CS 4740: Introduction to Natural Language Processing
Instructor: Claire Cardie
Jake Bass (jab783) and Ishaan Jhaveri (iaj8)

Project 2: Word Sense Disambiguation

Introduction

Program Structure

The bulk of our program was implemented in three classes: LexicalElement, Extractor, and Predictor. The LexicalElement class stores all the relevant information contained within the <lexelt>...</lexelt> tag. It stores the target word, the targets part of speech, the number of instances of the target word, all the possible sense ids associated with the target word, the total number of occurrences of each sense, a list of feature vectors, which contains the feature vector for each instance of the word and whether the element is from the training or test set.

In storing this data, a couple of interesting design decisions were made. For a lexical item in the training set, the label is obviously the sense id that is given. However, for an item in the test set where there is obviously no label given, we assigned label to the *instance id*. While this is technically erroneous (in the sense that the instance id has no direct relation to the label), we did this because is eased the formatting for the kaggle submission. Additionally, for an instance that is associated with two senses, two is added to count of total sense occurrences, whereas only one is added to count of word instances. However, this can only happen in a training set, as we only allow for one sense to be chosen for a given instance in a test set.

Additionally, we have a constants file and a validator. The constants file contains, in addition to typical universal constants, some booleans that determine which implementation of our classifier we are trying. (For example, there is a constant that determines which way we handle words with sense ids of "U".) By doing so, we were able to quickly switch between different implementation to find the best combination. The validator is also essential for this, as it allows us to make unlimited attempts (as opposed to the limit set by Kaggle) and consequently allows us to easily compare different approaches.

Preprocessing

Firstly, we did some basic preprocessing to help our parser. We removed the "&" symbol from the training set because it couldn't be processed by the ElementTree class we used. We also embedded both the test and training sets into a <dictionary> element so that it would be well-formed and so ElementTree could neatly parse it. Also, we replaced " with ', replaced ([a-z]*)='([a-z0-9 _%:]*)' with \1="\2", replace gloss=' with gloss=", and replaced '/> with "/>, as per the comment on piazza post @192. (Red text indicates the actual items replaced.)

We also encountered a problem in parsing the training data contexts. The contexts appeared in the form of "<context> text <head> targetWord </head> moreText </context>". The parser, when going through the context, would see the head start tag and assume that there was no more context. This occurred because our parser was parsing until it hit the next open bracket (<), regardless of whether it was the start tag for a different section or the end tag for the current section. Consequently, the "moreText" in the model context above would be cut off.

To remedy this, we searched through the toString representation of the ElementTree Element containing the context to manually isolate the context of each instance, and then deleted the <head>...</head> section so that the entire context was preserved. *Note: when we say deleted, we don't mean removed from the actual .data document. We mean that we preserve a version of the context that has the head tags removed. However, we would not want to actually change the .data document, as that would make it impossible to use collocation features."*

Approach

Our Classifier

We hypothesized that co-occurrence features would be the best predictors using supervised learning. Therefore we decided to use a multinomial classifier. A multinomial model is the best model for co-occurrence features because we have an exhaustive list of all possible co-occurring words - because for each element the number of token types is of the 10^3 magnitude, which is small enough that you can create an exhaustive list of all possible co-occurring words per training instance and then give them counts in the test instance. If you have this exhaustive list then this approach is very effective since it actually weighs features by their count and not just their presence. We believe that context is ultimately the best indicator of word sense and so we use this approach.

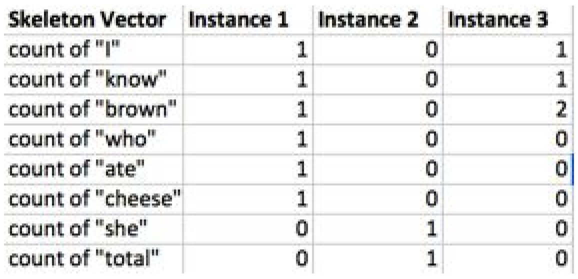
That being said we didn't want our model to be too biased towards co-occurrence features, so we included two types of collocation features in our model as well. Below in the section on Results we discuss the effectiveness of the different types of features.

We debated whether to put all these features into one model or try different models and compare their accuracies. We finally decided on a classifier based on only one model, we discuss the reasons for this below.

Feature Extraction

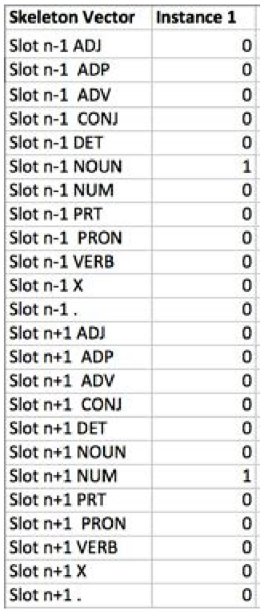
To extract the features we created a vector skeleton (a different one for each lexical element) that represented each of the features that we planned to extract. This skeleton dictated the format of the data, and then the specific data was collected for each instance of the word. Because of what I am about to mention, our skeleton vector was formed before populating any of the specific feature vectors.

For our co-occurrence features, we included the count of every unique word that appeared in the context of *any* of the instances of a specific lexical element. Therefore, we had to create our skeleton vector first in order to know which words were going to be features. Then we go through the context of each instance of a lexical element and add a vector for each one (with the couple of exceptions that we have previously mentioned, such as a word with multiple senses or heads, where we treat them as separate instances and therefore multiple vectors). In going through each context, we increment the count of the appropriate index of the feature vector for each unique instance of a word. The only exception to this is our filtration of “noise”, where there is a list of words that we do not add as features. To give an example of this, consider the lexical element “cow.n”. Consider one instance with the context “I know a brown cow who ate cheese.” and another with the context “She was a total cow.”, and another with the context “I know the brown cow is brown.” The vectors are pictured to the left.



Skeleton Vector	Instance 1	Instance 2	Instance 3
count of "I"	1	0	1
count of "know"	1	0	1
count of "brown"	1	0	2
count of "who"	1	0	0
count of "ate"	1	0	0
count of "cheese"	1	0	0
count of "she"	0	1	0
count of "total"	0	1	0

The skeleton vector is on the right, and then each column is a vector corresponding to the next instance. The words “a”, “was”, “the”, “is”, and “.” are all filtered out as noise.



Skeleton Vector	Instance 1
Slot n-1 ADJ	0
Slot n-1 ADP	0
Slot n-1 ADV	0
Slot n-1 CONJ	0
Slot n-1 DET	0
Slot n-1 NOUN	1
Slot n-1 NUM	0
Slot n-1 PRT	0
Slot n-1 PRON	0
Slot n-1 VERB	0
Slot n-1 X	0
Slot n-1 .	0
Slot n+1 ADJ	0
Slot n+1 ADP	0
Slot n+1 ADV	0
Slot n+1 CONJ	0
Slot n+1 DET	0
Slot n+1 NOUN	0
Slot n+1 NUM	1
Slot n+1 PRT	0
Slot n+1 PRON	0
Slot n+1 VERB	0
Slot n+1 X	0
Slot n+1 .	0

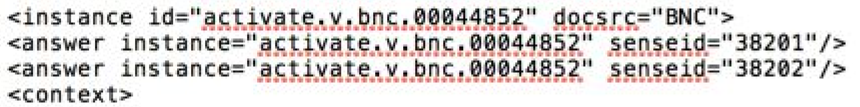
Next we added in parts of speech as a collocational feature. We did this with a list of 12 possible parts of speech taken from nltk.pos_tag, including a marker for an unknown part of speech and for punctuation. We added in the POS by, for a given slot (i.e. one word to the left of the target word, which would be referred to as index n-1, with n being the index of the target word), there is a feature for each part of speech. The feature is assigned the value of 1 if that slot contains a word of that part of speech, and otherwise is assigned a 0. We experimented with different window sizes, and we found that a window of 4 was the best. Shown to the left are the feature vectors for the Lexical Element “count” with one instance that has the context “Stephen counts twelve.” and a window of 1. As we saw before, the “.” will be ignored. The skeleton vector for the collocational POS is on the left, and the vector for the first (and only) instance is on the right. “Stephen” is tagged as a NOUN, and “twelve” is tagged as a “NUM”. This is a somewhat unconventional approach, but doing so allows us to make an almost hybrid model by incorporating the categorical parts of speech of nearby words feature into a multinomial model, an otherwise impossible task.

We also incorporate the categorical feature of counting the actual words around the target word (as opposed to the word’s part of speech). We do by combining the two methodologies shown above (the co-occurrence counts and collocational POS). When we increment the counts of co-occurrence words, we also check if that word is within the specified window. If it is, we then increase that word’s co-occurrence count by a weighted factor. Our weighted factor was increasing the count by 4 if it's 4 words before up to 7 if it's one word before, and by 5 if it's one word after down to by 2 if it's 4 words after. We decided to weight the words before by slightly more because we found that they were a slightly better indicator of the target word. The resulting vector from using this technique on our previous vector for "I know a brown cow who ate cheese." is shown to the left.

We made a few more design decisions in our feature extraction that we feel are worth specifically mentioning.
Consider the following case :
<answer instance="A" senseid="1">
<answer instance="A" senseid="2">
<answer instance="B" senseid="1">
for a single word, we chose to give P(senseID == "1") a probability of ⅔. We're disambiguating senses, so our probabilities should reflect the probabilities of senses, independent of words.

We also chose to remove all instances in the training set that had a senseid of "U". We considered not treating those cases (immediately proven substantially less effective than removing those instances) or simply assigning any instance with a senseid of "U" to the senseid with the highest probability for that lexical element. However, given the small number of instances with U's in the training set and the relatively low probabilities of each sense (because most Lexical Elements had many senses), it wound up being most effective to remove those instances. #TODO: add actual numbers in here if possible

Additionally, for every element that was associated with multiple senses (<answer>'s), we added the same training vector that many times, but with the different answer senseIDs. For example,



```
<instance id="activate.v.bnc.00044852" docsrc="BNC">
<answer instance="activate.v.bnc.00044852" senseid="38201"/>
<answer instance="activate.v.bnc.00044852" senseid="38202"/>
<context>
```

, this instance of activate would have two almost identical feature vectors, where the only difference is one would have the label "38201", and one would have the label "38202".

When there was a context that had multiple target words in it, we would do something similar and treat them both as entirely separate instances. Thus, two vectors would be added that would have the exact same co-occurrence features (because they have the same contexts) but have different collocation features.

Software

We used: sklearn.naive_bayes library's MultinomialNB class, nltk's word_tokenize and pos_tag, numpy, and xml.etree.ElementTree.

Results & Discussion

It is very difficult to pick the three best performing features with our methodology because we created our features entirely through the data, as opposed to handpicking features we expected to perform well. Because of this approach, there aren't really any features that stuck out. The strength of our approach is in that it accounts for all of the data, but this fact makes examining specific features largely irrelevant.

As discussed above we decided to use a Supervised approach using a Naive Bayes classifier trained on feature vectors with a multinomial representation. We have discussed above why we chose to use multinomial and not categorical feature vectors.

The approach we followed was to begin with a classifier and see its results, and then keep adding features into the classifier to see which ones improved our results and which ones did not. For the purpose of getting results about the precision of our classifier we made a validator tool that would split our dataset into 85% training and 15% test, randomly each time.

Our first try with our initial classifier using our validator tool on a smaller subset of the training set gave us a precision of 0.667481146305.

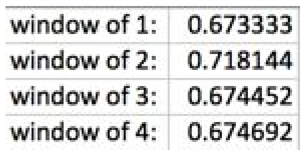
We decided that our co-occurrence features would probably be more accurate if we didn't use *all* the co-occurring words as we initially decided, and not track the counts of common words like "is", "a" etc. (For a full list of noise see Appendices). We tried this and indeed our results from our validator increased to 0.692180578005.

At this point we thought we had a good precision but we wanted to see how good it was relative to the standard of other people in the class. So we submitted the output from this classifier to Kaggle. Our Kaggle precision score was 0.60801which put us at number 11 on the leaderboard.

At this point we decided we needed to either try a different model using collocational features or add these features to our regular classifier. The issue we faced was that collocational features are typically extracted as categorical vectors while our co-occurrence feature vector was multinomial. We decided to first try adding collocational features to our multinomial co-occurrence model and see the results with this approach, and if the results didn't improve substantially only then would we try making a new categorical model from scratch.

The way we made the collocation word and collocation word part of speech features multinomial is described above. We first added collocation word part of speech features. We then needed to decide how many collocated words we wanted to include as features. We used our validator to compare precisions of 1 word before and after, 2 words before and after, 3 and 4.

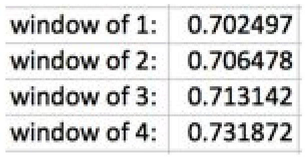
Here are the results we got:



window of 1:	0.673333
window of 2:	0.718144
window of 3:	0.674452
window of 4:	0.674692

Since a window of 2 had the best results we submitted our new classifier to Kaggle to see how much our score improved. We submitted the classifier that had co-occurrence features and collocation words' parts of speech for 2 words before and 2 words after. Our score improved dramatically to 0.61359, which put us at number 5 on the leaderboard. At this point we decided to stick with our multinomial approach and just keep the hybrid collocation features multinomial rather than training a new categorical model from scratch. We decided to do so because number 5 on the leaderboard meant that this approach was probably better than a categorical collocation approach.

So we had collocation words' parts of speech but we didn't have features for what these words actually are. We then extracted these features as described above. We needed to once again decide a window size of how many words before and after the target word we extracted features for.



window of 1:	0.702497
window of 2:	0.706478
window of 3:	0.713142
window of 4:	0.731872

Since a window of 4 had the best results we submitted our new classifier to Kaggle to see how much our score improved. We submitted the classifier that had co-occurrence features and features for collocation words' parts of speech and collocation words for 4 words before and 4 words after. Our score improved to 0.62170 which we were very happy with.

At this point we decided the only way to improve our classifier would be to experiment with adding more words as "noise" to be discounted. Our score improved once more to 0.62475.

Finally we thought we should see how we do using a Logistic Regression model instead of Multinomial Naive Bayes with all the above features. Our score was 0.62120, which was not an improvement.

So we used our final score as the score obtained using Multinomial Naive Bayes with all three types of features with a window of 4 for colocation.

Conclusion

Overall, our strategy of using a multinomial classifier with hybridization to include some collocational features did very well. One other thing we felt was a key to our success was the decision to use the entire context for forming our co-occurrence features. This was only effective because our contexts were short enough that all the words that were present were relevant to the target word, and was only a good decision because our training and test sets had few enough contexts/instances per Lexical Element that the increased amount data considered was substantial for us. However, there are a few downsides to this approach. By considering more of the context, we increase the chance of words that are far away from the target word act primarily as noise. This approach also does not scale well to a very large dataset, and does not work at all when considering very large contexts. However, we do think this strategy was well optimized to the environment of this project.

Additionally, there were areas for improvement in our approach and some varied approaches we would have liked to try. We never actually implement a purely categorical approach. While we don’t think it would have performed as well as ours, it would have been interesting to see those results. Additionally, we wanted to experiment more with our noise. While including noise filtration substantially increased the effectiveness of our approach, we never truly optimized what noise we filtered. Also, we considered weighting the collocation part of speech in a different way. Instead of weighting the counts of the words that showed up near the target words, we considered increasing the number of features we assigned to that word. (To clarify, we mean instead of counting the a word next to the target words seven times, we wanted to have it appear as if was seven different features.)

After all of the small tweaks we made to our model, our best score was obtained using Multinomial Naive Bayes with all three types of features with a window of 4 for colocation. The score is 0.62475.

Appendices

Division of Labor

We figured out how we wanted to implement our classifier together. We also figured out the preprocessing of the data together, and the basic structure of our program. Jake implemented the LexicalElement class and most of the extractor. Ishaan implemented most of the predictor and the validator. For the report, Ishaan wrote mostly about the results and discussion, and Jake wrote about the rest.

List of Noise

['the', 'and', 'or', 'it', '<', '>', 'what', 'is', 'was', 'be', 'at', 'by', 'its', 'not', 'until', 'very', '' 'to', 'of', '.', ',', "'s", 'for', '-', 'this', 'as', 'a', 'an', 'but', 'with', 'are', 'for', 'from', 'further', 'get', 'gets', 'got', 'me', 'more', 'most', 'mr', 'my', 'near', 'nor', 'now', 'of', 'onto', 'other', 'our', 'out', 'over', 'than', 'that', 'the', 'their', 'them', 'then', 'there', 'thereby', 'therefore', 'therefrom', 'therein', 'thereof', 'thereon', 'thereto', 'therewith', 'these', 'they', 'this', 'those', 'through', 'thus', 'to', 'too', 'was', 'we', 'were', 'what', 'when', 'where', 'whereby', 'wherein', 'whether', 'which', 'while', 'who', 'whom', 'whose', 'why', 'with', 'without', 'would', 'you', 'your', 'then', 'that', '/context']