Matching Dispute Finder Claims to Wikipedia Articles

Michael Armbrust
`marmbrus@eecs.berkeley.edu`

Beth Trushkowsky
`trush@eecs.berkeley.edu`

Abstract

Dealing with large datasets is increasingly becoming a problem for natural language processing researchers. For our class project we investigate applying the opensource Hadoop MapReduce framework to the problem of information retrieval using TFIDF.

1 Introduction

It has been shown that one of the best ways to improve the accuracy of natural language processing algorithm is to supply more data for training (Norvig, 2009). However, unless the computational power of the tools used by researchers increases with the size of the data, processing time will increasingly become a limiting factor. For our project we investigate applying the map/reduce distributed processing paradigm to the NLP task of information retrieval.

Specifically we hoped to match claims from the website DisputeFinder[1] to relevant Wikipedia articles. In this paper we describe the methods we used to implement the TFIDF algorithm as a pipeline of successive Map/Reduce jobs. Additionally, we discuss improvements that we made and how they affected accuracy of the results.

2 Related Work

The most similar work to ours would be the Wikify! algorithm described in (Michalcea and Csomai, 2007). In this work, the authors extract important keywords from an input document, then link these keywords to Wikipedia pages. Possible keywords are the set of Wikipedia articles titles, with various morphological transformations. Their algorithm additionally uses wiki link text as a means for word sense disambiguation, which helps determine the correct article for a keyword. Matching keywords to articles is unsupervised, as in our approach, and the authors tried ranking articles with TFIDF and chi-squared measures However, they found the measure *keyphraseness* most effective: for a given keyword, how likely is it to appear as a link in the Wikipedia corpus. Our approach also uses TFIDF, but does not limit the potential keywords to titles of Wikipedia articles. We also do not do word sense disambiguation, as seeing the various senses of an entity might be beneficial to our goal of providing background information.

Providing links to Wikipedia articles is also described in (Milne and Witten, 2008). The goal of this work is to automatically enrich a document with links to Wikipedia articles. The key differences between this work and that of Wikify! is that (1) the former performs word sense disambiguation before keyword detection, and (2) the former trained a machine learning algorithm on the Wikipedia corpus to determine whether a keyword should be linked to (rather than simple link probability). This approach is different than ours due to its use of disambiguation and a supervised method for extracting keywords.

Finding related information using Wikipedia was explored in (Koolen and Kazai, 2009). While our approach aims to connect Dispute Finder claims to Wikipedia articles, this work connects related books to Wikipedia articles as a means of finding books for a user's query. Wikipedia is used as an intermediary between query terms and books, and the authors employ query expansion techniques. This work's goal was different than ours, but highlights the value of providing links between different corpuses of information.

3 Data Sets

The first dataset we used was a set of claims from the Dispute Finder website. This was provided as

[1] http://disputefinder.cs.berkeley.edu/

a JSON[2] file with each claim taking the form of a id and text tuple. The text for each claim is usually a single phrase, for example: "Global warming has natural causes" or "Penguins fall over onto their backs while trying to observe airplanes flying overhead." There are approximately 2000 such claims in the system. Of these, we labeled by hand 20 of them with up to 5 relevant articles as determined by human labeler.

Wikipedia provides dumps of the entire website for researchers who wish to do bulk processing of the corpus. These can be found on their download site[3]. These dumps come as large bziped XML files and the actual text of the article is stored in wiki-markup. They provide both a copy that has only the most recent revision (20 GB uncompressed) and a copy that contains revision history (2.8 TB uncompressed). For our experiments we used the version with only the most recent revision for our experiments, as this was already a lot of data. Also, past revisions often contain wiki-spam which could reduce accuracy.

4 Models and Methods

4.1 MapReduce and TF-IDF

In order to reason about which articles are relevant to a given claim, we compute the TF/IDF score for every unique word in every article on wikipedia. The input to our system is a raw wikipedia dump and a set of claims, both in the forms discussed in Section 3. The output is a set of up to five relevant article for each claims.

The first step was to take the raw data from wikipedia and input it into Hadoop. We created a custom loader that streamed over the compressed XML, extracting article titles and article text. These tuples were streamed into a SequenceFile[4] on Hadoop's distributed file system (HDFS) in the form of (Article Title, Raw Article Text). This allowed us to transform the data into a form that is easier to process without ever needing to fully decompress it on a single machine.

The next step is to clean the wiki markup from the raw text and produce a set of tokenized words for each article. We chose to separate this step from the previous so that we could easily try different cleaning methods without needing to repeat the relatively slow non-parallelized loading process. We applied a set of regular expression transformations to each article and the resulting output was a new sequence file in the form (Article Title, Article Tokens).

We compute two values from our cleaned article tokens. First, for every distinct word in wikipedia we compute the number of articles that contain that word. This is done by a simple map job that creates a set of distinct words for each article and then emits pairs with those words and the key, and a value of one. Both the reduce and the combiner simply sum the values for all equal keys and emit the result. The result from this step is of the form (Word, Number of articles on wikipedia that contains this word). We use a separate job that takes the many resulting files from this step and condenses them into a single MapFile which can be used to quickly lookup the value for any given word. While we tried a number of other alternatives, including sequential lookups or using an embedded database like the BDB JavaEngine[5], this method best balances the trade-off between index creation time and lookup time.

Next we compute the TF score for each (article,word) pair in wikipedia. This is done with a straightforward function that takes in (Article Title, Article Tokens) and emits (Word/Title, TF Score) pairs These are combined with the IDF value in the reduce function to produce the final (Word/TF Score, Title) pairs, which are stored in another Hadoop MapFile. Note that we have moved the TFIDF score from the value to the key, and moved the title to the value. This is due to the ordering provided by the output of a Hadoop job. We then used the resulting MapFile index to quickly locate the first entry for a given word. At this point, each article that contains this word will be sorted automatically by its TFIDF value.

We use exactly this technique to locate all of the words for a given claim and compute a complete score by summing the TFIDF scores for each article and for each word that the claim contains. As an approximation, we only look at the top 10 articles

[2]http://json.org/

[3]http://en.wikipedia.org/wiki/Wikipedia_database

[4]http://wiki.apache.org/hadoop/SequenceFile

[5]http://www.oracle.com/database/berkeley-db/je/index.html

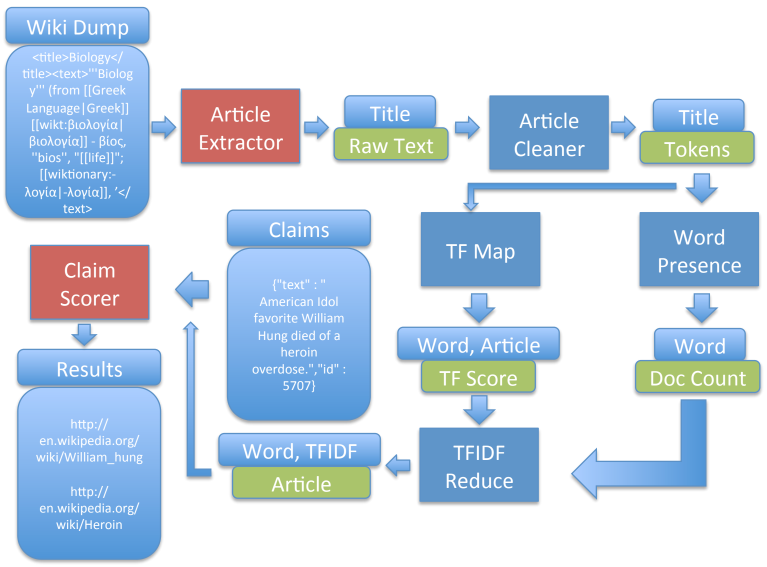
Wiki Dump
<title>Biology</title><text>'''Biology''' (from [[Greek Language|Greek]] [[wikt:βιολογία|βιολογία]] - βίος, "bios", "[[life]]"; [[wiktionary:-λογία|-λογία]], '</text>
Article Extractor
Title
Raw Text
Article Cleaner
Title
Tokens
TF Map
Word Presence
Word, Article
TF Score
Word
Doc Count
TFIDF Reduce
Word, TFIDF
Article
Claims
{"text" : " American Idol favorite William Hung died of a heroin overdose.","id" : 5707}
Claim Scorer
Results
http://en.wikipedia.org/wiki/William_hung
http://en.wikipedia.org/wiki/Heroin

Figure 1: The steps of the TFIDF pipeline. Red boxes represent tradition single node sequential jobs, and dark blue boxes represent Map/Reduce processing.

for any given word when computing this sum.

4.2 Improvements

4.2.1 Redirect Removal

After choosing articles for some claims with the results of the TFIDF MapReduce pipeline, we saw that many of the top results included Wikipedia redirect pages, which we decided to remove from consideration. These pages get high TFIDF scores because they are relatively short and contain a high proportion of the query term. Redirect pages are not necessarily incorrect, however they dominate the results set because there are so many redirect pages for the same article. For example "Corean" and "Korea'ng" are both redirected to "Korean"; we don't need all three in our results set.

4.2.2 Noun Extraction

The first attempt to use TFIDF for scoring wikipedia articles uses all words in the claim to compute the score for each article. We felt, however, that a more appropriate way to filter the articles would be by the noun phrases in each claim. This technique seems intuitive because our goal is to provide information on the entities and events within a claim, which are really those noun phrases. Noun phrase extraction is a type of information extraction problem, whose aim is to transform unstructured text into structured data. In general, the steps of information extraction include: (1) sentence segmentation, (2) tokenization, (3) part-of-speech tagging, and (4) entity detection.

We skip step (1) since each claim is already a single sentence, and leverage NLTK for steps (2-4). We tokenize by splitting each claim into an array of words, then tag each word with its part-of-speech with `nltk.pos_tag()`. The default part-of-speech tagger in NLTK works reasonably well. The next step is *chunking*, which will segment the claim into noun phrases. We define a set of rules, part-of-speech regular expressions, that define what it means to be a noun phrase. One rule defines a noun phrase as zero or one determiners, followed by zero or more adjectives, followed by one or more nouns: `<DT|PP$>?<JJ.*>*<NN.*>+`. This rule works great for claims like, "the tim hortons chain of coffee and baked goods stores adds nicotine to its coffee to keep customers hooked on it". Another rule we tried chunks sequences of proper nouns, `<NNP>+`, but that rule isn't effective when applied after converting the claim to lowercase.

After eyeballing the results of the chunking, we felt the essence of most of the claims was captured. The final transformation step was to remove stopwords from the claims. Most transformed claims we considered successful, e.g. the claim "cups of instant noodles pose a danger to consumers due to their wax linings" became "cups instant noodles danger consumers wax linings". An example of a less successful transformation was the claim "refried beans are beans that have been fried more than once" becoming "beans beans once". However, that issue would likely be resolved by training a part-of-speech tagger to understand that "refried beans" is an actual thing.

4.2.3 Using Article Titles

We initially thought that computing TFIDF scores for all the words in each Wikipedia article would lead to finding articles that talk a lot about the entities in the claims. However, Wikipedia article titles are also good descriptions of the content of the article. To compare how searching over the titles differs from searching over the whole article, we also computed TFIDF scores for just the words in the titles.

5 Evaluation and Results

To evaluate our algorithm, we hand-labeled 20 claims with 2-5 Wikipedia articles we felt would be meaningful for understanding the people and events discussed in the claim. We then ran the various versions of our algorithm, returning the top 5 scored documents for the 20 claims. We tried using both the full text of the articles and just the titles, with both the full tokenized text of the claims and the noun-extracted transformation of the claims. This creates four combinations that we evaluated: (articles,tokens), (articles,nouns), (titles,tokens), and (titles,nouns). Both noun-extraction of the claims and searching over articles titles provided better results.

The results are shown in Figure 2 and Table 1.We first calculated how many of the pre-selected relevant articles, the ones we hand-labeled, were returned. After looking at the results, we decided to also count how many of the returned articles were relevant to the claim, even if they were not in the original set of pre-selected relevant articles. The bar plot in Figure 2 shows both these counts. Table 1 shows the average percentage of returned articles from our pre-selected set, as well as the average percentage of returned articles that were relevant to the claims.

One baseline to compare our results to would be to randomly select five articles from the nine million possible articles for each claim. While not very meaningful, our initial algorithm using full articles and all the words in the claims actually performs just as poorly at finding relevant articles–with respect to our particular definition of relevant. The results do make sense from a search query perspective; they weren't totally random in that regard. See discussion below for a few examples.

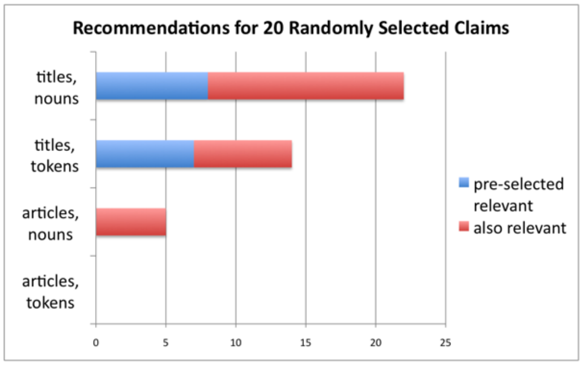

Figure 2: Relevant articles returned by the four algorithms. *articles,tokens* and *articles,nouns* used full article text with all claim words or with noun-extracted words, respectively. *titles,tokens* and *titles,nouns* used articles titles and all claim words or noun-extracted words.

Table 1: Average percent of the hand-labeled relevant articles returned, and average percent of returned articles that were relevant for the four techniques

	% of hand-labeled	% relevant
titles,nouns	12.3%	22%
titles,tokens	10.8%	14%
articles,nouns	0%	5%
articles,tokens	0%	0%

6 Discussion

One thing we noticed when seeing the results of using the full articles for TFIDF with all the words in the claim was that common words (not necessarily stop words) were giving very high scores to irrelevant articles. For instance, a claim about the black panthers yields the article "Black". Other times, high-scoring articles were reasonably returned based on their content, but probably dominated over actual relevant articles. This was likely the case for the returning the article "Buddhism in China", which mentions China frequently, for the claim "fortune cookies originated in china".

Our evaluation also revealed that the correct set of relevant articles for a claim isn't completely obvious or objective. For example, we pre-selected the article "Instant Noodles" for the claim "Cups of instant noodles pose a danger to consumers due to their wax linings". This article was not chosen by the algorithm, but the article "Cup Noodles" was selected. As hand-labelers, we simply didn't know that that article existed. Similarly, we pre-selected "California law" for a claim discussing a particular California state law, but saw in the results set the article "State law". While we did not initially choose the article for state law, it does provide insight on the notion of state law versus federal law, and provides links to articles about each state's laws.

7 Conclusion and Future Work

The goal of this work was to find Wikipedia articles for Dispute Finder claims in order to provide relevant background information. This proved to be a difficult task, both in computation and in evaluation. Using MapReduce in Hadoop was infinitely valuable in allowing us to try different algorithms in a reasonable amount of time and effort, due to its ability to quickly process a lot of data. Our first algorithm, searching for all words in a claim via a TFIDF index on the full text of the articles, returned results that made sense but were ultimately irrelevant for our task. The best algorithm transformed each claim by extracting noun phrases, and performed a TFIDF search over the articles' titles. This algorithm's results were 22% relevant, a significant increase from 0%. Still, these results could clearly be improved.

Leveraging page structure by searching over titles greatly improved our results, and we believe using other structural elements would provide additional improvement. Wikipedia articles have informative subsection titles, and the few sentences of a section are often informative as well. Extracting noun phrases from the claims helped remove irrelevant terms, but we did notice that the bag-of-words approach often finds unrelated articles with some key term (e.g. "China"). One possible extension to our algorithm would be to create a TFIDF index of bigrams or trigrams, rather than just unigrams.

References

Bird, Steven, and Klein, Ewan, and Loper, Edward Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. O'Reilly Media. 2009.

Koolen, Marijn and Kazai, Gabriella and Craswell, Nick. Wikipedia pages as entry points for book search. WSDM '09: Proceedings of the Second ACM International Conference on Web Search and Data Mining. 2009.

Mihalcea, Rada and Csomai, Andras. Wikify!: linking documents to encyclopedic knowledge. CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management. 2007.

Milne, David and Witten, Ian H. Learning to link with wikipedia. CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management. 2008.

Norvig, Peter Innovation in Search and Artificial Intelligence. CITRIS invited speaker. September 2009.