Algorithms for Data Science: Finding Similar Items

Pablo Mollá Chárlez

September 19, 2024

Contents

1 Introduction

Data mining often involves identifying similar items within large datasets. This task is similar to finding near-neighbors in high-dimensional space, where each data point is represented as a vector in a multi-dimensional space. Efficiently identifying similar items has a wide range of applications, including:

- **Duplicate Detection for Search Engines**: Identifying web pages with similar content to improve search results.
- **Customer Purchase Patterns**: Recognizing customers who purchase similar sets of products to tailor marketing strategies.
- **Image Recognition**: Finding images with similar features for tasks like facial recognition or product recommendation.

2 Similarity and Distance Measures

To find similar items, we first need to define what it means for two data points to be "close" to each other. This involves:

- **Input**: A set of high-dimensional data points represented as vectors $\mathbf{x} = (x_1, x_2, x_3, \ldots, x_n)$ and a distance function $d(\mathbf{p_i}, \mathbf{p_j})$ that measures the distance between two points $\mathbf{p_i}$ and $\mathbf{p_j}$.
- **Problem**: Identify all pairs of data points $(\mathbf{p_i}, \mathbf{p_j})$ such that the distance $d(\mathbf{p_i}, \mathbf{p_j}) \leq \tau$, where τ is a predefined threshold.

A naive approach would involve comparing all possible pairs of data points, resulting in a time complexity of $O(N^2)$, where N is the number of data points. However, more sophisticated algorithms can reduce this complexity to approximately $O(N)$, making the process feasible for large datasets.

2.1 Jaccard Similarity and Distance

One effective way to measure similarity between sets is the Jaccard Similarity. It is particularly useful when dealing with sets of items, such as words in documents or features in images.

2.1.0.1 Jaccard Similarity

The Jaccard Similarity between two sets A and B is defined as:

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

For instance, let's consider two sets: $A = \{\text{apple}, \text{banana}, \text{cherry}\}$ and $B = \{\text{banana}, \text{cherry}, \text{date}, \text{fig}\}$. The intersection $A \cap B = \{\text{banana}, \text{cherry}\}$ and the union $A \cup B = \{\text{apple}, \text{banana}, \text{cherry}, \text{date}, \text{fig}\}$. Thus,

$$J(A, B) = \frac{2}{5} = 0.4$$

2.1.0.2 Jaccard Distance

The Jaccard Distance is derived from the Jaccard Similarity and measures dissimilarity:

$$D_J(A, B) = 1 - J(A, B)$$

Considering the previous example with sets A and B: $D_J(A, B) = 1 - 0.4 = 0.6$

3 Shingling

When comparing documents for similarity, a naive approach might involve representing each document as a set of its words. However, this method often leads to many documents appearing similar due to common words (e.g., "the", "and") that are prevalent across the language, not necessarily indicative of true content similarity.

3.1 Shingling: A Better Approach

Shingling mitigates this issue by representing documents as sets of contiguous substrings (called shingles) of length k. This method captures more context and reduces the impact of common words. Let's define it properly.

A k-shingle is any substring of length k found within the document. The document is then represented as the set of all unique k-shingles it contains.

For instance, consider the document D represented by the string "abcdabd". The set of 2-shingles (where $k = 2$) is:

$$\{\text{ab}, \text{bc}, \text{cd}, \text{da}, \text{ab}, \text{bd}\}$$

Removing duplicates, the unique 2-shingles are:

$$\{\text{ab}, \text{bc}, \text{cd}, \text{da}, \text{bd}\}$$

3.2 Shingling in Practice

Selecting an appropriate value for k is crucial for effective shingling. In fact, k should be large enough so that the probability of any given shingle appearing in any given document is minimized. This reduces the likelihood of different documents sharing shingles by chance.

- Assume a document uses the 27-character ASCII set, and $k = 5$. The number of possible unique shingles is $27^5 = 14,348,907$. For most documents significantly smaller than this size, such as emails or articles, $k = 5$ ensures that shingles are relatively unique, enhancing the distinction between different documents.
- For large documents, larger values, such as $k = 10$, are more appropriate to capture sufficient context.

However, shingle sets can become large, especially for long documents. To manage this, shingles are often hashed into integers with a limited number of bits. For example, using 10-bit hashing for 2-shingles:

$$\{\text{ab}, \text{bc}, \text{cd}, \text{da}, \text{bd}\} \rightarrow \{342, 825, 312, 54\}$$

This hashing reduces storage requirements while preserving the ability to compute similarity measures like the Jaccard Similarity.

3.3 Representing Sets of Documents as a Matrix

To efficiently compute similarities between multiple documents, we can represent the sets of shingles as a Boolean matrix. The matrix has the following structure:

- **Rows**: Represent the unique shingles across all documents.
- **Columns**: Represent individual documents.
- **Entries**: A binary value indicating the presence (1) or absence (0) of a shingle in a document.

Let's consider another example, with 4 documents represented as sets:

$$D1 = \{a, d\}$$
$$D2 = \{c\}$$
$$D3 = \{b, d, e\}$$
$$D4 = \{a, c, d\}$$

The shingle hash matrix might look like this:

$$\begin{bmatrix} \text{Shingle} & \text{D1} & \text{D2} & \text{D3} & \text{D4} \\ 1 & 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 0 & 1 & 0 & 1 \\ 4 & 0 & 0 & 1 & 0 \\ 5 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Here, each row corresponds to a shingle, and each column corresponds to a document. A '1' indicates that the document contains the shingle, while a '0' indicates absence.

4 Min Hashing

Previously, we discussed the process of shingling as a way to represent documents as sets of substrings (shingles), which allows for measuring similarity using the Jaccard similarity. However, representing documents as sets of shingles can become memory-intensive, especially when dealing with large datasets. Storing and comparing these sets in a Boolean matrix, where each row represents a unique shingle and each column represents a document, can quickly become computationally expensive due to the large number of possible shingles.

To address this issue, we introduce Min-Hashing, a technique designed to efficiently compress the sets (columns in the matrix) into smaller signatures while preserving the Jaccard similarity between the documents. A signature is a compact representation of a set, such as the set of shingles. Min-Hashing allows us to avoid storing or comparing the entire matrix, reducing both space and computational costs.

4.1 Min-Hashing and Jaccard Similarity

The goal of Min-Hashing is to find a hash function h on the shingle sets of the documents such that:

- If two documents D_1 and D_2 have high similarity (high Jaccard similarity), then with high probability, the Min-Hash signatures $h(D_1)$ and $h(D_2)$ will be the same ($h(D_1) = h(D_2)$).
- If two documents D_1 and D_2 have low similarity, the signatures will likely be different ($h(D_1) \neq h(D_2)$).

The key property of Min-Hashing is that the probability of two documents D_1 and D_2 having the same Min-Hash value (after a random permutation) is equal to the Jaccard similarity of the two sets. Specifically:

$$\Pr[h_\pi(D_1) = h_\pi(D_2)] = \mathrm{J}(D_1, D_2)$$

In other words: $\boxed{\text{the similarity of signature sets}} = \boxed{\text{the similarity of shingle sets}} = \boxed{\text{similarity of documents}}$

4.2 How Min-Hashing Works

The main operation to remember is permuting the Rows (Shingles) of the Boolean Matrix. We start with a Boolean matrix, where:

Rows represent unique shingles & **Columns represent documents**

Each entry in the matrix is either 1 (indicating the presence of a shingle in the document) or 0 (indicating its absence). To implement Min-Hashing, we randomly permute the rows (the shingles) of the matrix. For each column (document), the Min-Hash value is defined as the index of the first row (after permutation) that contains a 1. In other words, for each document, the Min-Hash function returns the first shingle (in the permuted order) that appears in the document.

4.2.1 Min-Hashing in Practice

In practice, we need to use multiple hash functions to improve accuracy. For each document, we generate a signature consisting of multiple Min-Hash values, one for each hash function (or random permutation). The similarity between two documents is then estimated as the fraction of hash functions for which the Min-Hash values are the same.

Implementation Details:

- Instead of explicitly permuting the rows (which can be costly), we use a set of well-chosen hash functions that simulate the effect of random permutations.
- For each row (shingle) and each document, compute the Min-Hash value by applying each hash function and taking the minimum value.

- The more hash functions we use, the more accurate the similarity estimate, but the computational cost also increases.

4.3 Min-Hashing Algorithm

Let's consider the previous example and 3 permutations π_1, π_2, π_3 applied to a matrix where documents D_1, D_2, D_3, and D_4 are represented as columns. After applying a permutation π, the first row that contains a 1 for each column gives us the Min-Hash value.

$\overbrace{\begin{bmatrix} \text{Shingle} & \text{D1} & \text{D2} & \text{D3} & \text{D4} \\ 4 & 0 & 0 & \mathbf{1} & 0 \\ 1 & \mathbf{1} & 0 & 0 & \mathbf{1} \\ 5 & 0 & 0 & 1 & 0 \\ 3 & 0 & \mathbf{1} & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 \end{bmatrix}}^{\pi_1}$ $\underbrace{[1,3,4,1]}$

$\overbrace{\begin{bmatrix} \text{Shingle} & \text{D1} & \text{D2} & \text{D3} & \text{D4} \\ 1 & \mathbf{1} & 0 & 0 & \mathbf{1} \\ 5 & 0 & 0 & \mathbf{1} & 0 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 0 & \mathbf{1} & 0 & 1 \\ 4 & 0 & 0 & 1 & 0 \end{bmatrix}}^{\pi_2}$ $\underbrace{[1,3,5,1]}$

$\overbrace{\begin{bmatrix} \text{Shingle} & \text{D1} & \text{D2} & \text{D3} & \text{D4} \\ 5 & 0 & 0 & \mathbf{1} & 0 \\ 2 & \mathbf{1} & 0 & 1 & \mathbf{1} \\ 3 & 0 & \mathbf{1} & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 4 & 0 & 0 & 1 & 0 \end{bmatrix}}^{\pi_3}$ $\underbrace{[2,3,5,2]}$

For the permutation π_1, we have the following results: For D_1 the first 1 appears at 2th row (after permutation) which corresponds to Shingle 1, For D_2 the first 1 appears at 4th row (after permutation) which corresponds to Shingle 3, For D_3 the first 1 appears at 1st row (after permutation) which corresponds to Shingle 4 and finally for D_4 the first 1 appears at 2th row (after permutation) which corresponds to Shingle 1). After the 3 random permutations of the rows, the Min-Hash values (i.e., the first row where 1 appears) for each document are:

- Min-Hash Values for π_1: $[1, 3, 4, 1]$
- Min-Hash Values for π_2: $[1, 3, 5, 1]$
- Min-Hash Values for π_3: $[2, 3, 5, 2]$

Signature Matrix: $\begin{bmatrix} \text{Document} & \text{Min-Hash 1} & \text{Min-Hash 2} & \text{Min-Hash 3} \\ 1 & 1 & 1 & 2 \\ 2 & 3 & 3 & 3 \\ 3 & 4 & 5 & 5 \\ 4 & 1 & 1 & 2 \end{bmatrix}$

With the final signature matrix, we can clearly see which documents are more similar between each other. For instance, D_1 and D_4 share 3 same values (intersection is 3 and the union is 3 as well), therefore, its Jaccard similarity would be 1, meaning that they are very similar.

However, considering D_2 and D_3, as there are no common values, we can conclude that their similarity is very low.

5 Locality Sensitive Hashing

The Locality-Sensitive Hashing (LSH) algorithm builds upon Min-Hashing by introducing a method to reduce the number of comparisons required between documents, particularly when dealing with very large datasets.

5.1 Problem with Min-Hashing

- **Problem**: Min-Hashing reduces the dimensionality of documents by transforming them into signatures. Each signature is a compact representation of the original document, and the similarity between two documents can be approximated by comparing their Min-Hash signatures.
- **Challenge**: Even after reducing the size of the documents to Min-Hash signatures, there could still be a large number of pairs to compare, particularly for massive datasets. We need a method to only compare document pairs that are likely to have a high similarity, avoiding unnecessary comparisons.

5.2 LSH: The Idea

The idea of Locality-Sensitive Hashing is to only compare pairs of documents that are likely to be similar. Instead of comparing every possible pair of Min-Hash signatures, LSH uses a hashing function to divide the signature matrix into bands. Each document's signature is hashed multiple times (once for each band), and if two documents' signatures hash to the same "bucket" at least once, they are considered candidate pairs for further comparison. LSH ensures that documents with high similarity are more likely to end up in the same bucket, while on the other hand, dissimilar documents are less likely to do so.

The following probabilities are used to understand and tune the behavior of LSH:

- **Probability of agreement in one band**: If two documents have similarity s, the probability that they agree in all rows of a band is s^r.
- **Probability of disagreeing in all bands**: The probability that two documents disagree in at least one row of all bands is $(1 - s^r)^b$.
- **Candidate pair probability**: Two documents are considered candidate pairs if they agree in all rows of at least one band. The probability of this happening is $1 - (1 - s^r)^b$.

5.3 The S-Curve in LSH

The S-curve describes the probability of two documents becoming candidate pairs as a function of their similarity. The steepest part of the curve, where the probability is around 0.5, is where the system is most sensitive to changes in similarity.

The threshold t is chosen where the curve is steepest. This value of t approximates the similarity at which the probability of two documents being considered candidate pairs is high. The formula for the threshold is:

$$t = (\frac{1}{b})^{1/r}$$

Choosing b and r: The values of b and r determine the shape of the S-curve. By selecting appropriate values, we can minimize the number of false positives (green area in the S-curve) and false negatives (blue area), ensuring efficient candidate selection.

5.4 General Steps of LSH

- **Step 1**: We have a signature matrix where each column represents the Min-Hash signature of a document.
- **Step 2**: We divide the matrix into b bands with r rows per band.
- **Step 3**: For each band, we hash the rows of each document's signature into a bucket.
- **Step 4**: If two documents fall into the same bucket for at least one band, we consider them candidate pairs for further comparison.
- **Step 5**: We then calculate the Jaccard similarity for only the candidate pairs. This reduces the number of comparisons significantly, while still capturing most similar documents.

5.4.1 Tuning Parameters in LSH

The effectiveness of LSH depends on the values chosen for b (number of bands) and r (rows per band). These parameters control the tradeoff between the number of false positives and false negatives.

The probability of two documents being hashed into the same bucket (i.e., becoming candidate pairs) depends on their similarity and the parameters b and r.