Bayesian Word Sense Disambiguation and Spelling Correction

Ido Milstein
04730545

April 12, 2000

1 Introduction

Several approaches to word sense disambiguation were suggested in class. I decided to implement the naive bayes algorithm [1], tested it and made several modifications to it. The modifications take place in two areas. In the way the learned probability distribution is smoothed, and in the learning and use of the probabilities $P(Word|Sense)$. Both modifications will be discussed later. I used the exact same methods for both word sense disambiguation, and spelling correction. The slight differences in code between the two is written in the README.txt file.

2 Implementation

I implemented the project in C++, with support of STL. The code is fairly simple, and regularly commented. One file is used for each part of the project. It contains both the training algorithm, and the disambiguation algorithm. Both of these are very similar, but I chose to duplicate the code because of two reasons: clarity, and more importantly, in some experiments I changed the training and disambiguation in non symmetric ways, and duplicating the code makes this easier.

3 The Naive Bayes Algorithm

The Naive Bayes algorithm, finds the word sense that maximizes the posterior probability $P(Sense|Context)$ using the assumption that the context is made of independently distributed words given the word sense. So for this matter the algorithm treats $P(Sense_i)$ and $P(Word_j|Sense_i)$ as sufficient statistics to represent the training data, and the disambiguation context. This is a very strong assumption to make because of several reasons.

First, the assumption that words in a phrase are independent of each other is far from true, both Semantically and Syntactically. Semantically, phrases usually have some general idea or topic that relates the words in them, so a word like “chocolate” would usually not

appear in the same phrase as the word "dog". The dependency on the word sense, that we have in the probabilities $P(Word_j|Sense_i)$ compensates for some portion of this effect, but not for all of it. Syntactically the assumption fails, because some combinations of words will never appear simply because they are wrong, like "the to". However, it seems to me that ignoring the syntax in this way, is not so bad for remote word, because the syntax of words that are remote from our ambiguous word gives us little or no information about the word. This may not be such a major problem to the Naive Bayes algorithm, because it ignores the order and syntax of the words in the phrase altogether. But, this brings me to the next point.

The assumption that the *local* syntax of the word has little influence over it is quite wrong. This is evident especially when the different senses are of different parts of speech. For example the if word "the" appears before our word, it completely eliminates the chance that the word is actually a verb.

The last major fault in this algorithm, which is more of the same but from a different point of view, is that it assumes that words are equally *semantically* related to the ambiguous word even when they are far from it. It actually assumes that whoever picked the context window for the training and/or disambiguation was very precise in selecting the relevant window size. This does not remain true when in real world problems we need the computer to select the context window. If we add to this the fact that the probabilities that we learn are quite noisy anyhow, we get that remote words may many times introduce more noise than information.

It is these last two faults that I tried to address in my modifications, and I will elaborate on that later.

4 Experimenting with Naive Bayes

Having thought about all these reasons as to why Naive Bayes shouldn't work properly, I was quite surprised to see that it actually performed rather well on most cases, scoring often $10-30\%$ above the baseline. However, the problems I discussed above manifested themselves in some of its failiors, and some other problems also showed up. Here's a selective part of them, taken from the word sense disambiguation data.

The first misidentification I came across was rather peculiar, it was one the "drug" examples that I myself couldn't disambiguate, probably because of a context that was too short. This shows the problem in deciding what size of window we should use.

Many other misidentifications resulted from supposedly rare instances where key words for once sense appeared in the context of another. For example, one phrase in the data spoke about "tobacco and drug companies". The word "tobacco" gave a very strong priming to "narcotic drugs", and this caused the mistake.

A good example for problems that occur when syntax is disregarded, showed on the "plant" data, where verb/noun were misidentified. The reason for this is obvious - both the verb and the noun are likely to appear in contexts that share the same words. The key for disambiguation in a case like this is mainly the local syntax.

This problem also showed in cases where words had different meanings that were used in same contexts, like talking about the "interest" of some businesses in doing something.

5 Modifications

5.1 Smoothing Probabilities

One issue that has to be addressed when learning probabilities, is that our training data is never large enough to accommodate all possible contexts, all possible words and word combinations, that occur. This may pose a serious problem if we give words that we've never seen in some context, probability 0, because no matter how much evidence we may have for that sense, we get that this sense can not occur with the word.

The common way to solve this problem is to say that

$$P(Sense) = \frac{C(Sense)}{C(Words)}$$

$$P(Word_j|Sense_i) = \frac{C(Word, Sense) + \alpha}{C(Sense) + \alpha * |Vocabulary|}$$

This is like saying that each word in our dictionary was seen in each sense α times. α here serves as a weight between the data we have, and the data that we are missing. As I will show later on, this truly improves the performance.

A close look at this smoothing shows that it actually damages our most reliable source of information - the prior probabilities of the senses. This is where my first modification comes in. Instead of saying that the words we've never seen appear with the same probability for all senses, I distributed them according to the priors of the senses. So the new smoothing formula is this:

$$P(Sense) = \frac{C(Sense)}{C(Words)}$$

$$P(Word_j|Sense_i) = \frac{C(Word, Sense) + \alpha * [\#Senses * P(Sense)]}{C(Sense) + \alpha * [\#Senses * P(Sense)] * |Vocabulary|}$$

5.2 Word Location

As I mentioned above, it seems that a word is more dependent on its surrounding words, than it is or the more remote words. To accommodate for this I added weights to the conditional probabilities, that depend on the distance of the words from the disambiguated word. The functions I used were all simple functions of the distance, with maxima at distances of +1 and -1.

$$Score(Sense_i) = \sum_{j \in Context} Weight(D[Word_j]) * log(P(Word_j|Sense_i)) + log(P(Sense_i)$$

I tried using these weights both when learning the probabilities, and when disambiguating. In training I counted the close words as if they appeared more times, and in disambiguation I used the above folmula. These gave complimentary increases in performance, and so in the final version I used them both. This modification actually raised performance in some cases by about 10-15%. The greatest increases in performance were in cases in

which local syntax was important, like the "effect||affect" spelling correction (18% increase), and the least in cases where unrelated words were connected, like the "bank||drug" set (6% increase).

5.3 Reliability of Prior vs. Likelihood

How good our estimates of the probabilities? What is more reliable, our estimate of $P(Sense_i)$, or that of $P(Word_j|Sense_i)$?

Our estimates for the prior probabilities rely on more relevant data, than our estimates for the likelihood, simply because the number of instances we have for a word in a sense is low. In cases where we have a very small training set This is particularly true. This gave me the idea to use a weight for the prior/likelihood components, that gives more importance to prior the smaller the training set we have, and an equal weight for an infinite training set (or just any finite large enough set). I tried functions of the sort $e^{\frac{1}{x}}$. Testing this hypothesis gave improvement on some of the cases, and made them worse in others. I didn't find a good weight function that gave a reasonable consistent increase in performance, and therefore did not include this in the final version.

6 Results

The following table summerises the results for a small fraction of the disambiguation experiments.

	Percent Correct (#)			
	Drug (33)	Plant(34)	Interest(392)	Space(27)
Baseline	69.7(23)	76.5(26)	61.5(241)	63.0(17)
Naive Bayes with No Smoothing	93.9 (31)	85.3(29)	77.3(303)	88.8(24)
Naive Bayes with Modified Smoothing [1]	93.9 (31)	94.1(32)	83.2(326)	92.6(25)
With Weights and Normal Smoothing	87.9 (29)	85.3(29)	88.5(347)	96.3(26)
With Weights and Modified Smoothing [2]	90.9 (30)	91.2(31)	92.6(363)	96.3(26)

Table 1: *Notice that the "Interest" testing data was significantly larger, and changes in its performance are therefore more reliable.*

References

[1] Christopher D. Manning and Hinrich Schutze, *Foundations of Statistical Natural Language Processing (229-261), 1999.*