Heteronymous Ambiguity Resolution for Text-to-Speech Synthesis

Paloma Casteleiro Costa

Desmond Caulley

Sonali Govindaluri

Jinsol Lee

Alex Parisi

ECE 6254

Statistical Machine Learning

Spring 2017

Project Summary:

Understanding natural languages (such as English) can be a very complicated task for machines. Understanding a language doesn't come only with knowing the vocabulary, it also requires a clear comprehension of grammar. If we want to fully understand what a person is trying to communicate, we need to pay close attention to the context and intention behind the message. This seems straightforward for most humans; but, can a computer do the same thing? Can we make a computer fully understand a message and the intentions behind it?

When two words are spelled the same but have different meanings and are pronounced differently, they are called heteronyms. Having heteronyms in a written text can lead to ambiguities when reading or translating the text if we don't consider the context.

Our actual implementation followed many of the guidelines we set during the planning phase, but did rely on some small adjustments due to time constraints. The N-Gram tagger was implemented using a package that did all the parts-of-speech labeling automatically. While it is most important to know how to interpret these results, it is also important to understand how it works and how to optimize it – due to time constraints, we were not able to develop our own algorithm for assigning parts-of-speech, nor were we able to optimize the N-Gram tagger. The Bayesian classifier was implemented almost exactly according to our specification, with the only missing aspect being the optimization. Had we performed some form of optimization, perhaps our results would have been a bit more accurate; however, due to time constraints, we opted to not perform optimization. The decision trees were also implemented very similarly to how we defined it, and like the other classifiers, was not optimized due to time constraints.

Our results were promising, but require more insight to fully understand the accuracy. When each classifier was tested using a user-generated sentence containing one of the four target heteronyms, our classification was accurate almost 100% of the time. Since it is not feasible to test our classifiers using user-generated sentences, we opted to find another corpus, vectorize it in the same way we did our training data, and run it through the classifiers. We found that most of our target heteronyms could be identified, although with a lower accuracy than we had hoped. The results also highlighted the individual differences between each classifier type – for example, the heteronym "close" was able to be resolved with a 90% accuracy using the N-Gram tagger, but had an accuracy <40% for the other two classifiers. This is because "close" can be very easily resolved using word placement ("to close" vs "close to") – since the Bayesian classifier does not account for word placement, we can expect it to struggle with this heteronym. However, the decision trees also performed poorly for this heteronym, even though it is supposed to account for word placement. We believe the issue lies within our testing dataset, although it could originate from either improperly setting up the decision trees, since they are prone to large changes from small differences, or from not optimizing our classifiers. Either way, we need to look into this issue more in order to identify what went wrong.

Detailed Project Description:

The focus of this project is to determine a statistical decision method for homographic ambiguity resolution to accomplish proper pronunciation when attempting text-to-speech synthesis. A homograph is a word that has the same written form as another, without sharing the same meaning. When these two words also have a different pronunciation, they are called heteronyms. One example would be "lead": this word can be interpreted as the verb "to lead", or as the noun "lead". Although both words are spelled identically, they have different meanings, as well as different pronunciations. Since the scope of this project is primarily concerned with text-to-speech synthesis, we will only be concerned about determining the pronunciation of the word, and as such will be working with heteronyms specifically.

In the past, N-gram taggers, Bayesian classifiers, and decision trees were used independently to solve this problem, and each method comes with both their strengths and weaknesses. N-gram taggers work to solve this issue by tagging each word in a sentence with its part of speech (i.e. noun, verb, adjective, etc.). This would resolve cases like "lead", in which its ambiguity can be identified by its part of speech, but not cases like "bass", in which both heteronyms are nouns ('bass' as the fish, or 'bass' as the instrument). Since it relies on the part of speech of the heteronym, it may also run into problems when the parts of speech are used in an unorthodox manner - e.g. using the musical definition and pronunciation of "bass", an N-gram tagger method would make its decision based on whether the target is a noun; however, the phrase "bass player", which would only describe the musical pronunciation and not the piscine, uses the target heteronym as an adjective, or specifically as a noun adjunct. Due to these types of inconsistencies, this classification scheme may run into significant problems. N-gram taggers also struggle to make longer distance word associations that may be necessary for more semantic ambiguities. Bayesian classifiers approach this problem by taking a selection of words from the left and right of the heteronym as a "bag", and classify based on word association. This implementation improves upon the N-gram taggers method because longer distance word associations are weighted the same as shorter distance word associations, allowing for a more semantic interpretation; however, these models are not able to make interpretations based on local sentence structure or word sequence, and make the mistake of assuming the context words surrounding the heteronym are independent (Naive). Decision trees, while good at handling conditional dependencies and interpreting dependent data, encounter problems when dealing with high-dimensional data.

Our implementation will use a combination of all three previous implementations, using their strengths to collectively offset their individual weaknesses. Our dataset will be accumulated using text corpora to build a large set of possible classification scenarios. We would like our corpus to contain as many instances of the target heteronym, with each pronunciation having an equal occurrence rate. Training will first begin by selecting a target heteronym, and scanning our text corpus for each instance of the word. Each target will then be given a classification label based on which definition and pronunciation is correct. Then, we will use the Bayesian classifier method to collect data, in which we analyze a specified number of surrounding words to determine context. Due to the nature of semantics, each heteronym definition will have an uneven distribution of collocations, or context words that are shared between target words of the same pronunciation. For example, if we were analyzing the collocations of the heteronym "bass", using the piscine definition and pronunciation, we would likely find that many instances of the target contain the collocations "fish" or "ocean" as a context word, whereas the musical definition and pronunciation of "bass" would not. The collocations

are then sorted by their log-likelihood values into a decision list, where larger values are indicative of a strong correlation between context word occurrence rate and the proper pronunciation of the target heteronym. The decision lists are then used to determine the pronunciation of a target heteronym in a new context.

To begin, we first had to collect our data. After doing some research, we decided used the "Reuters-21578, Distribution 1.0" text corpus as our dataset. It is a diverse collection of articles on the Reuters newswire in 1987. This data is written in Standard Generalized Markup (SGM) language, and therefore required some cleanup to reduce it to raw text form. This was accomplished with help from the "regular expression operations (re)" library for Python 2.7. Using the "sub" operation, we could remove all irrelevant information from our dataset, which included all SGM markups. The result was a raw text file consisting only of the sentences we would like to process. Most classification algorithms will not be able to accept raw text documents of variable lengths, so we must then convert the words into a series of tokens. This is accomplished using a vectorizer, which assigns an integer ID to each unique word present in the document. We used a vectorizer function that was included in the "scikit-learn" toolbox for Python 2.7. Because our final cleaned dataset contains 3,676,508 words, the process of vectorizing the entire document would take too long – to avoid this issue, we only vectorize 20 words surrounding the target heteronym.

The N-gram tagger was the easiest to implement, because most of the mechanisms were contained within the "Natural Language Toolkit (nltk)" for Python 2.7. This library contains functions that allow us to assign parts of speech labels to each word in a sentence or collection of sentences by using a context-based approach. The "classifier", for lack of a better word, contained within this library has already been trained, and therefore does not require our corpus to make a decision. All we have to do is pass the sentence containing the target heteronym into the N-gram tagger, and a proper part of speech will be assigned to the heteronym. If the heteronymous ambiguity is resolvable via its part of speech, like for the heteronyms we are testing for, the N-gram tagger should be able to assign the proper label and therefore resolve the ambiguity. Since each heteronym will have a different part of speech, we can define which label is associated with which part of speech beforehand.

After vectorizing the 20 words that surround the target heteronym, we pass them into a generic Naïve Bayes classifier as a "bag of words". The classifier determines the probabilities of the vectorized word belonging to the target heteronym classification, and trains based off word association. Because this classifier is naïve, it assumes that the surrounding words are not correlated with one another, and therefore cannot account for word placement. This is especially bad in cases where the placement of a single word can separate the two classes – e.g. "close to" and "to close" have different pronunciations and can be separated solely by the placement of the word "to". We send the three vectorized lists of words into the naïve Bayesian classifier – the first list is composed of the 10 words before the target heteronym, the second list is composed of 3 words before and 3 words after the target heteronym, and the third list is composed of the 10 words that come after the target heteronym. By combining these three results into one, we can obtain better results than each single classifier.

After performing the naïve Bayes classification, we then send the three vectorized lists described above into a decision tree classifier. Just as we did before with the Bayes classifier, we will

combine the results from each in such a way that the classification accuracy is increased. This works by extracting the most relevant features of each class for the target heteronym – this is usually the words that appear the most in one class and the least in the other class. Although one may set many "root" nodes, we decided to use one for each class. The best feature is set as the root node, and two nodes are then branched off it. Then, the second-best feature is selected to be the next root, and so on. After the tree is constructed, all we have to do is vectorize the target sentence and send it through the classification tree.

The results from our implementation were promising. We found that our classifiers each handled user-entered sentences incredibly well, with a high accuracy; however, when using a secondary corpus and using that as a testing dataset, our accuracy became much lower. Perhaps this issue lies with our vectorizer, or how we remove markup language from the corpora – we will require more testing to verify this. After choosing four heteronyms, we set them as our targets, and began vectorizing the surrounding data before plugging it into our classifiers. First, they are trained off the extracted data, and then tested by the data extracted from the secondary corpus. Typically, there is a layer of optimization present after training, but we opted to not include this due to time constraints. The table below shows the results of our classifiers.

N-Gram Tagger	"Object"	52.5%
	"Minute"	87.5%
	"Close"	90.0%
	"Use"	97.5%
Naïve Bayes	"Object"	80.0%
	"Minute"	97.5%
	"Close"	32.5%
	"Use"	82.5%
Decision Trees	"Object"	80.0%
	"Minute"	97.5%
	"Close"	12.5%
	"Use"	87.5%

As you can see from the results above, the classifiers performed generally well on the testing data; however, the Naïve Bayes and decision tree classifiers struggled significantly with the heteronym "close." The Naïve Bayes should understandably struggle with this word since it can be resolved by word placement, something the Naïve Bayes doesn't look for – however, the decision tree classifier should not have struggled as much as it did. We think there may be some discrepancies with our testing dataset since the decision trees classify user-generated sentences containing heteronyms very accurately; but, this may not solely be the case since the N-Gram tagger is able to resolve "close" very accurately. More research needs to be done on both the testing set and any optimizations we can perform on the decision tree algorithm to increase accuracy. This is a first-hand example of how decision trees are unstable, and how the accuracy can be drastically affected by tiny variations in the input data.

We would have liked to implement some combination of each classifier into one singular classifier. This is an example of an ensemble method, which we learned in class can produce very good results even when the individual classifiers do not perform well. Unfortunately, due to time constraints, we were unable to create this implementation. Even without editing our current classifiers, we believe that we would be able to produce a high classification accuracy.

Overall, the ramifications of this project are generally related to quality-of-life increases for text-to-speech synthesis. Mispronounced words tend to sound "grating" in their context sentence, and can produce an end-product that isn't as satisfying for the user as it should be. One market that can benefit directly from this technology is for visually impaired computer users who rely on text-to-speech synthesis software in order to understand what is written in front of them. In this case, mispronouncing a word can cause confusion to the user, since they are not able to resolve the ambiguity themselves by sight. Such a mispronunciation may cause the listener to miss the next sentence, or may require them to play the sentence back multiple times in order to understand the meaning – a frustrating procedure for someone who struggles to interact properly with a computer. Our implementation lays the framework for a larger implementation that could resolve the ~150 heteronyms in the English language, which can then be used to improve existing accessibility software.

References:

Hearst, Marti A., and Xerox Palo Alto Research Center. "Noun Homograph Disambiguation Using Local Context in Large Text Corpora." (n.d.): 1-15. Web.

Rivest, Ronald L. "Learning Decision Lists." (2001): 1-18. Web.

Weber, Heinz J. "The Automatically Built Up Homograph Dictionary - A Component of a Dynamic Lexical System." (n.d.): 1-11. Web.

Yarowsky, David. *Homographic Disambiguation in Text-to-Speech Synthesis*. N.p.: n.p., n.d. Print.