothesis.

The Assembler algorithm (Figure 4) first ranks the candidate RHs produced by the Generator. Ranking is done according to measure $R = (|P_C|+|N_C|) \ / \ |I|$, where P_c is the set of positive instances that correctly match the hypothesis, N_c is the set of negative instances that correctly do not match the hypothesis, and I is the complete set of instances. R indicates the quality of an RH.

The Assembler then considers whether each candidate RH h should be added to the list H of accepted hypotheses, as follows. First h is added to H to form H'. If the classification accuracy is higher with H' than with H, then h is kept in the accepted list H. If other hypotheses are already present in H, then h is tested in all positions in the list, and the position resulting in the highest accuracy is selected. This process is repeated for each candidate RH. Given c candidate RHs and d accepted hypotheses, the Assembler performs $O(c(c+d))$ tests. Finally, the Assembler considers whether any accepted hypotheses could now be deleted because of other hypotheses that have been inserted in the list of accepted hypotheses.

4 A Descriptive Example

We now present an application of IVSA to learning rules for pronouncing English graphemes. For efficiency, we used MVSA [HZ94] instead of VSA. The ordered list of rules for pronouncing a single grapheme is treated as a single disjunctive concept. IVSA is applied separately for each grapheme. Here, we consider the grapheme <a>, which has very complicated pronunciation rules. The input instances are drawn from the NETtalk Corpus [SR88], which is an online pronouncing dictionary, and pronunciation is specified according to the International Phonetic Alphabet (IPA) [JG81]. Both the input instances and the regional hypotheses have 10 attributes (Table 4). For a RH, '?' may appear as a value for any attribute to indicate that no restriction is placed on its value.

Attr.	Purpose	Example Values
A_1	sound unit	/æ/, /gz/, /ei/
A_2	stress on a syllable	p, s, n
A_3	type of a syllable	open, closed
A_4	second grapheme before target	any grapheme
A_5	first grapheme before target	any grapheme
A_6	first grapheme after target	any grapheme
A_7	second grapheme after target	any grapheme
A_8	first grapheme of the syllable	any grapheme
A_9	last grapheme of the syllable	any grapheme
A_{10}	# of syllables	1, 2, 3, 4

Table 1: Attributes in Input Instances

Each input instance describes one occurrence, in a word, of the grapheme <a>, including its IPA sound symbol. In the NETtalk Corpus, the grapheme <a> happens to correspond to 10 different sound symbols (10 values for the decision attribute). A positive instance contains the target grapheme <a> and the target sound symbol, while a negative instance contains the target grapheme but not the target sound symbol. We let the sound symbol /æ/ be first learning target; i.e., instances with /æ/ are positive and instances with the other 9 sound symbols are negative. Two instances from the word *abacus* are:

(ash, p, open, empty, empty, b, a, a, a, 3)

(schwa, n, open, a, b, c, u, b, a, 3)

where 'ash' represents /æ/. While learning /æ/, the first instance is treated as positive and the second as negative.

We denote the individual general hypotheses as $G_1, G_2, \ldots, G_j$, according to the order in which they are generated by VSA. Similarly, we denote the individual specific hypotheses as $S_1, S_2, \ldots, S_k$. For example, in Figure 5, the initial G set contains one hy-

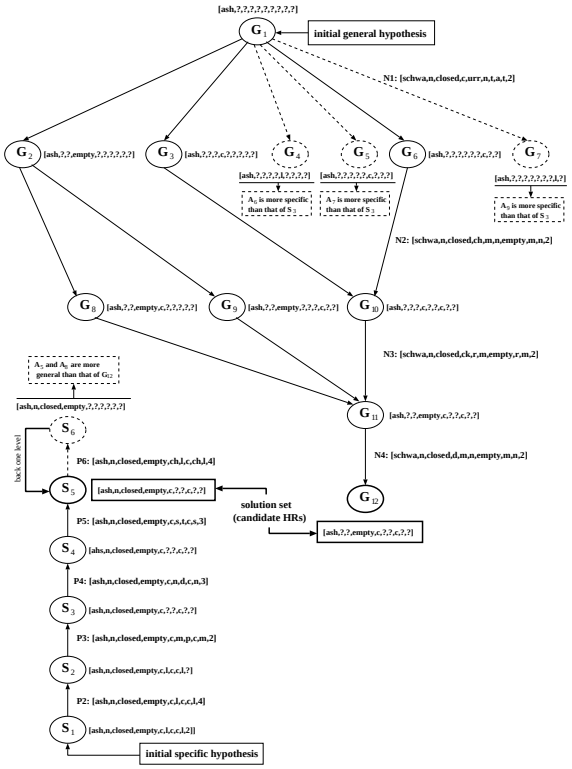

Figure 5: Hypothesis Space after One Iteration

pothesis G_1 = (ash, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?), and the initial S set contains S_1 = (ash, n, closed, empty, c, l, c, c, l, 2). Positive instances are denoted as $P_1, P_2, \ldots, P_p$, and negative ones as $N_1, N_2, \ldots, N_n$.

The Generator does not include all positive instances and exclude all negative instances in one step. Instead, an alternating sequence of positive and negative instances is examined by the MVSA algorithm. Figure 5 shows an example of one iteration of the inner loop of the Generator. When positive instance P_6 = (ash, n, closed, empty, ch, l, c, ch, l, 4) is examined, MVSA produces a new specific hypothesis S_6, (ash, n, closed, empty, ?, ?, ?, ?, ?, ?). Since the current general hypothesis G_{12} is (ash, ?, ?, empty, c, ?, ?, c, ?, ?), S_6 is overly generalized in A_5 and A_8. By the definition of MVSA, S_6 is pruned, which produces an empty S set. An empty hypothesis set indicates an inconsistency in the instances and MVSA (or VSA) ordinarily would find no solution. The key idea in IVSA is that instead of failing, we take the previous S set, which here is S_5, as a solution. Hypotheses S_5 and G_{12} correctly describe all input instances before P_6. Thus, S_5 and G_{12} are candidate RHs.

In Table 2, we show selected iterations of the IVSA algorithm for this problem. During the first iteration, 54 candidate RHs are generated. When the Assembler is applied to these RHs, the first candidate RH h_1 in RH results in an accuracy of 0.88% (not shown). Next h_2 is tested both before and after h_1. Since the classification accuracy does not increase, h_2 is discarded. When all 54 candidate RHs have been examined, the classification accuracy is 70.25% with 7 selected RHs in H. After the second iteration of IVSA, the accuracy is 72.46% and so forth as shown in Table 2. When no change in the accuracy of several iterations is detected, the learning process stops after 296 iterations with 112 RHs in H and a final accuracy of 99.56%.

i	$\|CH\|$	Rejected	Accepted	$\|H\|$	Acc_i
1	54	47	7	7	70.25%
2	9	5	4	11	72.46%
3	12	8	4	15	74.23%
4	16	14	2	17	74.52%
5	17	16	1	18	76.29%
6	16	15	1	19	76.44%
7	26	23	3	22	77.76%
8	13	11	2	24	85.13%
9	7	6	1	25	85.27%
10	7	7	0	25	85.27%
50	7	7	0	37	87.04%
100	7	6	1	54	90.72%
200	7	6	1	87	95.58%
296	7	7	0	112	99.56%

Table 2: Accuracy for Grapheme <a>

Rules Learned from 90% Input Instances		
Run	Correct Words	Correct Graphemes
1	96.51%	98.86%
2	95.16%	98.74%
3	97.85%	99.43%
4	95.16%	98.71%
5	96.24%	98.85%
6	93.01%	98.12%
7	96.24%	98.99%
8	94.64%	98.50%
9	95.71%	98.83%
10	95.98%	98.87%
Average	95.65%	98.79%

Table 3: Test Results on 10% Unseen Instances

5 Results

We implemented IVSA in Prolog and C and applied it to learn pronunciation rules for all graphemes in English. In our experiments for this paper, the input was 3,724 one-syllable words (from the NETtalk Corpus), which include 17,723 grapheme examples, 98 different graphemes and 46 different sound units. Results are shown in Table 3.

IVSA was applied to 90% of the input instances (positive and negative) and tested on the unseen 10% instances (positive and negative). For the experimental run k, the unseen instances were $\{x_j \mid j \ mod \ 10 = k\}$, i.e., $x_1, x_{11}, x_{21}, \ldots$ for run 1. Correct IPA pronunciations were produced for 93.01% to 97.85% of the unseen words and 98.12% to 99.43% of the graphemes in the unseen words. For example, if the word *abacus* is translated as /æ, b, ei, k, ∂, s/ instead of /æ, b, ∂, k, ∂, s/, then 5 out of 6 graphemes are translated correctly, but 0 words are translated correctly. On average, 269 rules were used more than once, and an additional 665 rules (perhaps representing exceptions) were used only once.

6 Related Research

Other version-space based approaches are given in [CR94] and [VB87]. Mitchell's *extended version space* approach first sorts the instances into groups, each consistent with a single disjunct in a disjunctive hypothesis, and then applies VSA to each group individually [Mit78]. That is, if the concept *parent* is disjunctively defined as *male and has_a_child* ∨ *female and has_a_child*, the first group of instances produces the hypothesis *male and has_a_child*, while the second group of instances produces the hypothesis *female and has_a_child*. Unfortunately, no method is provided for determining in advance which attribute should be used to preclassify the instances. This makes the approach impractical for problems with many attributes.

Mitchell also suggests a method for parallel learning with version spaces [Mit78], which learns from smaller subsets of input instances on several processors and then merges the results by intersecting the version spaces. Based on this approach, Hirsh developed a version space intersection approach called *Incremental Version Space Merging* (IVSA) [Hir94]. Given two sets of specific hypotheses S_i and S_{i+1}, IVSA repeatedly selects a pair (s_j, s_k) of hypotheses with s_j from S_i and s_k from S_{i+1}, then generalizes each pair to get the intersection set: $S_{i \cap i+1} = \{s_{j \cap k}, s_{j+1 \cap k+1}, \ldots, s_{j+n \cap k+n}\}$ This intersection process is repeated for both G and S until the last G and S hypotheses which were produced from individual version spaces. The IVSA approach removes the assumption of strict consistency between a version space and its input instances. Unfortunately, this approach cannot handle cases of true inconsistency, which are misclassified according to "near by" consistent instances.

7 Conclusion

We have presented the IVSA algorithm for concept learning. Generally speaking, IVSA inherits the advantages and reduces the major weaknesses of VSA. It selects regional hypotheses that cover groups of instances and uses them to form a concept hypothesis composed of a disjunctive list of regional hypotheses. IVSA has good immunity to noise, and test results show that IVSA can obtain high accuracy if applied to unseen instances.

References

[CF82] P.R. Cohen and E.A. Feigenbaum, editors. *The Handbook of Artificial Intelligence, Volume 3.* William Kaufmann, Los Altos, CA, 1982.

[CR94] M. J. Cavaretta and R. G. Reynolds. Discovering search heuristics for concept learning using version-space guided genetic algorithms. In *Proc. of the 7th Florida Artificial Intelligence Research Symposium (FLAIRS-94)*, pages 71–75, May, 1994.

[Hir94] Haym Hirsh. Generalizing version spaces. *Machine Learning*, 17:5–46, 1994.

[HZ94] H.J. Hamilton and J. Zhang. Learning pronunciation rules for English graphemes using the Version Space Algorithm. In *Proc. of Seventh Florida Artificial Intelligence Research Symposium (FLAIRS-94)*, pages 76–80, Pensacola Beach, FL, May 1994.

[JG81] Daniel Jones and A.C. Gimson. *Everyman's English Pronouncing Dictionary.* J.M. Dent, London, 1981.

[Mit78] T.M. Mitchell. *Version Spaces: An Approach to Concept Learning.* PhD thesis, Stanford University, CA, 1978.

[Mit82] T.M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[SR88] T. Sejnowski and C. Rosenberg. NETtalk corpus, (am6.tar.z). ftp.cognet.ucla.edu in pub/alexis, 1988.

[VB87] K. Vanlehn and W. Ball. A version space approach to learning context-free grammars. *Machine Learning*, 2:39–74, 1987.

[Win92] P.H. Winston. *Artificial Intelligence, Third Edition.* Addison-Wesley, Reading, MA, 1992.