2011 10th International Conference on Machine Learning and Applications

Unsupervised Grammar Induction of Clinical Report Sublanguage

Rohit J. Kate
Department of Health Informatics and Administration
Department of Computer Science
University of Wisconsin-Milwaukee
Email: katerj@uwm.edu

***Abstract*—Clinical reports are written using a subset of natural language while employing many domain-specific terms; such a language is also known as a sublanguage for a scientific or a technical domain. In this paper, we present a method which automatically induces a grammar for the sublanguage of a given genre of clinical reports from a corpus of reports with no annotations. The method first identifies the semantic classes of the clinical terms used in the reports, then it induces a grammar that is based on these semantic classes and part-of-speech tags. Experiments show that the induced grammar is able to parse novel sentences and obtains a reasonable accuracy.**

I. Introduction

Obtaining a syntactic parse is an important step in analyzing a sentence. Syntactic parsers are typically built using supervised learning methods. Several hundred or thousand sentences are first manually annotated with syntactic parses, then some learning method is employed that learns from this annotated data to parse novel sentences. A major drawback of this approach is that it requires a lot of manual effort from trained linguists to annotate sentences. Also, a parser trained in one domain does not do well on another domain without adapting it with extra annotations from the new domain. This drawback becomes even more severe when the domain is that of clinical reports or medical text, because the annotators need to be not only trained linguists but also need to have sufficient clinical knowledge to understand the clinical terms and the sentence forms. This is a rare combination of expertise which makes the annotation process for clinical reports even more expensive. Also, different genres of clinical reports, like discharge summaries, radiology notes, cardiology reports etc., are different from each other and hence will require separate annotations. On top of that, different hospitals or medical centers may be using their own convention of clinical terms and sentence styles in writing clinical reports which may require separate annotation effort to adapt a syntactic parser to work for clinical reports from a particular institution.

Besides the annotation effort required, another drawback of supervised syntactic parsing is that it forces a particular "gold-standard" of syntactic parses which may not be best suited for the end-application in which the syntactic parses will be used. For example, in the application of semantic parsing, the task of converting a sentence into an executable meaning representation, it was found that the conventional gold-standard syntactic parse trees were not always isomorphic with their semantic trees [1] which lowered the performance of semantic parsing. In the domain of clinical reports, where sentences are often succinct and may not follow typical English grammar, it is not easy to decide the gold-standard parses in advance. For example, a sentence like "Vitamin B12 250 mcg daily" could be parsed with brackets such as "((Vitamin B12 250 mcg) daily)" or such as "((Vitamin B12) (250 mcg daily))" depending upon whether the end-application emphasizes the "250 mcg" quantity with "Vitamin B12" or with "daily". However, during the annotation process, a particular form will get forced as part of the annotation convention without regards to what may be better suited for the end-application down the road. Syntactic parses are not an end in themselves but an intermediate form which is supposed to help an end-application, hence it will be best if such an intermediate form is not set in advance but gets decided based on the end-application.

An alternate to supervised learning for building parsers is unsupervised learning. In this framework, a large set of unannotated sentences, which are often easily obtainable, are given to an unsupervised learning method. Using some criteria or bias, for example, simplicity of grammar and corresponding sentence derivations, the method tries to induce a grammar that best fits all the sentences. Novel sentences are then parsed using this learned grammar. While unsupervised parsing methods are not as accurate as supervised methods, their almost no demand of manual supervision makes them an attractive alternative, especially for the domain of clinical reports for the reasons pointed out earlier.

An additional advantage of unsupervised parsing is that the grammar induction process itself may be adapted so as to do best on the end-application. For example, instead of using a simplicity bias to guide the grammar induction process, a criterion to maximize accuracy on the end-application may be used. This way the induced grammar may choose one parse over another for the "Vitamin B12 250 mcg daily" depending upon which way is more helpful for the end-application. In [2], an analogous approach was used to transform a semantic grammar to best suit the semantic parsing application.

In this paper, we present an approach for unsupervised grammar induction for clinical reports, which to our knowledge is the first such attempt. We adapt and extend the simplicity bias (or cost reduction) method [3] of unsupervised

978-0-7695-4607-0/11 $26.00 © 2011 IEEE
DOI 10.1109/ICMLA.2011.150

grammar induction. We chose to use this method because its iterative grammar-modifying process using grammar transformation operators is amenable to be adapted to any criterion besides simplicity bias. This could be useful for adapting the grammar induction process to maximally benefit some end-application. Another advantage of this method is that it directly gives the grammar in terms of non-terminals it creates on its own; some other existing methods only give bracketing [4], [5] or force the user to specify the number of non-terminals [6]. The induced grammar is also not restricted to binary form unlike in some previous methods [6], [7].

We used sentences from discharge summaries of the Pittsburgh corpus [8] as our unannotated data. Most unsupervised grammar induction methods work with part-of-speech tags because due to large vocabulary size it is difficult to directly induce grammars using the words themselves. Since the language used in clinical reports is a domain-specific sublanguage [9], [10], [11], it uses several terms, like disease names, medications etc., not generally found in normal language. This makes the vocabulary even larger. We also note that for parsing clinical reports, besides using part-of-speech tags, it will be a good idea to also use semantic classes of words because they often affect syntactic structure of a sentence. Hence we decided to also utilize UMLS semantic types of the clinical terms (for example, disease, sign or symptom, finding etc.), which, in a way, are treated like additional part-of-speech tags in the grammar induction process (Figure 1(d) shows an example). These semantic types and the part-of-speech tags are obtained using MetaMap [12]. The grammar is then learned in terms of part-of-speech tags and the semantic types of clinical terms.

In the experiments, we first show that the learned grammar is able to parse novel sentences. Measuring accuracy of parses obtained through an unsupervised parsing method is always challenging, because the parses obtained by the unsupervised method may be good in some way even though they may not match the correct parses. The ideal way to measure the performance of unsupervised parsing is to measure how well it helps in an end-application. However, at present, in order to measure the parsing accuracy, we annotated one hundred sentences with parsing brackets and measured how well they match against the brackets obtained when parsed with the induced grammar.

II. Cost Reduction Method for Grammar Induction

For inducing a context-free grammar from training sentences, we adapted the cost reduction method [3] which was based on Wolff's idea of language and data compression [13], also known as simplicity bias method or minimum description length method. The method starts with a large trivial grammar which has a separate production corresponding to each training sentence. It then heuristically searches for a smaller grammar as well as simpler sentence derivations by repeatedly applying grammar transformation operators of combining and merging non-terminals. The size of the grammar and derivations is measured in terms of their encoding cost. We extended this method in a couple of ways. We describe the method and our extensions in this section. We first describe how the cost is computed and then describe the search procedure that searches for the grammar that leads to the minimum cost.

The method uses ideas from information theory and views the grammar as a means to compress the description of the given set of unannotated training sentences. It measures the compression in terms of two types of costs. The first is the cost (in bits) of encoding the grammar itself. The second is the cost of encoding the sentence derivations using that grammar. In the following description we make use of some of the notations from [14].

A. Computing the Cost

1) Cost of Grammar: A production in a context-free grammar (CFG) is written in the form of $A \rightarrow \beta$, where A is a non-terminal and β is a non-empty sequence of terminals and non-terminals. The cost, C_P, of encoding this production is:

$$C_P = (1 + |\beta|)log|\Sigma| \quad (1)$$

where $|\beta|$ is the length of the right-hand-side (RHS) of the production, and $|\Sigma|$ is the number of terminals and non-terminals in the symbol set Σ. Since it will take $log|\Sigma|$ bits to encode each symbol and there are $(1 + |\beta|)$ symbols in the production (including left-hand-side (LHS)), hence the cost C_P of encoding the production is as given in the above equation. Thus the cost of encoding the entire grammar, C_G, is:

$$C_G = \Sigma_{i=1}^{p}((1 + |\beta_i|)log|\Sigma|) \quad (2)$$

where p is the number of productions and β_i is the RHS of the ith production.

2) Cost of Derivations: Given the grammar, a derivation of a sentence proceeds by first expanding the start symbol of the grammar with an appropriate production and then subsequently recursively expanding each of the RHS non-terminals till all the symbols of the sentence are found as a sequence of terminals. At every step in the derivation process, an appropriate production needs to be selected to expand a non-terminal. This is the only information that needs to be encoded in order to encode the sentence. Hence the information to be encoded at every step of the derivation is: which of the $|P(s_k)|$ productions was used to expand the kth non-terminal, s_k, in the derivation process, $P(s_k)$ being the set of productions in which s_k is the LHS. This information can be encoded in $log(|P(s_k)|)$ bits. For example, if there is only one way to expand a non-terminal then this information is obvious and would require zero bits to encode. Hence the cost of an entire derivation, C_{D_j} of the jth sentence will be:

$$C_{D_j} = \Sigma_{k=1}^{m_j}(log(|P(s_k)|)) \quad (3)$$

where m_j is the length of derivation of the jth sentence. Thus the cost, C_D, of encoding all q sentences in the training set is:

$$C_D = \Sigma_{j=1}^{q}\Sigma_{k=1}^{m_j}(log(|P(s_k)|)) \quad (4)$$

3) Total Cost: In previous work, like [3] and [14], the total cost of grammar and derivation was taken as simply the sum of the individual costs. However, as we show in the experiments, this does not always lead to good results. The reason, we believe, is that the total cost of derivations depends on the number of sentences and simply adding this cost to the grammar's cost may lead to an unequal weighting. To remedy this, we introduce a parameter, f, that takes value between 0 and 1, to separately weigh the two components of the total weight C as follows:

$$C = f * C_G + (1 - f) * C_D \tag{5}$$

where C_G is the cost of the grammar and C_D is the cost of all derivations as described before. Note that $f = 0.5$ is equivalent to adding the two components as in the previous work. In the experiments, we vary this parameter and empirically measure the performance.

B. Grammar Search for Minimum Cost

It is important to point out that there is a trade-off between the cost of the grammar and the cost of the derivations. At one extreme is a simplest grammar in which there are productions like $NT \rightarrow t_i$, i.e. a non-terminal NT that expands to every terminal t_i, and two more productions $S \rightarrow NT$ and $S \rightarrow SS$, (S being the start symbol) which will have a very little cost. However, this grammar will lead to very long and expensive derivations. It is also worth pointing out that this grammar is overly general and will parse any sequence of terminals.

On the other extreme is a grammar in which each production encodes an entire sentence from the training set, for example, $S \rightarrow w_1 w_2 .. w_n$, where w_1, w_2 etc. are words of a sentence. The derivations of this grammar will have very little cost, however, the grammar will be very expensive as it will have long productions and as many of them as the number of sentences. It is also worth pointing out that this grammar is overly specific and will not parse any other sentence besides the ones in the training set.

Hence the best grammar is in between the two extremes, which will be general enough to parse novel sentences but at the same time not too general to parse almost any sentence. This grammar will also have a smaller cost than either extreme. According to the minimum description length principle as well as Occam's razor principle, a grammar with minimum cost is likely to have the best generalization. We use the following search procedure to find the grammar which gives the minimum total cost where the total cost is defined as in equation 5. We note that by varying the value of the parameter f in that definition, the minimum cost search procedure can find different extremes of the grammars. For example, with $f = 1$, it will find the first type of extreme grammar with the least grammar cost, and with $f = 0$, it will find the second type of extreme grammar with the least derivation cost.

The search procedure begins with a trivial grammar which is similar to the second extreme type of grammar mentioned before. A separate production is included for each unique sentence in the training data. If the sentence is $w_1 w_2 .. w_n$, a production $S \rightarrow W_1 W_2 .. W_n$is included along with productions $W_1 \rightarrow w_1$, $W_2 \rightarrow w_2$, etc., where W_1, W_2, etc. are new non-terminals corresponding to the respective terminals w_1, w_2, etc. The new non-terminals are introduced because the grammar transformation operators described below do not directly work with terminals. Instances of the two grammar transformation operators described below are then applied in a sequence in a greedy manner, each time reducing the total cost. We first describe the two operators, *combine* and *merge*, and then describe the greedy procedure that applies them. While the *merge* operator is same as in [3], we have generalized the *combine* operator (which they called *create* operator). The search procedure is analogous to theirs but we first efficiently estimate the reduction in cost obtained by different instances of the operators and then apply the one which gives the most reduction in cost. They on the other hand do not estimate the cost but generate new grammars for every instance and then calculate the cost. They also follow separate loops of applying a series of merge and combine operators, but we follow only one loop for both the operators.

1) Combine Operator: This operator combines two or more non-terminals to form a new non-terminal. For example, if the non-terminals "DT ADJ NN" appear very often in the current grammar, then the cost (equivalently size) of the grammar can be reduced by introducing a new production $C1 \rightarrow DT\ ADJ\ NN$, where $C1$ is a system generated non-terminal. Next, all the occurrences of $DT\ ADJ\ NN$ on the RHS of the productions will be replaced by $C1$. As can be seen, this reduces the size of all those productions but at the same time adds a new production and a new non-terminal. In [3], the corresponding operator only combined two non-terminals at a time and could combine more than two non-terminals only upon multiple applications of the operator (for example, first combine DT and ADJ into C1 and then combine C1 and NN into C2). But we found this was less cost-effective in the search procedure than directly combining multiple non-terminals, hence we generalized the operator.

It may be noted that this operator only changes the cost of the grammar and not the cost of the derivation. This is so because in the derivations, the only change will be the application of the extra production (like $C1 \rightarrow DT\ ADJ\ NN$), and since there is only one way to expand the new non-terminal $C1$, there is no need to encode it (i.e. $|P(C1)|$ is 1, hence its log is zero in equation 4). It is also interesting to note that this operator does not increase the coverage of the grammar, i.e., the new grammar obtained after applying the *combine* operator will not be able to parse any new sentence that it could not parse before. The coverage does not decrease either.

The reduction in cost due to applying any instance of this operator can be estimated easily in terms of the number of non-terminals being combined and how many times they occur adjacently on the RHS of current productions in the grammar. Note that if the non-terminals do not appear adjacent enough number of times then this operator can, in fact, increase the cost.

(a) The patient was stable and started on heparin anticoagulation protocol for ventricular assist device.
(b) The patient was stable and started on **biologically_active_substance therapeutic_or_preventive_procedure** protocol for **medical_device**.
(c) det noun aux adj conj verb prep noun noun noun prep noun
(d) det noun aux adj conj verb prep **biologically_active_substance therapeutic_or_preventive_procedure** noun prep **medical_device**

Fig. 1. (a) An original sentence, (b) some of its words replaced by UMLS semantic types (bold), (c) its words replaced by part-of-speech tags, (d) its words replaced by part-of-speech tags and UMLS semantic types (bold) wherever applicable.

2) Merge Operator: This operator merges two non-terminals into one. For example, it may replace all instances of NNP and NNS non-terminals in the grammar by a new non-terminal $M1$. This operator is the same as in [3]; we did not generalize it to merging more than two non-terminals, because unlike the *combine* operator, it is combinatorially expensive to find the right combination of non-terminals to merge (for the *combine* operator, we describe this procedure in the next subsection).

The *merge* operator could eliminate some productions. For example, if there were two productions $NP \rightarrow DT\ NNP$ and $NP \rightarrow DT\ NNS$, then upon merging NNP and NNS into $M1$, both the productions reduce to the same production $NP \rightarrow DT\ M1$. This not only reduces the cost of the grammar by reducing its size, but also reduces the $|P(NP)|$ value (how many productions have NP on LHS) which results into a further decrease in the derivation cost (equation 4). However, if there were productions with NNP and NNS on the LHS, then their getting combined will make the cost of $|P(M1)|$ equal to the sum of $|P(NNP)|$ and $|P(NNS)|$ and replacing NNP and NNS by $M1$ everywhere in the derivations will increase the cost of the derivations.

To estimate the reduction in cost due to applying any instance of this operator, one needs to estimate which productions will get merged (hence eliminated) and in how many other productions does the non-terminal on the LHS of these productions appear on LHS. In our implementation, we efficiently do this by maintaining data structures relating non-terminals and the productions they appear in, and relating the productions and the derivations they appear in. We are not describing those details here due to lack of space. As mentioned before, while the cost may decrease for some reasons, it could also increase for other reasons. Hence an application of an instance of this operator can also increase the overall cost.

It is important to mention that application of this operator can only increase the coverage of the grammar. For example, given productions $NNS \rightarrow apple$, $VB \rightarrow eat$ and $VP \rightarrow VB\ NNP$, but not a production $VP \rightarrow VB\ NNS$, then "$eat\ apple$" cannot be parsed into into VP. However, merging NNP and NNS into $M1$ will result in new productions $M1 \rightarrow apple$ and $VP \rightarrow VB\ M1$ which will parse "$eat\ apple$" into VP. Hence this operator generalizes the grammar.

3) Search Procedure: Our method follows a greedy search procedure to find the grammar which results in the minimum overall cost of the grammar and the derivations (equation 5). Given a set of unannotated training sentences, it starts with the trivial, overly specific, extreme type of grammar in which a production is included for each unique sentence in the training set, as mentioned before. Next, all applicable instances of both the *combine* and *merge* operators are considered and the reduction in cost upon applying them is estimated. The instance of the operator which results into the greatest cost reduction is then applied. This process continues iteratively until no instance of the operator results in decrease in cost. The resultant grammar is then returned as the induced grammar.

In order to find all the applicable instances of the *combine* operator, all "n-grams" of the non-terminals on the RHS are considered (the maximum value of n was 4 in the experiments). There is no reason to consider an exponentially large number of every combination of non-terminals which do not even appear once in the grammar. However, in order to find all the applicable instances of the *merge* operator, there is no such simple way but to consider merging every two non-terminals in the grammar (it is not obvious that any other way will be significantly more efficient with regards to estimating the reductions in cost). The start symbol of the grammar is preserved and is not merged with any other symbol.

Note that this search procedure is greedy and may only give an approximate solution which could be a local minima.

III. Experiments

A. Methodology

We took the first 5000 sentences from the discharge summaries section of the Pittsburgh corpus [8] using Stanford CoreNLP's[1] sentence segmentation utility. We ran MetaMap [12] on these sentences to get part-of-speech tags and UMLS semantic types of words and phrases. MetaMap appeared to run endlessly on some long sentences hence we restricted to sentences with maximum length 20 (i.e. all 5000 sentences were of maximum length 20). Since many UMLS semantic types seemed very fine grained, we chose only 27 of them which seemed relevant for clinical reports. All the occurrences of these semantic types were substituted for the actual words and phrases in the sentence. Figure 1(a) shows an original sentence from the corpus and 1(b) shows the same sentence in which some words and phrases have been substituted by their UMLS semantic types. Figure 1(c) shows the part-of-speech tags of the words of the original sentence as obtained by MetaMap. Note that the part-of-speech tags that MetaMap outputs are not as fine-grained as in the Penn treebank. Also note that the entire last phrase "ventricular assist device" is tagged as a single noun. Finally, the words which were not

[1] http://nlp.stanford.edu/software/corenlp.shtml

replaced by the chosen UMLS semantic types were replaced by their part-of-speech tags. Figure 1(d) shows the original sentence from 1(a) with words and phrases replaced by part-of-speech tags and UMLS semantic types. We ran all our experiments on sentences transformed into this final form. We note that our experiments are obviously limited due to the accuracy of MetaMap in determining the correct part-of-speech tags and UMLS semantic types.

We separated 1000 sentences from the 5000 sentences and used them as test sentences to determine how well the induced grammar works on novel sentences. The rest were used as the training data to induce grammar. Out of these 1000 test sentences, we manually put correct parsing brackets on 100 sentences to test the performance of the parses obtained by the induced grammar. We are not aware of any annotated corpus of clinical report domain which we could have used to measure this performance.

B. Results and Discussion

We first show that by varying the parameter f of the total cost (equation 5), which weighs the relative contribution of the cost of grammar and the cost of derivations, the grammar induction method is capable of inducing a range of grammars, from very general ones to very restrictive ones. In this experiment, we only considered sentences which have at most 10 words for both training and test sentences (80% of the sentences in the corpus were of maximum length 10). We later show that performance decreases as we increase the maximum length of the sentence. We apply the method of inducing grammar on 2000 training sentences (we later show that more training sentences did not improve results) and measure how many novel sentences in the test data (i.e. which are not same as any of the training sentences) were parsable using the induced grammar.[2] Out of the 791 sentences in the test data with maximum length of 10, 554 sentences overlapped with the training sentences (clinical reports have many repeated sentences). The remaining 237 sentences were novel. In Figure 2, we plot what percentage of these novel sentences the induced grammar was able to parse as we varied the f parameter from 0 to 0.5. The parses were obtained using Earley's context-free grammar parsing algorithm [15]. If f is more than 0.5, the induced grammar becomes so restrictive that it never parses any novel sentence. It can be seen that with smaller f value, the induction method tries to minimize the cost of the grammar more than the cost of the derivations and hence comes up with a small grammar that is very general and is able to parse almost any sentence. But with larger f values, the induction method tries to minimize the cost of the derivation more and comes up with a large grammar which is not very different from a production for every training sentence, and hence cannot parse many novel sentences. In this experiment we disallowed learning recursive productions

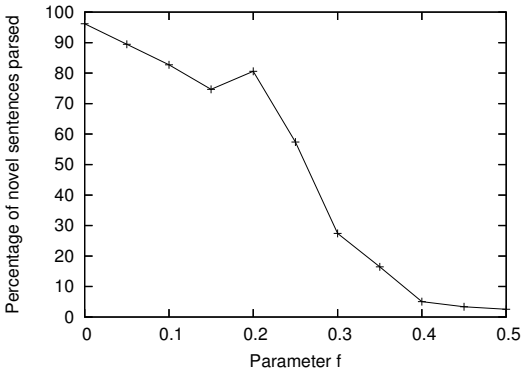

Fig. 2. Percentage of novel sentences parsed by different grammars induced by minimizing the total cost of the grammar and the derivations as the value of parameter f, which relatively weighs the two costs, is varied.

by making sure a grammar transformation operator does not lead to a recursive production. We did so because we found that by allowing recursive productions, the percentage of novel parses go from near 100% down to near 0% with nothing in between.[3]

Please note that being able to parse a sentence does not necessarily mean that the parse is correct; we show this accuracy in Figure 3. Out of the 100 sentences that we manually annotated with correct parse brackets, 70 sentences were of maximum 10 words. We measured *precision*, how many of the brackets (start and end word) in the parses obtained by the induced grammar were present in the correct parses, *recall*, how many of the correct brackets were present in the obtained parses, and *F-measure*, the harmonic mean of precision and recall. We did not consider brackets containing one word and the entire sentence since they will be always correct. We measured these on novel sentences as well as the sentences which are also present in the training data because the system is not given the correct parses of the training sentences and so it makes sense to also measure the parsing performance on them. We measure the performance of bracketing because labeling accuracy cannot be measured since the system can only come-up with system-generated labels (like $M1$, $C1$ etc.). As it can be seen from Figure 3, the precision increases with higher values of f (more restrictive grammars) but the recall overall decreases. The F-measure is found to be maximum in between (40.1% at value 0.2).

Earlier, in the above results, the F-measure used to be maximum around 30%. Upon error analysis, we noticed that most of the errors were because the induced grammar would incorrectly pair subject and verb instead of traditional way of pairing verb and object. There were similar errors of pairing nouns followed by a preposition. Similar errors have been reported previously [6]. These errors could either be because the search for an optimum grammar is only approximate or it

[2]Since the grammar induction process starts with the grammar that can parse the training sentences and the grammar transformation operators never reduce its coverage, the induced grammar will always parse the sentences which are present in the training set.

[3]Recursions in productions drastically reduce the size of the grammar, hence the process otherwise prefers recursive productions which often leads to small overly general grammars (similar to an example given at the beginning of Subsection II-B).

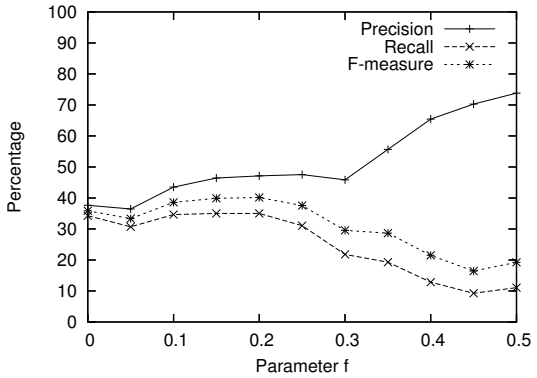

Fig. 3. Precision, recall and F-measure of the parsing brackets while varying the value of parameter f which relatively weighs the cost of the grammar and cost of the derivations during the grammar induction process.

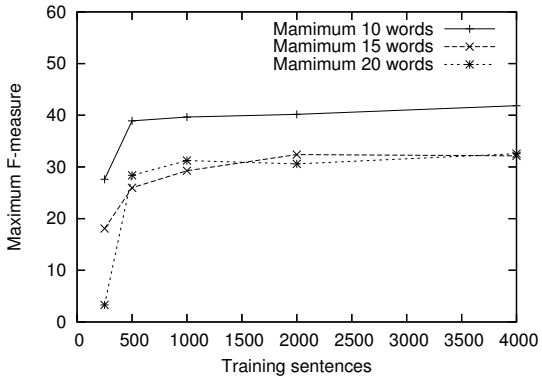

Fig. 4. F-measure of the parsing brackets with different amounts of training data and different maximum length of sentences.

could be because these are in fact reasonable alternate parses. Nevertheless, in order to see its effect we introduced hard rules in the system to never let the part-of-speech tags of verb, det, prep, aux and conj be the second or later RHS term in any production. This increased the F-measure. But as can be seen from the wide range of F-measures in Figure 3, these rules alone are not sufficient to guarantee good performance. In future, perhaps these biases could be learned from a small amount of supervised data in a semi-supervised grammar induction setting.

Finally, we present results on increasing the size of the training data and increasing the maximum length of the sentences (for both training and test sets) in Figure 4. The performance was measured on the same bracket-annotated corpus of 100 sentences all of which were of maximum length 20, 70 of these sentences had maximum length of 10 and 92 had maximum length of 15. For each point in the graph, we are showing the maximum F-measure that was found upon varying the parameter f. As can be seen, the accuracy decreases with longer lengths of sentences. It is interesting to note that the performance seems to have plateaued with the number of training examples.

IV. Future Work

An obvious future task is to apply this approach to other genres of clinical report present in the Pittsburgh corpus. We, in fact, already did this, except for manually creating a corresponding bracket-annotated corpus for measuring parsing performance. Also, a bigger annotated corpus for evaluating the current results on discharge summaries genre is desirable. Another avenue of future work is to improve the search procedure for finding the optimum grammar. One way would be to do a beam search. Besides using the UMLS semantic types, in future, one may decide additional semantic types which could help in some application, for example, a negation class of words, a class of words representing patients etc.

V. Conclusions

We presented an approach for inducing grammar for clinical report sublanguage in terms of part-of-speech tags and UMLS semantic types. We showed that using the cost-reduction principle, the approach is capable of learning a range of grammars from very specific to very general and achieves the best parsing performance in between.

References

[1] R. Ge and R. J. Mooney, "Learning a compositional semantic parser using an existing syntactic parser," in *Proc. of ACL-IJCNLP*, 2009, pp. 611–619.

[2] R. J. Kate, "Transforming meaning representation grammars to improve semantic parsing," in *Proc. of CoNLL*, 2008, pp. 33–40.

[3] P. Langley and S. Stromsten, "Learning context-free gramamr with a simplicity bias," in *Proc, of ECML*, 2000, pp. 220–228.

[4] Y. Seginer, "Fast unsupervised incremental parsing," in *Proc. of ACL*, 2007, pp. 384–391.

[5] E. Ponvert, J. Baldridge, and K. Erk, "Simple unsupervised grammar induction from raw text with cascaded finite state models," in *Proc. of ACL-HLT*, 2011, pp. 1077–1086.

[6] D. Klein and C. Manning, "A generative constituent-context model for improved grammar induction," in *Proc. of ACL*, 2002, pp. 128–135.

[7] D. Klein and C. D. Manning, "Corpus-based induction of syntactic structure: Models of dependency and constituency," in *Proc. of ACL*, 2004, pp. 479–486.

[8] W. W. Chapman, M. Saul, J. Houston, J. Irwin, D. Mowery, H. Karkeme, and M. Becich, "Creation of a repository of automatically de-identified clinical reports: Processes, people, and permission," in *AMIA Summit on Clinical Research Informatics*, San Francisco, CA, March 2011.

[9] Z. Harris and et al., *The Form of Information in Science: Analysis of an Immunology Sublanguage*. Kluwer Academic, 1989.

[10] N. Sager, C. Friedman, and M. S. Lyman, *Medical Language Processing: Computer Management of Narrative Data*. Reading, MA: Addison-Wesley, 1987.

[11] C. Friedman, P. O. Alderson, J. H. M. Austin, J. Cimino, and S. B. Johnson, "A general natural language text processor for clinical radiology," *Journal of the Americal Medical Informatics Association*, vol. 2, pp. 161–174, 1994.

[12] A. R. Aronson and F. Lang, "An overview of MetaMap: Historical perspectives and recent advances," *Journal of the Americal Medical Informatics Association*, vol. 17, pp. 229–236, 2010.

[13] J. G. Wolff, "Language acquisition, data compression, and generalization," *Language and Communication*, vol. 2, pp. 57–89, 1982.

[14] T. Chen, C. Tseng, and C. Chen, "Automatic learning of context-free grammar," in *Proc. of Conference on Computational Linguistics and Speech Processing*, 2006.

[15] J. Earley, "An efficient context-free parsing algorithm," *Communications of the Association for Computing Machinery*, vol. 6, no. 8, pp. 451–455, 1970.