Brian Loi
Professor Peter Jansen
ISTA355 Natural Language Processing
May 1, 2019

ISTA355 Final Project Write-Up

Task

The natural language processing task that my project performs is a question answering classifier through a pipeline and format similar to that of a search engine (open question and answers). Essentially, two tasks are involved in this project. The first one is an open-ended, free question answering algorithm solely using cosine similarity, and the second is a question answering classifier that utilizes the aforementioned algorithm as a feature.

The task for the open-ended question answering algorithm that when given a question, phrase, set of words, or word, the algorithm produces a ranked list of answers, in some range. For example, if given "famous suspension bridge in San Francisco, California," the algorithm will output a list of the top n number of similar "titles" to that statement. Ideally, 'Golden Gate Bridge' will be on this list and the higher it is on the ranked list (closer to 1) the better the performance of the algorithm. Likewise, the lower it is (closer to n), the worse the performance of the algorithm is. By "titles," I am referring to titles of Wikipedia pages, therefore a title could be a concept, person, place, object, etc. The algorithm will attempt to answer questions through these titles.

The primary task of the question answering classifier is to generate informative features in classifying answers in a question answering data set to achieve the highest performance score I can achieve in the project time frame. These informative features will include cosine similarity scores, containment of bigrams, containment of trigrams, number of words in the question, and number of words in the correct answer's Wikipedia page's text content. The naïve Bayes classifier will be trained on positive and negative train sets consisting of questions and answers extracted from a data source. Part of this task will inherently include identifying the most informative features and trying to comprehend why.

Overall, the task is to is to create a question answering model that replicates the functionality of a search engine. Accordingly, the primary task also involves training and testing a classifier with the goal of accurately identifying correct answers for questions. The program file containing all code for this task is named "Final.py".

Data

The data I used for my corpora, or text source, was a Wikimedia dump. The snapshot of the pages was of 3:56AM on April 9, 2019 retrieved from https://dumps.wikimedia.org/enwiki/latest/. The dump contains 15030 Wikipedia pages in a XML file format. This XML file was converted into multiple text file pages by using WikiExtractor so that the Wikipedia pages could be more easily read in by python. Every individual text file was read in and parsed into a Pandas DataFrame so that each row contained information on the page, such as the page's title and the text content of the page. This DataFrame is then referenced for vectors and the weighing of vectors.

The question answering data set that I used for my model primarily came from the source https://data.world/sya/200000-jeopardy-questions. This dataset contains 200,000 questions and answers that were shown on the television show Jeopardy. Of those 200,000 questions and answers, only about 15,000 were used in the model. This is because the questions and answers were filtered so that the answers had to be a title of a Wikipedia page in the Wikipedia data set. I decided to do this, because it would be unreasonable to ask and evaluate a model with a question in which its answer did not exist in the model. After this filtration, the questions and answers are randomly shuffled and placed into a train set, dev-test set, and a test set. 60% of the shuffled question and answers were placed into the train set, 20% into the dev-test set, and 20% into the test-set. This data is then used for training and testing the classifier model.

Existing Software/Tools

Many libraries were imported for this project for various reasons. The reasons ranged from optimization, visual tracking of calculation progress in the model, to functionality. The libraries used include: NLTK, OS, Pandas, NumPy, math, progressbar, string, collection's counter, operator's itemgetter, gc, random, and pickle.

To convert the WikiMedia XML dump file into multiple txt files for easier reading, I used WikiExtractor. WikiExtractor has a GNU General Public License and is free to the public. The link to WikiExtractor's documentation and home is http://medialab.di.unipi.it/wiki/Wikipedia_Extractor#Introduction.

Methods

First, the WikiMedia dump needs to be converted from a XML file into multiple text file by using WikiExtractor. Next, the text files are read individually, parsing each Wikipedia page into a Pandas DataFrame. In the DataFrame, the indices consist of the title of the pages, and has two columns: the textual content of the Wikipedia page, and the vector of that textual content. Here, the vectors are in the form of a dictionary, in which the key is the word mapped to the number of occurrences of that word in the text content for that page. As the text files are read in, each cell of the DataFrame is created. After this is done, an overall dictionary is created in which the keys are words that map to the number of Wikipedia pages that contain that word in their text content. This is done by going over each key in a vector dictionary, and adding counts to that overall dictionary. This dictionary is kept in the DataFrame as a string object. Next, that overall dictionary is used for inverse document frequency when weighing each page's vector though TF-IDF. The term frequency is calculated using the logarithm technique: log (1 + (occurrences of a word/total words)). The inverse frequency frequency is calculated as log (total number of pages / number of pages that contain the word). After these are calculated, they are multiplied together to produce the new value that will take the place of a single value in a page's vector dictionary. This weighing occurs for every page for every single value in a vector and replaces the old vectors in the DataFrame. Once all the vectors have been weighed, the DataFrame and overall dictionary are saved as their own pickle files for easy access. Note that the dictionaries are stored as strings in the DataFrame so that the dictionary object can easily be converted back and forth with the str() and eval() functions.

Next, the question answering data needs to be split and separate files for each set need to be created. To do this, we will first need the Wikipedia DataFrame and overall word to page dictionary, so those objects will be needed to be loaded in for this. Then, the 200,000 question and answer Jeopardy CSV file is read in row by row, storing questions and answers into a list of tuples (question, answer). During this process, questions and answers are filtered so that the the questions and answers extracted have answers that are Wikipedia page titles in the Wikipedia DataFrame. Afterwards, the list of questions and answers are shuffled and 60% of them are placed in a train set, 20% are placed in a dev-test set, and the last 20% is the train set. Each set is then saved into their own CSV file for ease of access later on. This process of shuffling and splitting the sets can be done multiple times to get different outcomes of data in the randomly shuffled sets.

From here there are three different options (A), (B), and (C) for what can be done using the data:

(A) The data can be used for a naïve Bayes classifier model. First, the Wikipedia DataFrame and overall word to page dictionary along with all of the question answer sets need to be loaded in. Next, feature sets are generated for the train set and dev-test set. A negative feature set using the train data was also created in order to train the model on what features would be like with an incorrect answer. The features that are contained in these feature sets are the cosine similarity of the questions vector and the Wikipedia page's textual content's vector that is said to be the answer to the question, whether or not any bigrams in the question are contained in the Wikipedia page's text content, whether or not any trigrams in the question are contained in the Wikipedia page's text content, how many words are in the question, and how many words are in the Wikipedia page's text content. After all these features are generated for their respective feature sets, the training feature set and negative training feature set are used to train a naïve Bayes classifier. The performance of the classifier is then evaluated with the dev-test feature set, and printed out along with the top ten most informative features. Next, the user will be prompted on whether or not they would like to run that classifier on the test set. If the user wants to, the classifier will evaluate its performance on the test set and print out the performance score. Afterwards, the program will end.

(B) The data can be used for an open-ended, free question answering algorithm. First, the Wikipedia DataFrame and overall word to page dictionary will be loaded in. The algorithm will then continuously ask the user to enter a question, phrase, or any number of words until the user enters 'quit.' Next, that question will be converted in to a vector dictionary in which each word in the question will be mapped to the number of times it appears in the question. That vector dictionary will then be weighed using TF-IDF accordingly, using the word to page dictionary. A progress bar will then begin in order to help the user see the progress of the answer searching and speed of the algorithm. For every weighed vector dictionary in the Wikipedia DataFrame, both that vector dictionary and the question's vector dictionary will be extended so that they are the same length and normalized to unit length. Afterwards, the cosine similarity of those two normalized vectors will be calculated and stored into a list of tuples with tuples in the format of (page title, cosine similarity). This process is done for every single page in the Wikipedia DataFrame, and the progress bar is increased for each cosine similarity calculated. Once the cosine similarity of every page is collected, the top n number of tuples with the greatest cosine similarities are extracted and printed out on a ranked list for the user to see. The user will then be prompted to ask another question and the cycle continues forever until the user enters 'quit' to end the program.

(C) The data can be used to evaluate the open-ended, free question answering algorithm from option (B). Similar to the other option, the Wikipedia DataFrame and overall word to page dictionary will be loaded in first along with the dev-test question answer data. The dev-test question answer set is limited to thirty questions and answers. This is because it takes approximately a minute or two to answer each question, meaning it would take unreasonably long to evaluate the algorithm on several thousands of questions and answers. A progress bar is used here to keep track of the percentage of questions that have been answered. Then, every question from the dev-test set is used as a question to ask the algorithm, and the algorithm outputs a mean reciprocal rank for each question, depending on where the correct answer is on the algorithm's ranked list of answers. This is done for each question in the dev-test set, and then the scores are accumulated and printed out as the performance of the algorithm. The program ends here.

Evaluation

The classifier model's performance was evaluated using NLTK's accuracy method. After being trained with a positive and negative train set, the classifier was then evaluated with the dev-test set. Once all adjustments were made for the dev-test set, the classifier was evaluated on the test-set. Both sets yielded performance scores that were very close to each other, so no over-fitting occurred.

The most informative features for the classifier model were quite interesting. The top ten most informative features for the model are usually all pertaining to the word length of the correct answer's Wikipedia page text content. This is most likely because the text content of many Wikipedia pages can be short, consisting of links to other Wikipedia pages. On the other hand, it could be that they are short in general, often being an incorrect answer to a question since they are not detailed enough to be an eligible answer. According to the most informative features, Wikipedia pages having about 9,000 to about 19,000 words in its text content is a good indicator of a correct answer. It is quite surprising to me to see that cosine similarity is not one of the more informative features.

For the evaluation of the performance of the open-ended, free question answering algorithm that solely uses cosine similarity for its question answering, mean reciprocal rank was utilized for calculating the performance. This was done so that the window of answers for the algorithm was enabled to be correct despite not having the correct answer as the top answer from the algorithm. As it is now, the algorithm's window of answers is set to size 10, and if the answer is not in the window, a score of 0 is given. If the correct answer is ranked 4 in the algorithm's list answers, then the score given for that question is 0.25. Unfortunately, due to the slow runtime of the algorithm, the data set passed in had to be limited, therefore I was unable to accurately evaluate the performance of the algorithm.

Due to the lack of time, this experimental model was not compared to a baseline model to see how effective the experimental model is or how much this experimental model accomplished at its task.

Results/Performance

The performance of the classifier is consistently around 80%. This is even with different random shuffled train and dev-test sets. In the end, when testing the classifier on the test set, the performance is also close to around 80%. The classifier's overall performance shows that the task is well-accomplished.

When testing the open-ended, user input question algorithm based solely on cosine similarity, the algorithm performed horribly, at <2% performance. The performance was based on a mean reciprocal rank scale. Although, due to a slow runtime, the performance is based on an extremely small portion of the dev-test set.

Issues

There are several issues that come from the structure of the model, data, and in general that I have come across while creating the model and program. The data creates an issue because it means that the model is limited to the data. In other words, the model and program are unable to answer any questions accurately if the answer is not in the Wikipedia dump, therefore limiting its capabilities. Ideally, an open ended question answering model should be able to answer any question. In addition, the model is restricted to using the titles of the Wikipedia pages as its answers. This constrains the questions that the model can accurately answer freely to "who" and "what" questions, and does not allow the model to answer any open-ended questions in the form of "why," "when," or "how." It is important to note that this only affects the free and open-ended questions and answers (based solely on cosine similarities) and not the classifier.

A large issue that has hindered my progress throughout this project is the slow runtime of finding the answers to the questions. Despite optimizing the program as much as I could, freely asking the model a question can take up to 2 minutes, and this is when the only feature considered is cosine similarity. The classifier is able to avoid this runtime by already knowing what the correct answer. For example, when running the program with command 2 (Ask the QA Model Questions), the model will calculate the question's vector with every single Wikipedia page's vector to find the top most cosine similar pages which is a long task.

Another issue with how I have structured and program the model is that vectors are initially stored as dictionaries. Since the dictionary is stored inside the Wikipedia Data Frame, the dictionary is stored as a string which is later converted back into a dictionary with the eval() function. This is an issue because the eval() function runs a string as if it were code, so any quotations inside that string could easily mess up the data. As a result, any possessive words had their apostrophes removed and no questions containing apostrophes were included.

Next Steps

Moving forward and the next steps for the program and model are quite minimal, though may be difficult tasks. Such next step tasks involve adding more features to the classifier to improve its performance, but I believe that the performance will only increase slightly from here. A major task that should be done in the future is optimization for answering questions freely. The model currently answers questions freely at a very slow rate, about 2 minutes per question, and should be optimized further as much as possible. This would allow for better evaluation of the question answering model based solely on cosine similarity for open-ended questions and answers.

Since the lack of time prohibited me from comparing this experimental model to a baseline model through non-parametric bootstrapping, this task is one of the first priorities for next steps. This is an important future task because it will tell us how effective this experimental model was and how well the experimental model accomplished its task.

In general, with moving forward, the next step would be to expand the capabilities of this model from solely the Wikipedia dump and onto a broader network, to make the model more similar to a search engine. This way, the restrictions of the Wikipedia data dump are removed and a more truly open-ended question answering model can be utilized. Along with this, a next step would be to output more specific answers, rather than just the title of the page, perhaps output a paragraph or window in which the answer is within.