Efficiently Finding Similar Reviews in Amazon Reviews dataset using Hashing Techniques

Tejas Srivastava

February 23, 2020

Download and clean the data

1. Read the data in pandas dataframe, as it is.
2. Dropped irrelevant columns.
3. Removed stop words by using replace function, and creating a regex matching dictionary of all the stop words. Converted the reviewText to lowercase, and removed all punctuations. Further, split all the words and joined them by a single space to replace multiple spaces by only a single space.
4. Remove all the reviews which had a length less than the chosen value of 'k'.

Analyze randomly selected 10,000 reviews and find the jaccard distance, and plot the histogram of Jaccard distance for the pairs

1. Defined a character mapping, where each valid character (alphanumeric and space) would map to a unique index between 0 to 36. (a-z:0-25, 0-9:26-35, " ":36).
2. Further chose a value of k, and for each k length string (shingle) in the reviews, calculated the position the shingle would take if all k length shingles had been sorted according to the mapping, thus getting a unique value for each shingle.(function find_index_in_binary_matrix)
   For example, if k=4, string 'aaaa' would get index 0, string and so on.
   So, basically value for each shingle would be calculated as follows, considering shingle of length k:

   37^{k-1}*mapping(shingle[k-1])+37^{k-2}*mapping(shingle[k-2])+....+37^{0}*mapping(shingle[0])

   This would give us the unique row index of each shingle in the representation of the reviews in the form of a binary matrix.
3. **Further, this function was applied on the whole pandas dataframe, and the reviews were converted into a sparse(Scipy coo matrix) matrix of the shape (37^4,157680), where rows represent the shingles and each column represents a review. A value of 1 in the matrix indicates that the given shingle(row) is present in the given document (column).**

4. The **value of k** was chosen to be **4**, since larger values of k gave us very less similarity in similar reviews (decreasing the true positives) whereas smaller values of k, gave a lot of reviews which were not similar as similar (increasing the number of false positives). Thus the value of k was chosen to be **4**.

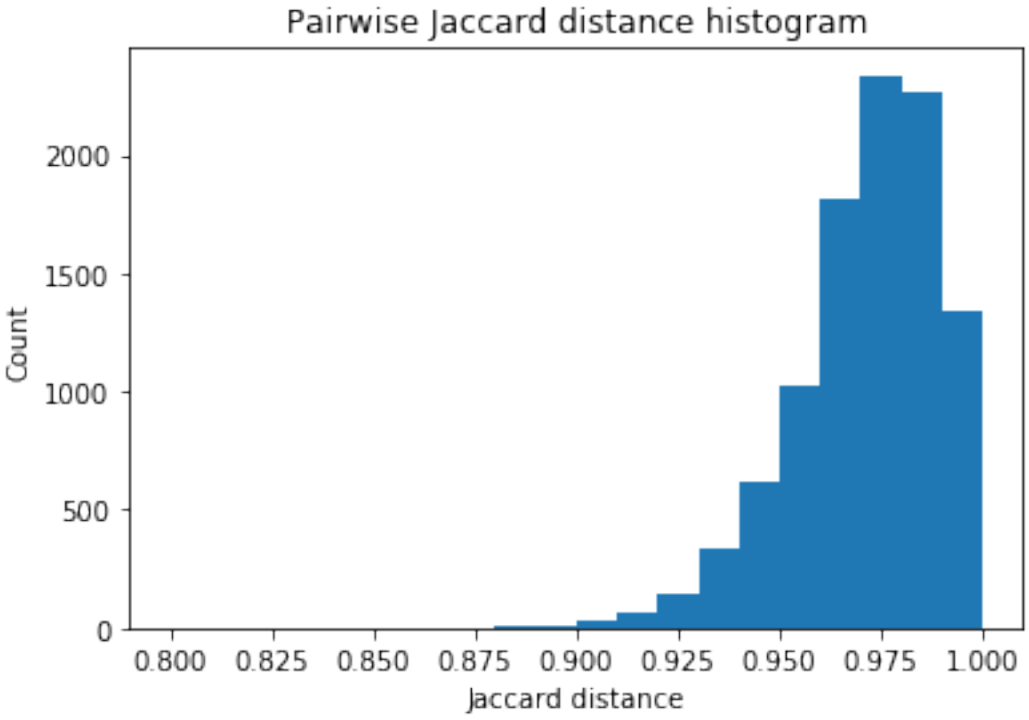
5. **For efficiently finding the distance between the reviews, we further converted out sparse matrix in a dictionary of size same as the number of reviews (157680), where key was the review index and each value contained a set of non-zero row indices for that review (column). This made calculation of jaccard distance very fast and efficient.**

6. Next, about 142 random reviews were chosen out of all of the reviews, and jaccard distances was found for all the 10,000 pairs of reviews which were made by the possible combinations of the 142 reviews. The jaccard distance was calculated by calculating the length of set intersection and length of set union of the non zero positions(row index) in each document, and then using the formula:

$$dist = 1 - (sizeof(intersection)/sizeof(union))$$

7. The **minimum Jaccard distance** in the random 10,000 samples was **0.8584** and the **average jaccard distance** was **0.9709**.

8. The distribution obtained was as follows:



Find a better representation for more efficient querying, and for further operation of finding similar reviews

1. We further convert the binary matrix representation into a signature matrix, by applying the technique of **Min-Hashing**.

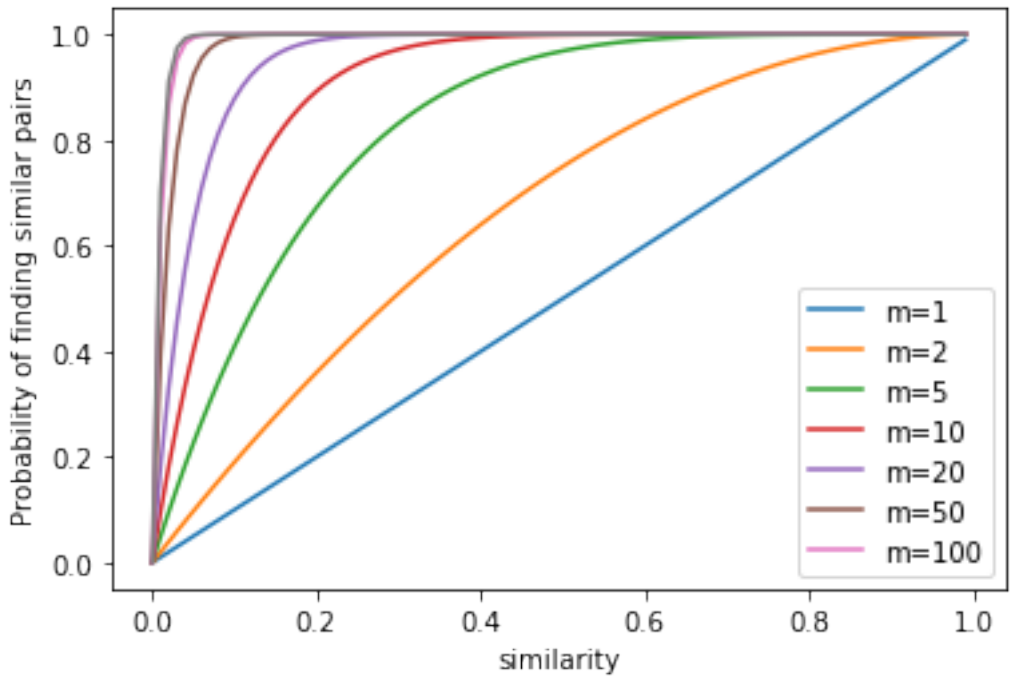
   - We permute the indexes of each row using a hash function of the form

     $$(a * (row\_index) + b)\%p$$

     where a and b are randomly generated numbers in the range of (0,2000) and p is a prime number greater than the number of rows in the binary matrix(37^4 in our case).
   - The hash value of each column is the minimum index generated for that column, thus giving us one hash value for each document.
   - Further **120** such random hashes are applied, to obtain better results.
   - Increasing the value of m (the number of hashes) gives us better chances of finding documents with a given similarity. This is because the probability of finding similar documents for a given value of similarity is given by

     $$Pr(hit) = 1 - (1 - similarity)^m$$

     where similarity is the similarity for two documents to be similar, and m is the number of min hashing functions applied.
   - This is evident from the following graph :



2. Applying **multiple min-hashing** increases the probability of finding similar documents (that too in an efficient manner), but it also **increases the probability of finding false positives**, which is evident from the graph above.

3. Thus, we further apply **Local Sensitive Hashing**.

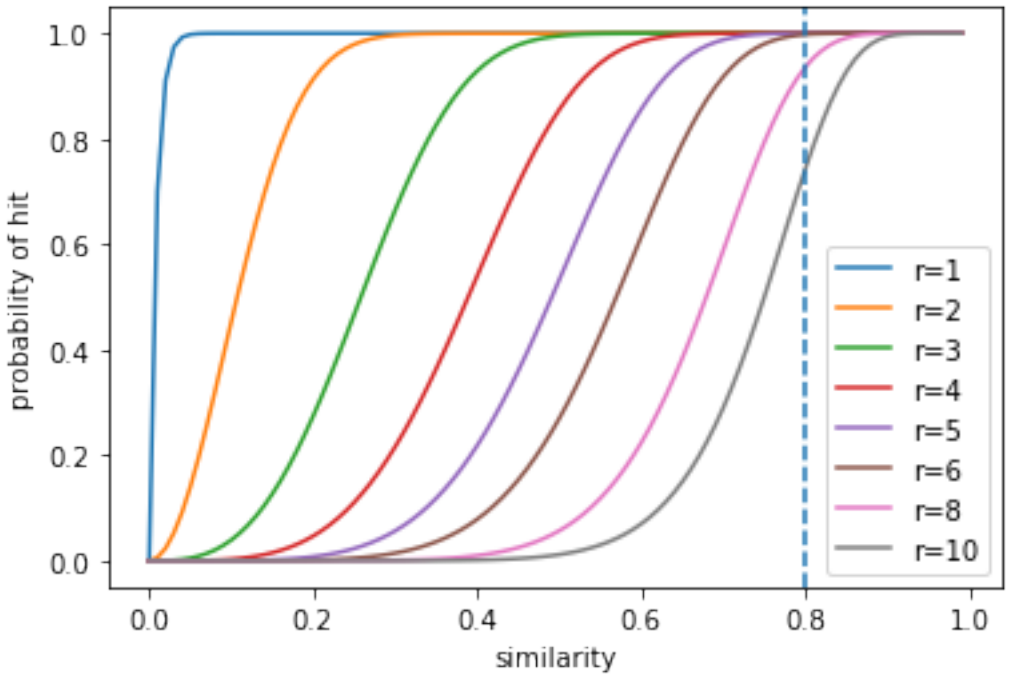
   - We split our signature matrix (shape: 120,157680) into 'b' bands, where each bands consist of 'r' rows.
   - Further, for same hash function is applied to corresponding rows of all the bands, But for each row in a band, the hash functions applied are all different.
   - The hash function applied is again $(a * x + b)\%p$, where 'a' and 'b' are randomly selected large numbers, and p is a very large prime number (982451653 in our case).
   - Further the values in each band are summed up and thus, we get a hashed signature matrix of the shape ('b',157680), where 'b' is the number of bands selected and 157680 is the number of reviews.
   - We definitely know that applying local sensitive hashing **decreases the number of false positives** for our estimate. This is because the probability of finding similar documents for a given similarity after applying this techniques would be given by

     $$Pr(hit) = 1 - (1 - similarity^r)^b$$

     , where b is the number of bands the signature matrix is divided into, and r is the number of rows in each band. This would become more clear in the plots in the next part.

4. **Choosing the values of 'r' & 'b'**

   - We chose the value of m=120 in the previous step, but for choosing the values of r and b,we list the various factors of 'm', and try to plot the graph of we plot the following graph:



- As we see from the graph above, applying local sensitive hashing results in the graph being changed, and we surely decrease the number of false positives. But we also see that by choosing different values of 'r' and 'b', we get different results, and different number of reviews for a given similarity.
- In our problem statement, the assumption was that two documents are considered similar if they have a similarity greater than 0.8.
- So, for a value of similarity of 0.8, we see from the graph above, that if we choose the value of 'r'=10 (and 'b'=12), we get the grey graph, which misses on some documents which have a similarity greater than "0.8", thus it rejects some true positives.
- Also, if we choose the value of 'r'=3 (and 'b'=40), we get the green curve, which has all the documents with similarity greater than 0.8, but it also has a large number of documents whose similarity is less than 0.8.
- Thus, we need to choose the value of 'r' and 'b' properly, so that we do not miss out on any true positives, but also do not have many false positives. (We can filter out the false positives easily is they are lesser in number). Thus, we choose the value of **'r'=6 & 'b'=20** for our case, as it does not miss out on any true positives, and also has a less number of false negatives.

5. Thus, we choose the mentioned representation of reviews so as to find similar pairs of reviews in an efficient and a correct manner.

Detect all pairs of reviews that are close to one another using the selected representation

1. Further, after we have the hashed signature matrix representation (shape: 'b',157680) after applying local sensitive hashing, we say that two reviews are similar if any of their corresponding row values are equal in this representation.

2. Thus, we then make a list of all similar pairs of documents, by checking if any values are same in row of any documents, (storing them in a list of tuples), and further taking out the unique pairs and sorting them.

3. Thus, we now reduce our set to a list of 9341 pairs of similar documents. But we know from the plot in the previous section that these also contain some false positives, which actually have a similarity of less than 0.8, but still have been counted as similar.

4. So, it is still easy to actually calculate the distance between them using the hashed representation and then filter out those pairs which have a distance greater than 0.2, since we have less number of pairs than the very initial stage (without considering any similar documents, we had about ${}^{157680}C_2$ pairs).

5. Thus, we filter out those reviews whose distance is greater than 0.2, by constructing a mask for the list, by using map function (which returns 1 if jaccard distance is less than 0.2, and returns 0 if it is greater than 0.2). Thus we get the list of final pairs(about **1112 pairs**) of documents, which actually have distance less than 0.2.

6. We use the pairs to write the reviewText and reviewerID from the original dataframe to a CSV file.

Given a review return its approximate nearest neighbour

1. The function takes in a new review as a string, further applies the same cleaning and pre-processing which was applied to all our reviews. Further it is also similarly converted to a binary matrix representation in a similar value as all the reviews.

2. Next, it is converted into signature matrix, and hashed signature matrix, by applying the same procedures,and more **importantly, using the same values of parameters for the hash functions**.

3. The row values in the new reviews representation is matched with the other reviews, and even if anyone corresponding row values match, we say that the new review and the other review are similar.

4. Further, we determine the actual jaccard distance from the similar reviews, to filter out false positives and find the nearest neighbour with the minimum jaccard distance, and write that to a separate CSV file (NearestNeighbor.csv).

Complexity of this approach as compared to Naive Approach

1. For the problem of finding similar pairs of reviews, the naive approach basically involves calculating the distance between all possible pairs, which basically involves comparisons of the order O(n^2), as there are ${}^nC_2$ pairs possible in all (if we consider calculation of distance between two reviews as $O(1)$ operation).
   Thus,the naive approach is very inefficient and computationally expensive for large amounts of data or large number of reviews.

2. The approach of min hashing and local sensitive hashing used here, instead, is much efficient, as it only involves direct computation of distance with a much lesser number of reviews (approximate similar reviews), and finds the approximate similar pairs and can be considered of the order $O(n)$.

3. This is evident from our example itself where the number of reviews were of the order of 150,000 (157,680 to be exact). So if the naive approach had been followed, it would take about 12,431,412,360(${}^{157680}C_2$) comparison, whereas after applying min hashing and local sensitive hashing on all these reviews and converting into signature matrix and hashed (banded) signature matrix, the number of similar pairs was reduced to only 9341, thus we had to do only 9341 comparisons to find the similar reviews. And the computation was reduced by the order of 10^8. Thus, this approach is much efficient.

4. For the problem of finding the most similar document also, this technique is much faster since it only involves comparison with the documents which have the same bucket values (after applying both the hashing techniques and converting to hashed signature matrix), whereas the naive approach involves calculating the distance by every other document.

Thus, we conclude that applying techniques such as min hashing and local sensitive hashing give us a fast, efficient and less computationally approach for finding similar data points. This is specifically useful when dealing with sparse, very high dimensional data, and when there are a large number of data points to compare with, as approximate similarity can be used in such cases to decrease the run time.