Final Report

12.11.2017

Li Breite, Mitko Nikolov
Northeastern University
Boston, MA
Problem Description and Project Introduction

We started out this project wanting to create an Android application that can help improve life for people who are blind or visually impaired. The product would be able to take a picture and then answer questions such as “What is in this picture?” “Is there a cat in this picture?” “What color is this shirt?” or “What does this text/sign say?” The software can prove to be extremely beneficial to people who cannot see as they are trying to navigate their day-to-day lives in an unfamiliar setting. For example, the act of choosing matching clothes might appear to be a rather mundane task for a seeing person, but could be a real challenge for someone who cannot see. With this app, they can just take a picture of the shirt they are planning to put on, and ask “What color is this shirt?” The app would then be able to answer something along the lines of “This shirt is red,” “That is not a shirt, it is a dress,” or “I do not see any articles of clothing in this picture.” It is important that this problem is studied and worked on because visually impaired people are a significant and important part of the human race. Just as any other person, they deserve opportunities and support and creating something that can assist them in the described way would be a step on this path. We hoped that the described software will be able to go a long way in assisting people who need a virtual pair of eyes in navigating the world. However, along with great, this software was very challenging to implement for two students given that we both had 3 other classes which included projects and multiple exams. Due to time restrictions and the amount of things we needed to learn and experiment with, we ended up implementing an app that has similar functionality but is simplified. Given an already taken picture through a separate camera app, our mobile software analyzes it and can answer simple questions about it, such as “Is there a dog on this picture?”, “What is written on this picture” or if they would like a general description - “What is in this picture?” We have not integrated the camera of the phone to take a picture directly through our app yet. We also did not implement text to speech or speech recognition software. Instead, all interactions with the app must be done with orthography (typing the question into a textbox), and we use the last picture that the phone took. While we understand that in its current state the app could not be used by blind people, we believe that we have implemented the most important functionality - understanding a question and answering it based on the results that the Microsoft Computer Vision API generates after analyzing the request.

Now that we have the most important functionality implemented, we hope to, in the future, add text to speech and speech recognition (using the built-in services in android) and to integrate the phone’s camera. In this way, we would make the app more easily accessible for people who are blind or visually impaired.
Related Work

There is a lot of work that can answer questions. For example, Google is able to understand questions and answer them accordingly. Additionally, there is technology that can look at a picture and return a description of what is happening in it. However, the combination of the two technologies is, as far as we know, rather novel. Our application would need to be able to understand a question, look in the picture for the answer, and respond correctly to the question. In terms of competition, there is a company called pivothead (see reference 7) that more than a year ago expressed their interest in creating sunglasses with an embedded camera that could do something similar. However, they designed their product as a tool for people to share where they are and generally read signs around them. We are targeting specifically blind people and are focusing on serving specifically their needs. Alongside with this, to use our product, people would only need their phone. We do not know of any other applications or software that can do all of these things. After submitting the progress report, we came across a report for a student project that would generate captions for pictures to help visually impaired people, but that project did not answer questions about the picture (see reference 1). Creating a caption for a picture is akin to answering “What is in the picture?” However, our application is much more powerful in that it will be able to answer more detailed questions about the photo. While we relied heavily on APIs that can analyze photos and also a framework for semantic parsing, we implemented the question answering, the core of our project, ourselves.

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 had to do.

We did need the POS tagger in the final version of our project. Instead of implementing our own, however, we ended up using the Stanford Parser. We experimented by running our application on different request using our own tagger and Stanford’s and the latter generated better results on average. So we decided to take advantage of it as it was freely available.

**Methodology**

**Language Model**

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.

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.

**Question Answering**

We originally wanted to implement a semantic parser ourselves, but it soon proved to be much too complicated for us to implement, and Professor Wang suggested that we use a premade one. We then decided to use SEMPRE, but that also proved to be unnecessary. We ended up using a simpler approach: first parse our sentence and then look for patterns in the way that the questions were formed. We made this decision after after analyzing what is the domain of questions that our software would need to respond to. It would initially just need to understand questions about whether something is on a picture, what is written on a picture and what is on the picture generally. Because the questions’ type variety was limited, we decided that, at least initially, we actually do not need semantic parsing whatsoever.

Our question answering part of the application works as follows. First, we used the Stanford Parser to parse the input string. If the first word in the question was “is” or “are,” we assumed that the user wanted to know if a specific item existed in the picture. In this case, we then extracted all of the adjectives and nouns in the query string, and compared them against the tags in the analysis returned by the Microsoft Computer Vision API. If the tags and adjectives/nouns matched, we answered yes. Otherwise, no.

If the first word is “what” we looked for “is written,” “is typed,” “is said,” or “does say” (not necessarily immediately one after another) in the rest of the string. If we found one of those, we returned what was written in the picture.

If neither of those cases matched, we just returned a description of the picture.

Taking this approach was a decision that simplified our task significantly and allowed us to implement the most important functionality of our software - answering questions on an analysis of a picture. We do realize that we would need to instruct people to always start questions with is/are if they want to check whether something is on the picture and to always mention written/typed/said if they would like to get information about what text is on a picture. We do not consider this a burden, however, and the advantages that it brings are significant.
Experiments and Evaluation

Language Model

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.

Question Answering

Our evaluation for the question answering was done mostly extrinsically. We ran the app on multiple devices (virtually) with both taken and downloaded photos, and confirmed that our question answering worked in all cases. We have created videos of us using the app on different pictures for different purposes. Please find attached demos in the folder “Demo”.

Conclusion

We felt that we learned a lot about NLP from doing this project - mostly creating a language model and question answering. We enjoyed working in a team on this project, and, while we did not implement all of the functionality that we originally planned to implement, we are proud of the work that we did accomplish. We managed to create the functionality which we consider vital - extracting information from the tags and description that the Microsoft API returns in a way that would make the analysis useful to blind and visually impaired people. We both hope to continue working on this project after the completion of this course and plan to discuss and arrange it. We believe that by implementing the rest of the ability to understand speech and speak back responses and to directly take pictures through the app, we can make a change for a group of people. We are excited to be able to help those who are blind and/or visually impaired navigate the world a bit easier.
References


6. Pivothead’s website
   http://www.pivothead.com/