## horizontal line



Project Progress Report

10.23.2017

**─**

Li Breite, Mitko Nikolov

Northeastern University

Boston, MA

# Changes

## Technological changes

Everything technologically has gone according to the way we envisioned our process, besides from one thing. We originally intended to use the WikiLinks (Amar Subramanya, 2012) corpus by Google in order to create our data model. However, after further investigation, we found out that this source was not a good option for us as it focuses on classification of data. We still think that WikiLinks could be useful to us eventually, but we are not sure whether we would use it at all. We did some research and settled on the Brown University Corpus of Present-Day American English (Francis and Kucera, 1964). We used *Form A* which is the original form of the Corpus, as it was prepared in 1963-64. With *Form A* of the Brown Corpus, we now have a big corpus from which to extract and process data in order to generate unigrams, bigrams, trigrams and POS tags.

## Non-technical changes

Another change that we had to deal with was the fact that one of our teammates dropped the class. As all members of our team were valuable contributors, this was a disruption that led to us planning again how to approach the development process and re-distributing the work so that we can handle the same workload with one less person.

# Data preprocessing

*Form A* of the Brown Corpus is preprocessed on multiple levels and in different ways. On the highest level, all the data from the corpus is processed into separate files. Each file is about 20KB and has between 60-80 sentences of varying lengths. Each file falls into a certain category. Some of the categories are: news, editorial, reviews, religion, hobbies, lore, government, fiction, mystery, adventure, etc. Each category has its own naming convention so for example files from news are named “caXX” where XX is the consecutive number of the file - starting at 00.

On a file level, the Brown Corpus is preprocessed by using data identified with sentence sequences. Before the beginning of each sentence, there are two empty lines. After that follows the sentence itself. At the end of the file there are two empty lines too. In this way, it is trivial to identify separate sentences so that they can be processed in our code.

The data also contains the POS tag of which a word is after the word itself. Furthermore, it has each punctuation mark separated from the word before it so that processing punctuation marks is not any different from processing other words.

The quality of the preprocessing was indeed one of the features that we considered when we chose the Brown Corpus. Because we did so, we did not need to do a lot of preprocessing work. The only problem we had to decide how we would like to deal with was possessive forms. For sentences like “John’s car is slow”, we had to decide whether we would want “John’s” to be one word or to be broken down into separate words somehow. What we decided was that every time we see an apostrophe as in “the players’ ” or “John’s“, we are going to split the word into two parts: one word would be everything before the apostrophe and the other the apostrophe and everything after it. So after we process the data on our end, “ the players’ “ becomes “ the players ” and “ ’ “ whereas “ John’s ” is now split into the words “ John “ followed by “ ’s “. This is all the preprocessing that we have had to do so far.

# Methods

## Development Management

In terms of managing the development process, we are working on two different perspectives. One is developing a good language model and the other is creating an application that is able to interact with the Microsoft Computer Vision API (Microsoft Azure). Our plan is to develop these two modules and then make them work with one another. We decided to use this approach because we believe that by dividing the problem we have into smaller ones and then combining the solution into one big solution, we will be able to solve it more effectively and probably better. So far we have focused on developing a robust language model and we believe that by next week, we should be able to shift our focus on creating a mobile application.

## Development

While developing our language model, we used different smoothing methods. We implemented unknown-word handling as both of us felt that this was an important part of our work when we did assignment 1 for the class. We used the same methodology as in the assignment - if a word was seen less than 5 times, we considered it part of a class called UNK. We did this because it helps us smooth the data and enables us to still find probabilities for words that are only rarely seen or have been completely unknown to us up to this point.

On assignment 1, both of us got better results from interpolation than from add-1 smoothing so we implemented interpolation in our project. As I will describe in the next paragraph, we fine-tuned the proportion variables for unigrams, bigrams, and trigrams in our interpolation smoothing so that we decrease perplexity as much as absolutely possible.

## Testing

In terms of testing methods, we first split the data into respective parts for training, development (fine-tuning) and testing. From each category, as described above, we used 70% of the data for learning. The rest 30% we split into two chunks: one of 10% and one of 20% for fine-tuning and testing respectively. We decided to do it this way because first of all we knew that we had to use the majority of the data for training. Ultimately, our goal is to create something that can analyze and generate sentences relatively well. However, how do we know whether something is good, if we cannot perform ample testing? So we decided to have 20% of the data for testing and 10% for fine-tuning. We believed that 10% of such a big corpus would be enough for improving our language model and, as I describe below, so far it seems so. After we trained our language model, we first performed testing on the development set. We spent some time with various examples in order to identify how we could improve our interpolation method proportion variables (the proportions between unigrams, bigrams, and trigrams) in order to achieve the best results. The main method for this fine-tuning was keeping one of the variables steady while adjusting the other two. We also tried to create an enumeration of different combinations of proportions between the different variables for unigrams, bigrams and trigrams and finally settled on the proportions 0.14, 0.25, 0.61 for unigrams, bigrams and trigrams respectively. These specific methods were used because as far as we knew there were no standard ways to choose the interpolation variables so we came up with our own methods. We settled on the variables above because they led to the best perplexity result.

# Evaluation

After we trained our methods, as mentioned above, we used the unseen-before development set in order to fine tune the different proportion variables used for unigrams, bigrams and trigrams in our simple interpolation method in order to achieve the best perplexity. At the end of this, we ran tests on our test set by different categories. Our perplexity for test files from some categories (like news) is as low as 80 and for other varies around a 100. When we run the evaluation method on all the test files we have, the perplexity we get is 93.

So far everything seems to work to expectations and we have not encountered any serious problems. Our language model seems well-smoothed and generating good predictions and probabilities for sentences.

# Next steps

We have a lot more and exciting work to do before our project is done. We would like to generate some sentences with our language model in order to see how meaningful they would be. Then we need to understand how to get meaning from a sentence that our language model has parsed. Once we understand semantics, our language model would become much more sophisticated. As a result, we will be able to link a vocal request from our users to functionality in our application.

Next week, we will start focusing on developing our mobile application. We first need to understand the basics of Android development. After that, we are going to focus on understanding perfectly the Microsoft Computer Vision API and how to interact with it. Once we have done these substeps, we are going to focus on the bigger picture which is to create the application.

When we have a functioning app, we will also perform extrinsic testing by using the app to evaluate the quality of our language model.

As we start spending more time working on the application itself, we also plan to reach out to a specific foundation for visually impaired people. We will send them a survey which will ask them about how useful the features we have in mind would be to them and what would they need the most. In this way, we will find out if we should implement the features we want to, whether we should change them somehow and what other ideas we should spend time on.

# References

1. Microsoft Azure. “Computer Vision API.” Computer Vision-Image Processing and Analytics, Microsoft Corporation, <https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/>.
2. Francis, Nelson and Kucera, Henry. “Brown Corpus : Nelson Francis and Henry Kucera : Free Download & Streaming.” Internet Archive, Brown University, 1 Jan. 1964, https://archive.org/details/BrownCorpus. The corpus is being used under Creative Commons License (<https://creativecommons.org/licenses/by-nc/3.0/>)
3. Subramanya, Amar, et al. “Google Code Archive - Long-Term storage for Google Code Project Hosting.” Google, Google, 1 Oct. 2012, <https://code.google.com/archive/p/wiki-links/>.