I Know Your Name: Named Entity Recognition and Structural Parsing

David Philipson and Nikil Viswanathan
{pdavid2, nikil}@stanford.edu
CS224N Fall 2011

Introduction

In this project, we explore a Maximum Entropy Markov Model (MEMM) for named entity recognition and a probabilistic context free grammar (PCFG) parser for learning the structure of a sentence. We apply our algorithms to classify the biological and cellular entities in the GENIA and BioIE biological data sets. Through careful feature engineering and markov verticalization we demonstrate improved performance performance of these systems ultimately achieving an F1 score of 84.01 for our final parser.

Methods

In this section we will discuss the theoretical models and implementation techniques we used for the Maximum Entropy Classifier and the PCFG Parser. Along with our proofs of correctness we investigate our results and analyze the successes and failures of our models.

Maximum Entropy Markov Model

As discussed in the assignment handout, our model uses gradient descent to optimize on the objective function of the negative conditional log likelihood of the training data (plus an additional term for smoothing. To compute the probability, we must compute (as given in the handout):

$$p(c \mid d;\lambda) = \frac{exp\left[\sum_i \lambda_i f_i(c,d)\right]}{\sum_{c'} exp\left[\sum_i \lambda_i f_i(c',d)\right]}$$

To see that this is correct, note that the numerator and denominator are both positive, and further that we have chosen the denominator so that if we sum this expression over all classes *c'*, then the numerator and denominator will be equal, hence the probabilities sum to 1. However, what we are actually interested in is the log-probability (for numerical precision issues), in which case the numerator is just the sum but the denominator is the log of the sum of exponentials. Happily, we can compute this efficiently using the provided LogSum function, which computes the log of a sum given the sum of its logs. Hence, we compute the numerator for each category c', then call LogSum on these values in order to get the denominator.

We must also compute the derivatives, which are given by the formula in the assignment. We

note that the derivative $\frac{\partial F(\lambda)}{\partial \lambda_i}$ consists of two summations- as discussed in the handout, the first sum is the number of times the feature *i* occurs with class *c* in the training, while the second is the expectation of this probability according to the probabilities computed by the model (as discussed above). We can efficiently compute both for all parameters λ_i with a single pass through the training data. To do so, for each training datum (*c*, *d*), we increment the first summation (the count of observed instances) by 1 for all parameters corresponding to class *c* and whose features are active in *d*. For this datum, we also iterate through with each possible classification *c'*, incrementing the second count by $p(c' \mid d; \lambda)$ for the parameters corresponding to *c'* and whose feature is on in *d*. This allows us to compute the computations with a single pass through the data.

Feature Engineering

We experimented with several different types of features honed for the GENIA data set. Initially we started out with a base set of features that we thought would be useful after glancing at the data and briefly exploring the language.

Our initial features were:

- Word Length
- Word Contains a Digit
- Word Contains a Hyphen
- Word is Capitalized
- Word Contains All Capital Letters
- Word Contains Delayed Capitals (Capital letters not in the beginning)

A number of these seemed like features which might be relevant in some general sense for any sort of classification, such as length or capitalization. Others were targeted for specific patterns we observed in the data. For instance, there were a number of named entities which were entirely capitalized acronyms, such as RTB. Others had distinctive patterns involving hyphens and numbers, such as the protein VCAM-1. Finally, one distinctive pattern which we did not notice appearing anywhere other than named entities was capital letters appearing later in a word after lower-case letters, such as the protein NFkappaB. We threw these features together for our first attempt.

Our next attempt was to look for “keywords” in nearby words. Looking through our results, we noticed that several phrases such as “leukaemic cells,” were not being correctly labeled as cell types, despite being obvious to us humans. To catch such cases, we added new features to look for “keywords” in nearby words. For example, this example could be correctly categorized by observing the presence of the word “cells.” Hence, we added keywords of “cell” and “protein” as features.

For each of these words, we added features to check

- Whether the current word contains this keyword

Wither the previous word contains this keyword
Whether the next word contains this keyword

After this point, we were getting an FB1 of 43.28. Trying to improve the score, we decided to look into the individual entities and try to build features to target each of them individually. Digging into the details, we noticed that the RNA had a classification precision and recall of 0% (while our classification precision for the other entities was above 30%) so we decided that this was where we should focus our attention.

Features for RNA

Our first strategy was to look at the output of the NER classifier and try to eyeball patterns in the data. We noticed that the RNA frequently had either a left parenthesis, right parenthesis or both in the data. Adding this feature increased bumped the RNA precision and recall up to 7.68% and 3.05% respectively and interestingly slightly decreased the overall FBI from 43.28 to 43.22. We realized that the parenthesis feature probably probably applied to many of the many of the other entities so it might not be discriminative enough and even if the other entities contained parenthesis, RNA could be the most likely to have this feature so other entities which also have it get overwhelmed by this feature weight for RNA.

Now to make sure we found features which applied well to the RNA we looked at the words that the gold standard classified as RNA attempted to figure out what they shared in common. After looking the words tagged as RNA in the ground truth data, we quickly noticed that the string “mRNA” very often showed up showed up sometimes before, sometimes after and sometimes in the middle of a sequence of two or more words tagged as the RNA. Adding the string mRNA to the keyword features we had above for “cell” and “protein” immediately boosted the RNA precision to 70.11%, recall to 46.56% and FB1 to 55.96%. With this one change we achieved a 60% boost in precision in RNA detection.

Now that our RNA entity identification and recognition was better than any of our other entity classifications we looked at optimizing the others. Looking at the overall FB1 score, we were surprised to find out that it had only increased by a .74 even though our RNA had jumped to over a 55.96 FB1 score. Our main and only decrease in classification had been in the cell line which had an approximately 5 point FB1 score decrease.

We hypothesized that that we had added features that mainly identified RNA in the training data and to a lesser degree cell lines, which were getting overridden by the RNA likelihood in the training data. In an attempt to fix this, we tried to limit the extent to which we matched against the mRNA string by now only checking to see if it was an exact match in the subsequent word instead of a substring match in all neighboring words. This guess actually proved to be right and removing the extra features improved our FB1 score slightly. After trying the different keyword substring combinations, we found that the best feature strategy was to only have the exact match for the string mRNA.

After greatly increasing the RNA performance, we still only saw a moderate increase in the

overall named entity performance and upon further investigation realized that RNA was a much smaller subset of the total number of entities when compared to the much more common other entities such as proteins. After optimizing RNA we then turned our efforts to optimizing the score for the largest entity, proteins, with similar techniques and also focusing more on the structure of the protein entities instead of the actual letters.

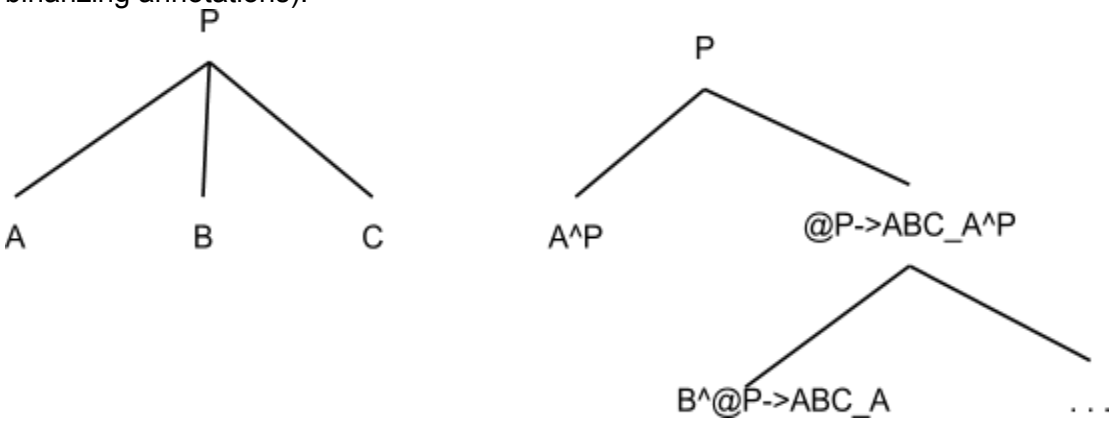
We often usually found that adding specialized features increased our score only slightly overall and that often individual entity skyrocketed in performance suggesting that a balancing act was needed.

Probabilistic Context Free Grammar Parser

Our parser is a direct implementation of CKY algorithm. Perhaps the only important decision in our implementation was using back-pointers to reconstruct the best parse after dynamic programming is complete- this seemed like the natural choice once we found (unsurprisingly) that we had enough memory to do so.

Second-order vertical markovization is implemented by recursively stepping through the tree, appending ^L to the label of each node we visited, where L is the (original) label of its parent. This process is easily reversed by cutting off the part of a label which appears at and past the first ^ character.

At this step, when annotating a tree we must binarize and we may markovize, and we have a choice of which order to perform the two. We tried both orders and discovered that binarizing and then markovizing gave us a gain of about 7% on our F1 score compared to just binarizing, indicating that vertical markovization was a success. On the other hand, markovizing and then binarizing actually gave us a decrease of about 4% to our F1 score. It makes some sense that markovizing and then binarizing would be more successful: to see why, consider the following diagram, in which the right-hand tree is the transformation of the left-hand tree by binarizing and then markovizing. Observe that the B node is annotated in such a way that we can read that it is the second node out of an series of A, B, C nodes generated by a P node. In effect, this gives us some level of horizontal markovization. By contrast, if we markovize and then binarize, we do not get this effect (and in fact some of the markov information is simply repeated by the binarizing annotations).



We also implemented third-order vertical markovization. With our implementation of second-order markovization, this was easy: we simply ran the function for second-order markovization

twice in a row. To verify correctness, consider the case where we have notes labled $A \rightarrow B \rightarrow C$ in the tree. In this case, after running our markovization function once, the latter two nodes become *B^A* and *C^B*. Then after running our markovization function a second time, the last node becomes *C^B^B^A*. The double-*B* in the middle might look odd, but as this notation appears throughout the entire tree, we might consider this simply an eccentric notation for what would normally be written *C^B^A*.

PCFG Results

Our CKY parser with 2nd order markovization obtained an F1 score of 84.01 and in general parsed most of the sentences with a reasonable parse even when it did not match the gold standard.

For example, consider the parse:

```
Guess:
(ROOT
  (S
    (NP
      (NP
        (NP (NN Identification))
        (CC and)
        (NP (NN cloning)))
      (PP (IN of)
        (NP
          (NP (NN TCF-1))
          (, ,)
          (NP (DT a)
            (ADJP (NN T) (JJ lymphocyte-specific))
            (NN transcription) (NN factor)))))
    (VP (VBG containing)
      (NP (DT a) (JJ sequence-specific) (NN HMG) (NN box)))
    (. .)))

Gold:
(NP
  (NP
    (NP (NN Identification))
    (CC and)
    (NP (NN cloning)))
  (PP (IN of)
    (NP
      (NP (NN TCF-1))
      (, ,)
      (NP
        (NP (DT a)
```

(ADJP (NN T) (JJ lymphocyte-specific))
(NN transcription) (NN factor))
(VP (VBG containing)
(NP (DT a) (JJ sequence-specific) (NN HMG) (NN box))))))
(. .))

With a score of: P: 66.67 R: 66.67 F1: 66.67 EX: 0.00

Our parser classified the training example as a sentence, while it is a dependent clause, and attached the verb phrase to the beginning of the clause, both of which are reasonable interpretations. Especially looking at the verb phrase attachment decsion, we can see that the specific interpretation of the sentence greatly depends upon the context of the material and is open to ambiguity even by humans.

Parser Results

	Precision	Recall	F1	EX
CKY	78.96	76.22	77.57	21.31
2nd Order Vertical Markovization then Binarization	74.61	73.16	73.88	19.67
Binarization then 2nd Order VM	86.69	81.50	84.01	32.79
3rd Order VM then Binarization	72.50	51.32	60.10	21.31
Binarization then 3rd Order VM	84.17	82.06	83.10	31.15

NER and Parser Combination

We observed the several of the chunked entities which were passed through our parser to see whether this improved the behavior of the parser. Overall we found that in many cases, our chunked results often occurred in conjunction with parses that were 100% correct even without NER. We also noticed that the chunked entities, which were all supposed to be noun phrases, were almost always correctly replacing a set of nouns or noun phrases in the gold standard parse.

Chunked Parse Guess:
(ROOT

```
 (S
  (NP
   (NP (DT These) (NNS changes))
   (PP (IN in)
    (NP (DT the) (NN enhancer) (NN element))))
  (VP (VBD were)
   (VP (VBN identified)
    (PP (IN in)
     (NP (DT the) (JJ original) (NN AL1_virus_stock)))))
  (. .)))
```

Gold (Unchunked) Standard:

```
(S
 (NP
  (NP (DT These) (NNS changes))
  (PP (IN in)
   (NP (DT the) (NN enhancer) (NN element))))
 (VP (VBD were)
  (VP (VBN identified)
   (PP (IN in)
    (NP (DT the) (JJ original) (NN AL1) (NN virus) (NN stock)))))
 (. .))
```

We can see that in this sentence the three nouns “AL1”, “virus”, and “stock” all were successfully combined by the NER algorithm into one phrase.

Occasionally we saw an adjective and several subsequent nouns get combined by the NER chunking.

```
(ROOT
 (NP
  (NP
   (ADJP
    (ADJP (JJ Positive))
    (CC and)
    (ADJP (JJ negative)))
   (NN regulation))
  (PP (IN of)
   (NP (NN immunoglobulin) (NN gene) (NN expression)))
  (PP (IN by)
   (NP (DT a) (JJ novel) (NN B-cell-specific_enhancer_element)))
  (. .)))
```

Gold:

```
(NP
 (NP
  (ADJP
   (ADJP (JJ Positive))
   (CC and)
   (ADJP (JJ negative)))
  (NN regulation))
```

(PP (IN of)
(NP (NN immunoglobulin) (NN gene) (NN expression)))
(PP (IN by)
(NP (DT a) (JJ novel) *(JJ B-cell-specific) (NN enhancer) (NN element)*))
(. .))

This phrase could be interpreted as a compound noun by a human so the chunker's still produced a reasonable parsing. All in all, we saw that chunking with our markov NER system before parsing often provided surprisingly good results, however in many of these cases, we noticed that the original parser often got the parse correct in the first place without even using the NER system before.

Summary of Results

Our parser by itself had a F1 score of 84 when using the CKY method with second order vertical markovization. Adding the named entity chunking often resulted in sensible parses, many times in cases which the original parser had already gotten correct.