Implementation and Evaluation of LSH with News Articles

Devon Kohler
kohler.d@husky.neu.edu

Shaw Wen
wen.sha@huskey.neu.edu

Introduction

Finding similar itemsets is an incredibly important process in many fields. One area in which they are of particular use is for web based applications. Online companies can use similar items such as users, pages, and articles for applications such as a part of recommendation systems or serving advertisements. Because the internet is mainly text based, finding similar web pages or articles based on text similarity is a large area of study. In this paper we attempt to solve the problem of finding similar articles, with an emphasis on optimization of the algorithm and evaluation of the results.

The dataset used is from Kaggle.com[1]. It is a reservoir of 143,000 news articles collected from New York Times, Breitbart, CNN, Fox News, and others between the years 2016 and 2017. The articles in the dataset are already cleaned, which allows us to avoid issues that come up when web scraping articles such as including advertisements as part of the article. Additionally the dataset contains article titles, which helped in confirming similarity quickly.

An important aspect of this document corpus is that it contains a very large number of documents. This mirrors the issues companies would see when trying to determine similarity of articles from online sources. Because there are so many documents, it is not possible to calculate the similarity of all possible combinations. In order to determine similarity other techniques must be employed to determine similarity. In this paper we will be using Locality Sensitive Hashing (LSH) to determine similarity.

While LSH is already a well defined algorithm for finding similar documents, there are still optimization and evaluation issues that have not been completely explored. In terms of optimization, this paper will explore the best combination of parameters for LSH. These are the number of bands and rows, and shingling strategies. Shingling has the biggest variance here as there are many options for both length and text parsing that will greatly affect the final results. For evaluation, there is not a standard set of parameters we can check to ensure the algorithm is working correctly. The obvious evaluation technique is to simply look at the documents and manually check if they are actually similar. However, this is not scalable when the number of documents grows large. We will explore different options for evaluation which are scalable and do not require a manual check.

Related Work/ Background Information

The main area of work that is drawn on in this paper is in regards to the algorithm Locality Sensitive Hashing. In this section the algorithm will be explained. The first step in running LSH on a corpus is to shingle the documents according to some heuristic. Determining the best shingling is non-trivial and this is something we will explore in detail in this paper.

An important aspect of the LSH algorithm is that it only works with jaccard similarity (although the theory can be expanded to other similarity measurements). Jaccard similarity is calculated

by dividing the intersection by the union of two itemsets, in this case shingled documents. The official formula can be seen below.

$$Sim(D_1, D_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} \quad (1)$$

As mentioned in the introduction, calculating the jaccard similarity for every combination of documents is infeasible. To overcome this, first Min-hashing is implemented, then LSH is run on the results of the min-hash. To create the min-hash, first a certain number of hash functions are chosen. For each hash function, each row is hashed if the document contains the specific shingle in that row and the minimum value is kept to represent that hash. In the end the min-hash will be represented as a table with the rows corresponding to the number of hash functions and the columns corresponding to the number of documents.

Once the min-hash is created, LSH can be run. LSH works by splitting the min hash into a certain number of bands (b) and rows per band (r). For each band the rows in the band are hashed to a hash table with as many buckets as possible, to avoid collision. If two documents are identical in a particular band they will hash to the same bucket and then considered to be a “candidate pair”. The number of bands and rows can be tuned to increase the probability of seeing similar documents and decrease the probability of seeing dissimilar documents. This probability can be described in terms of the following formula where t is the jaccard similarity, b in the number of bands, and r is the number of rows.

$$\text{Prob Sharing Bucket} = 1 - (1 - t^r)^b \quad (2)$$

Using this and/or approach with rows and bands, where rows must all match in any particular band, gives an S-shaped probability curve of seeing any particular document pair. The choice of b and r is critical to get the steepest possible curve at the desired similarity threshold. For example if a similarity threshold of .55 was desired we would use a band and row combination of b = 20 and r = 5. This threshold is determined by the following formula.

$$\text{Threshold} \sim \sqrt[r]{1/b} \quad (3)$$

When choosing the bands and rows to determine a threshold, it is important to take into account the false positive and false negatives returned by the algorithm. For false positives, a simple post processing check can be run on the candidate pairs to ensure that they are truly similar. However for the false negatives, no such test can be run. Thus it is extremely important to choose a threshold a bit lower than the similarity we are actually looking for, in order to reduce the number of false negatives as much as possible.

Proposed Approach

In terms of optimizing LSH, the first step is determining the shingling. While the documents in the dataset were already clean, we did not want capitalized letters or punctuation to effect the similarity results. Without fixing this the overall similarity of the documents would be lower than it truly should be. Thus all punctuation was removed and the letters were converted to lowercase before any shingling strategies were attempted. Additionally the python text parsing library NLTK was used to reduce words to their base state. For example a word ending -ing such as “writing” would be converted to simply “write”. This was done in order to reflect the true similarity of the documents. These words still have the same meaning behind them, even if they are in a slightly different form. Finally, after testing, it was decided that any duplicate shingles should be removed resulting in a binary array based on if the shingle appeared or not. This was done partially due to the uneven length of the documents.

The most import aspect of shingling is determining the length of the shingle. The length of the shingle should be long enough that the probability of any shingle appearing in any given document is low. This depends both on the length of the alphabet, in this case the number of possible words used, as well as the length of the documents. The articles in the dataset have an average of about 500 words per document, although some were much longer. We defined length to be the number of k words in the shingle. Four different values of k were tested: 1, 2, 3, 5. Some of the smaller values for k may make the probability of seeing a given shingle too high, while a very large k could make the probability too low and very few similar documents would be returned.

Another important shingling strategy was using stop words. Stop words are generally defined as words which are the most common in a language. In the case of finding similarity, stop words which appear in all documents will drive up the similarity between all documents. In the case of this project three different strategies for dealing with stop words were defined. These strategies were: keeping the stop words, removing the stop words entirely, and creating shingles only when seeing a stop word. The impact of the first two cases should be fairly straight forward, we would expect to see an uptick in similar documents with stop words than with them removed. The last strategy is generally used on webpages to avoid shingling advertisements. Our documents do not contain advertisements so the effect of this shingling strategy is unclear.

The next step of our project goes into the evaluation of the LSH results. The first evaluation technique is to simply look into the candidate pairs and confirm they are similar. This technique will be employed to double check the results, however there is not much more to say about the approach.

Initially we tried to validate the similarity between similar documents returned by our LSH function by directly comparing them using jaccard similarity, to our dismay, the precision rates were disappointing. After further research, we found out that the culprit wasn't with our LSH algorithm but with the way we were validating them. And that is what led us to embedding. Among the many advantages that embedding can offer, one of its major benefits in the context of NLP is the incorporation of meanings when comparing text. Instead of just comparing alphabets, text embedding in similarity matrix along with a proper distance measure allows us to detect similarity based on the meaning of the text. And in our case, we have chosen to use the soft cosine distance measure in lieu of the traditional or "hard" cosine because of soft cosine's ability to capture similarity of features in vector space model while the former does not. The two work perfectly in sync because embedding essentially projects texts from higher dimensional features into lower dimensional vector space, while the soft cosine measure can capture feature similarities in that vector space.

Another evaluation technique is to cluster the documents returned as similar. The idea for this evaluation is that the documents which are similar should generally be in the same cluster. If this was the case we would have more confidence that the LSH results were actually similar; essentially having another algorithm confirm the output of LSH. Additionally we should be able to see some trends within the clusters. One issue with this approach is that it requires effective clustering so there is another point where an issue could arise. In this paper a simple k-means clustering was used, although other algorithms were tested. The number of clusters was selected using the elbow method, looking at the decrease in SSE (sum of squared error) with each increase in cluster size and choosing the number where SSE started decreasing significantly less.

The final evaluation method was to look at the ideology, or political bias, of the documents.

The idea behind this is similar to the clustering evaluation. We would expect similar ideology based documents to cluster together, which is another indication that LSH is working well. This analysis assumes the documents show some ideological bias simply through the shingles which may or may not be the case. As mentioned, this is similar to clustering, however here the groups are given specific labels based on the leaning of the news source, left/right/center. These labels may help describe the results easier.

Experiments

After testing many different combinations of bands and rows, the final values used were a combination of 24 bands and 5 rows while looking for documents which were similar at a value of .6. This combination of bands and rows gives a threshold value of .5296, which is significantly lower than .6. This was done in an attempt to reduce the number of false negatives which can not be fixed in post processing. The final S-curve can be seen below.

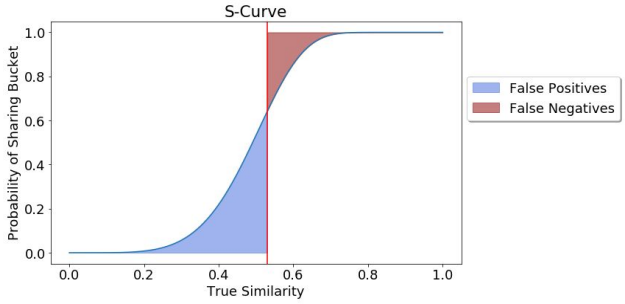

Figure 1: Final S-Curve

In terms of the three different stop word procedures we saw straightforward results. With stop words included more candidate pairs were returned, no matter the length of the shingle. It was determined that these additional candidate pairs were due to shingles which did not have much meaning to the overall documents. Thus the approach we chose was to remove stop words. Finally there was not a significant difference when removing stop words vs creating the shingles based on the stop words.

In terms of shingling length there were very different results per value of k. At a shingle length of 1 word there was an enormous number of candidate pairs returned, over 200,000 pairs. Due to the processing power available, this many pairs is too large to run a check for false positives. Additionally, with such a large number of similar documents the actual usefulness is questionable.

Conversely with the larger shingles of 3 and 5 word lengths there were very few candidate pairs returned, around 10,000 or fewer. This could be attributed to the majority of the articles being generally of shorter length (~500 words) and when the shingles get too large the probability of seeing any given shingle in a given document is extremely low.

In the end the 2 length shingle was determined to be the best length. The number of candidate pairs was large enough to be useful, but not too large that the false positive check could not be run. The final number of candidate pairs was around 25,000. For some applications this number of candidate pairs may be considered too low, however a simple adjustment of the threshold would yield more pairs, and there would not be the extremes seen in the other shingle lengths. The final candidate pair counts for the different shingle lengths can be seen below.

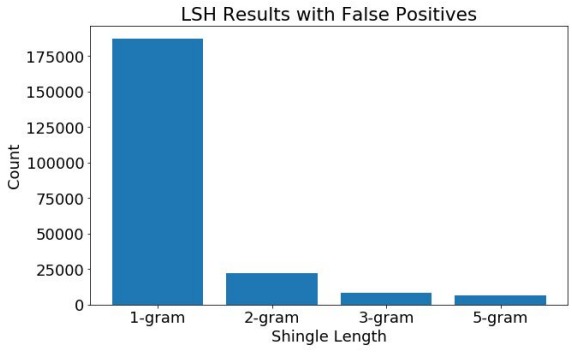

Figure 2: Shingle Counts by Length

After determining which shingle length was appropriate, a check on the false positives was run. After running the check we ended up with a total of about 5000 shingles. This false positive rate is much higher than we originally expected. An explanation for this is that there are inherently a very large amount of similar documents. Even at low levels of similarity there is a chance of pairs sharing a bucket in LSH. If there are a lot of low similarity pairs, the chance that some of these slip through is high.

For the evaluation techniques we saw many interesting results. First we simply checked the candidate pairs manually and made sure they were actually similar. One example of the observed similarity was in the pair with titles: “Complete Text of Inaugural Address” and “Trump Brings the Rain in Inauguration Address”. One article was simply Trump’s inaugural address, while the other had many quotes from the address, but included more detail and was not a simple transcript. Obviously both of these documents are strongly similar, with jaccard similarity of .73, and the algorithm can be evaluated as working well in this way.

With a threshold of 0.9, the precision rate according to our embedding and soft cosine validation method yields a range of accuracies between 85% and 96% with standard deviation of 0.8812. Below is a breakdown of the precision rates by various combinations of n-gram and stop words. And as demonstrated by barplot below, the level of accuracy increases commensurately with the number of grams.

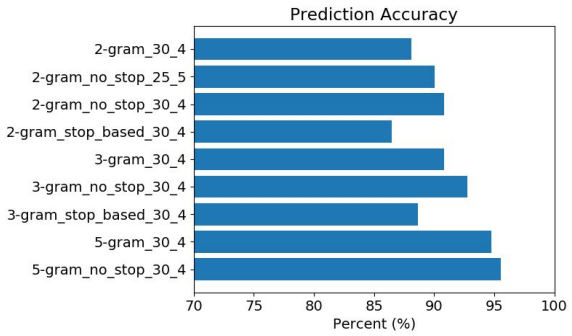

Figure 3: True Positive Rate

The clustering evaluation showed very interesting results. The majority of similar pairs returned by LSH were clustered together by the K-means clustering. About 76% of the pairs LSH returned as similar were clustered together by K-means. This shows that K-means picked up on the same similarity trends that LSH noticed. Using clustering allows us to increase the confidence in the results of the LSH without having to go into each pair and manually check them. With an extra confirmation by clustering we can assume that the results of LSH were correct and the pairs are truly similar.

With the political bias analysis the results were less conclusive. As noted earlier, this was another way to confirm the results of LSH were correct without having to look at each candidate pair. While the majority of candidate pairs were grouped according to ideology, about 91%, the majority of our documents were either left or center leaning. Due to the low amount of right leaning similar pairs, it is extremely hard to make any conclusions about the effectiveness of LSH from this analysis.

Conclusions and Future Work

As discussed in the previous sections, the biggest change in terms of optimization of LSH comes in the choice of shingling. Changing the length of the shingles and deciding how to deal with stop words produced very different amounts of candidate pairs. The best combination of parameters, two word shingles with stop words removed, should give future

work a solid starting point for expansion and additional research into this problem.

An expansion into the shingling area of this project is to use additional text parsing. As mentioned, the punctuation was removed, the letters were converted to lowercase, and words were reduced to their base, in an effort to bring out the true similarity of the documents. There are many libraries, in particular the nltk library in python, which can take a word and convert it to a simpler word with the same meaning. Doing this would definitely increase the number of similar documents, so additional tweaking to the band and rows may be needed, but it would be interesting to see if this allowed for better identification of similar documents.

One straightforward expansion to this paper is to test the methods we used with a different similarity measure. While the combination of min-hashing and LSH can only be used with jaccard similarity, the theory behind LSH, an S-curve created created by and/or methodology, can be used for other measures such as cosine and euclidean distance. Cosine distance in particular could be interesting. As mentioned briefly in the methods section, when creating shingles we used unique shingles only, not including duplicates. With cosine similarity we would have more options when converting text to vectors. One option would be to use TF-IDF which may be able to capture which words are most important to each document. This may yield better results compared to simply taking all words as equals.

Finally there are a number of expansions that could be done to the evaluation methods. In terms of clustering, it would be very interesting if we could confirm why some similar documents are clustered together and others are not. If the documents which are not clustered together are not similar, it may be best to run LSH in two steps: first normal LSH and second clustering to narrow down the results. Using and expanding on these methods is important in order to confirm the results of LSH in a scalable way.

References

1. https://www.kaggle.com/snapcrack/all-the-news
2. https://www.machinelearningplus.com/nlp/cosine-similarity/
3. https://medium.com/@Hironsan/why-is-word-embeddings-important-for-natural-language-processing-6b69dd384a77
4. http://www.scielo.org.mx/pdf/cys/v18n3/v18n3a7.pdf