Finding Similar Items

Francesca Ferraro 16125A

Contents

Description : "The task is to implement from scratch a detector of pairs of similar tweets, considering the text field of the dataset. You are free to preprocess the textual data using the techniques which you think are more appropriate, as well as to apply any of the methods explained during the lectures (or, of course, two or more methods)."

Declaration : I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

1 Dataset Loading

1.1 The Yelp dataset

The project refer to the «Yelp» dataset, published on Kaggle and released under the CC-BY-SA 4.0 license. Specifically, the project is focused on the analysis of the reviews contained in the JSON file containing the information about the reviews, including their texts.

The access to the Yelp datasets has been done on May 2023, therefore the version used in this project is the last updated by the dataset providers.

Json-encoded data has been transformed into Pandas Dataframes, and the considered column of the data for the goal of the project is the "text" field.

2 Data Preprocessing

In the first part of my project I applied all the preprocessing techniques needed to transform raw data into a clean and structured format, ready to be processed in a simple way.

2.1 Tokenization

The process of tokenization consists in splitting the text into smaller units, that in this project are words. These tokens will be used as the basic building blocks for further analysis and processing.

2.2 Stopwords removal

Here I used a stopwords list from Kaggle, that contains all the most common words in english grammar, that can be excluded from the tokens of the sentences in order to focus the analysis only on the relevant words. So I obtained a data structure that contains all the single words without punctuation, doesn't consider blank spaces, takes the words case unsensitive and doesn't consider stopwords.

2.3 Stemming

The purpose of stemming is to simplify words and reduce them to a common form. In English, words often exist in different forms (e.g., plurals, verb conjugations, tenses). Stemming algorithms are used to remove prefixes, suffixes, and other affixes from words to obtain their root form.

I choose to do stemming (with some Porter stemming rules) instead of lemmatization, because, even if lemmatization produces base forms that are valid words, while stemming may generate stems that are not necessarily valid words, it is computationally more expensive compared to stemming. I think that in this context it isn't necessary to have the exact valid words, but is important that the algorithm is able to understand the differences.

3 Scalable Approach

Scalability is crucial for making algorithms usable with large datasets. As datasets grow in size, traditional algorithms and approaches may struggle to handle the increased computational requirements and may become inefficient or impractical to use.

For this section of the project, in order to avoid problems linked to the large size of the dataset, I chose to use only a small fraction of the entire dataset (in particular, I used 69902 items that corresponds to the 1% of the dataset original size).

3.1 K-shingling

In this scalable version of the project, k-shingling is a crucial step. A shingle is a contiguous sequence of k words that allow to represent a text document as a set of shingles. In this project, every review has been splitted into substrings of length k.

3.2 Min-hashing

This technique consists on the compression of the set of shingles by computing the minimum hash codes for each item in the sets.

In my project I used a function that takes the hash values array, a shingle, and the number of permutations as input, then iterates over each permutation and computes the hash value for the combination of the permutation index and the shingle using the SHA-1 hashing algorithm (that takes an input message of any length and produces a fixed-size 160-bit hash value, typically represented as a 40-character hexadecimal number). If the computed hash value is smaller than the current minimum hash value stored in the array, it updates the array with the new minimum hash value. The hash value for the combination of the permutation takes index and the shingle in order to introduce randomness and reduce collisions in the hash values generated for different shingles.

Finally, an array of signatures is created for store all the compact representation of sets that enable efficient similarity estimation.

3.3 Locality-sensitive hashing (LSH)

By applying LSH, the algorithm can efficiently identify sets of shingles that are likely to have high Jaccard similarity, reducing the computational complexity of calculating pairwise similarities for the entire dataset. The LSH algorithm uses a technique called "banding" to divide the minhash signatures into multiple bands. For each minhash signature, it iterates over the bands. Within each band, a key is created by grouping the minhash values at corresponding positions in the signature. This key is a tuple representation of the minhash values for that particular band.

If the key does not exist in the buckets dictionary, a new set is created and associated with that key. Otherwise, if the key already exists, the index is added to the corresponding set in the buckets dictionary.

Finally, the algorithm returns the buckets dictionary, which contains the grouping of similar items based on their minhash signatures.

3.4 Jaccard similarity

Jaccard similarity is a measure used to determine the similarity between two sets. It is calculated by dividing the size of the intersection of the sets by the size of their union.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

In my project, the Jaccard similarity is used within a loop over the buckets obtained from the LSH step. For each bucket, it checks if the length of the bucket is greater than 1, indicating that there are multiple shingle sets in the bucket. It then iterates over the pairs of shingle sets within the bucket and calculates the Jaccard similarity. If the similarity value falls within a specified threshold range ($0.5 <$ similarity < 1) and the index of the first shingle set is less than the index of the second shingle set (ensuring that each pair of similar shingle sets is considered only once), it adds the pair to the similarity list.

The resulting list contains tuples of similar review pairs, along with their corresponding similarity scores.

4 "Brute Force" Approach

In this second version of the project, that was the first I made (using a definitely not scalable approach), I tried to apply row distances directly to the vectorized and normalized list of tokens.

As it can easily be predicted, this wasn't the best approach, because of the missing ability of the algorithm to scale with the increasing of the dataset size. For this part of the project, in effect, I had to use a very small subset of the data (I used only 1500 items, over the 6990280 total items in the original Yelp dataset).

I have decided to include this part as well, even though it is not the correct approach, for showing the differences with the first approach I presented and also for make it to be an exercise for experimenting with different vectorization and distance metrics.

4.1 Vectorization using Bag-of-Words (BoW)

Vectorization using Bag-of-Words (BoW) represents text documents as vectors, where each dimension corresponds to a unique word in the corpus. The value in each dimension represents the frequency or presence of the word in the document. This approach disregards the order of words and focuses on their occurrence.

The formula for BoW representation is:

$$\text{BoW}(d, w) = \text{frequency of word } w \text{ in document } d$$

where d represents the document and w represents the word.

4.2 Vectorization using TF-IDF

Vectorization using TF-IDF (Term Frequency-Inverse Document Frequency) assigns weights to words in a document based on their frequency in the document and their importance in the entire corpus. It aims to highlight words that are more discriminative for a particular document compared to the entire corpus.

The formula for TF-IDF representation is:

$$\text{TF-IDF}(d, w) = \text{TF}(d, w) \times \text{IDF}(w)$$

where $\text{TF}(d, w)$ represents the term frequency of word w in document d, and $\text{IDF}(w)$ represents the inverse document frequency of word w.

The term frequency $\text{TF}(d, w)$ measures how often word w appears in document d. The inverse document frequency $\text{IDF}(w)$ measures the rarity of word w across the corpus, giving more weight to words that appear less frequently in the corpus.

The TF-IDF representation provides a numerical representation of the importance of words in a document, considering both their local and global significance.

4.3 Normalization

Normalization is applied to the vectorized representations of the sentences to ensure that they have a consistent scale and to prevent the dominance of certain features based on their magnitudes. In this project, the L2-norm normalization technique is used, which scales each vector to have a unit length.

The formula for L2-norm normalization is:

$$\|\mathbf{w}\|^2 = \textstyle\sum_{i=1}^{n} w_i^2$$

The normalization step allows for fair comparison and similarity calculations across documents, as it eliminates the impact of varying vector magnitudes.

All the distance measures used in this project, including Jaccard distance, Cosine distance, and Euclidean distance, have been applied to the normalized vector for both the BoW and TF-IDF representations.

4.4 Cosine distance

$$\text{Cosine distance} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \cdot \|\mathbf{B}\|}$$

Cosine distance measures the similarity between two vectors based on the cosine of the angle between them. It is calculated by taking the dot product of the vectors and dividing it by the product of their magnitudes. A smaller cosine distance indicates higher similarity between the vectors.

4.5 Jaccard distance

$$\text{Jaccard distance} = 1 - \frac{|\mathbf{A} \cap \mathbf{B}|}{|\mathbf{A} \cup \mathbf{B}|}$$

Jaccard distance measures the dissimilarity between two sets. Jaccard distance is commonly used for comparing the similarity of binary data, such as presence/absence of features.

4.6 Euclidean distance

$$\text{Euclidean distance} = \sqrt{\sum_{i=1}^{n} (\mathbf{A}_i - \mathbf{B}_i)^2}$$

Euclidean distance calculates the straight-line distance between two vectors in the vector space. It is computed as the square root of the sum of squared differences between corresponding elements in the vectors. Euclidean distance is widely used for measuring the dissimilarity between numeric data.

4.7 Edit distance

Edit distance, also known as Levenshtein distance, measures the minimum number of operations (insertions, deletions, substitutions) required to transform one string into another. It quantifies the dissimilarity between two strings based on their character-level differences. In my project I used an optimized version of the Edit distance, computed by the Wagner-Fischer algorithm.

4.8 Plotted histograms

For analyzing the results I decided to plot the histograms to have visual representations of the distributions of similarity or distance scores. Histograms illustrate the frequency of scores falling within certain ranges, allowing for analysis of patterns and identifying any notable trends or outliers.

5 Results

The experiments in my project help to show that there is a big difference between using a sclalable approach and using a non-scalable one.

The benefits of a scalable approach, that in this case was made with minhashing and LSH for finding similar items, but that can easily be applyied to a lot of other Natural Language Processing problems, include :

- Managing Large Data: Scalable approaches can efficiently process and analyze large volumes of data.
- Efficiency: Scalable approaches are optimized for faster execution, reducing computational time.
- Flexibility: Scalable approaches can be applied to various data types, formats, and problem domains.
- Real-World Applicability: These approaches enable accurate and comprehensive analysis of large textual datasets, facilitating practical applications in real-world scenarios.