Leon Yao

LINGUISTICS 183

12 March 2014

Final Project Write Up

Machine Translation using Similarity Pruning

My final project produces a dictionary from English to French that is learned from 100,000 translated sentences from the europarl corpus. Each line from the corpus contains an English sentence or phrase and its corresponding translation in French. The corpus contains all sorts of sentence structures and forms of punctuation and captures a good representation of the many uses of the language, which makes the translation more accurate and realistic. I choose to use 100,000 lines of translation because it is large enough to capture the commonly used words reasonably well without having to increase the run time by too much. In my paper, I will first describe my program in 3 parts: the preprocessing, the pruning and the post-processing, then follow up with a discussion of how well the translation system does, and finally end with future considerations to better the project.

**Preprocessing:**

The English language is fairly straightforward in how the sentences are formed and does not need much preprocessing. French however is difficult because many nouns that begin with a vowel will have an “ l’ “ attached before it and verbs that start with a vowel will have a similar attachment depending on gender and number. These attachments are words themselves in the English language so in order for our translation system to recognize them, it is necessary to split them off from their base noun or verb. Similarly, words such as “du” or “des” are conjunctions of the form “de le” and “de les” respectively, which are also separate words in English. Because there are often punctuation attached to words in either language, such as periods and commas, when parsing, it will be recognized as part of that word. Thus, in the preprocessing, we must replace all forms of punctuation with a blank. In the case of a hyphen we replace it with a space instead, because both parts of the hyphenated phrase are often independent words themselves. In the europarl corpus, the French sentences use Unicode characters to represent the letters with accents. Unfortunately while converting to strings, these symbols get translated to their string representation of the Unicode in python and become a non-descript jumble of letters. To fix this, I manually found what each Unicode string translates to in the alphabet and replaced all instances of it.

**Pruning:**

The algorithm I use to actually create these translation is founded upon the idea that if I matched words from the sentence in English to every word in the French sentence and counted all pairs of the words in the entire file, then presumably the pair that is the actual translation will appear many more times. Thus in my code, I just loop through the words in each corresponding sentence translation and count the number of time each pair appears. I use a 2D indexable dict and a Counter to counts these frequencies.

However, the main difficulty with this approach is that there are many false positives if there are words that just always appear in any sentence. For example, many common words like “a”, “of”, “the” and “from” and its French translation “un”, “une”, “de” and “le” will be paired with every word tens of thousands of times, which will far surpass the count of the actual translation pair. One way to solve this is to look at a given pair (X,Y) and see how often X appears in English and how often Y appears in French. Ideally a word and its translation should appear in the text equal number of times. However, there are many words with multiple definitions and verbs in English with many conjugated forms in French that makes this one-to-one assumption inaccurate. However, there should be a pruning factor that can decide whether the counts of the English and French words are close enough to be considered a valid pairing. This pruning factor I will call the Jaccard Similarity, and this is defined for a pair (X,Y) as the size of the intersection of X and Y divided by the size of the union. This essentially is equal to the number of times both appear in the same sentence translation divided by how many times either appear at all in the corpus. Thus words like “un” and “le” which appear tens of thousands of times will have a very small Jaccard Similarity when compared to any normal word that appears much less. I implement this by counting the number of every word in the corpus in my preprocessing step.

**Post-Processing:**

After seeing how my translation system does with just the pruning, I realized that many words have counts that are very small and many ties occur. Because the size of the corpus is so large and many news words are introduced, it seems there is a huge tail end of words with only a count of 1 or 2. This makes it hard to prune because when the count is so small, a pairing between a rare English word with a rare French word is equally likely as it and its actual translation. Because python naturally sorts its lists alphabetically in a Counter, it seemed that the best translation outputted was always the first one alphabetically. In order to fix this and choose the right one from a list of words with equal frequencies, I used an edit distance approach based on the assumption that words in English and French are very similar because many words in English come from French words. There are many instances in which this assumption is false (i.e. false cognates), but for many translations, this allows a pretty accurate algorithm to choose the right translation.

**Results and Analysis:**

In order to determine how well my translation does, I wanted to determine if it translated the most common words in English correctly. Because the corpus does not often capture obscure words very well, I just chose the most used nouns, verbs, adjectives, prepositions, and misc. words according to Wikipedia. Attached are my results for Jaccard Similarities of 0.03, 0.04, 0.05, 0.08 and 0.1. It seems that for 0.03 and 0.04, the factor is just low enough that it does not prune away the false positives like “de” and “la”. With 0.08 and 0.1 the results show that many words don’t even have a translation at all because no pairing passed the pruning step. For this particular training set, it seems that 0.05 is the optimal similarity factor to use in order to capture the best translations. Additionally, I noticed that in all similarity constants used, nouns always performed the best. This is because nouns do not have conjugations except for the occasional plural. Nouns are also usually both represented as one word in English and French and often do not have multiple possible translations for the same word. This allows the frequencies of nouns to be roughly equal and this will always pass the pruner. Similarly, adjectives do quite well because adjectives only sometimes conjugate based on gender and number. However, for a given word like “big” or “high”, there are actually many possible translations, so it performs worse than nouns. Verbs perform terribly using any pruning constant. This is because there are at least 6 different conjugations of a verb based on gender and number. Thus even if a verb pairing is used many times, it will split its count between each conjugation, which may lower the count enough that it will not pass the pruner. It seems that for verbs, a less strict pruning constant performs better and 0.03 actually performs the best. In all cases however, prepositions and misc. words such as “to”, “of”, and “in” are too frequent in the text and the similarity constant needs to be shifted way lower in order for these translations to work, if at all. When the count of “de” is >300,000, it becomes very difficult to find a pruning constant that addresses this case and doesn’t invalidate all other cases.

**Future Considerations:**

Some future considerations that can make this translation system better are to instead do counts of an English word to a pair of words in French or vice versa. However, this is not only computationally heavy, the count of word phrases will be even lower than individual words and thus harder to capture through pruning. But with enough computational power and better pruning process, we can deal with even higher degree of N-grams, which will make the translation significantly better. This is because there are too many intricacies of the French and English language that is missed simply by pairing single words. For example, “not” in English is represented by the pair “ne…pas” in French. The distance between the “ne” and “pas” could extend many words, and this simple miss in translation can dramatically change the meaning of the sentence.

Another consideration is to use a different similarity constant for each type of word (noun, verb, etc.). However this implies that we already have a categorization of every word in English and French, which would contradict the statistical approach of machine translation. However, this seems to be a good approach to prune for all types of words. By increasing or decreasing the constant, it will better or worsen different types of word translations. Thus, we can analyze word structures based on the typical forms of nouns or verbs to determine what type a given word is.

**Notes on running the program:**

The program is run simply by the command “python translate.py” in the command line. I have a section that is commented out near the bottom. This section will print the translation of every word in the corpus. The bottom part that is uncommented is that test on the most frequent words I described above.