Book Recommendation System Based on Cosine Similarity

Your Name

July 22, 2025

Abstract

Contents

1 Introduction

2 Literature Review

3 Data and Feature Engineering

In the words of Yoshua Bengio: Good input features are essential for successful ML. Feature engineering is close to 90

page 6: Features them-selves are not so clear cut, going from raw data to features involves extracting features following a featurization process (Section 1.5.2) on a data pipeline. This process goes hand in hand with data cleaning and enhancement.

4 Data and Feature Engineering

4.1 Dataset Description

The dataset used for this study was sourced from *[insert source, e.g., Kaggle, Goodreads API]*, and contains metadata for approximately **10,000 books**. Each entry includes attributes such as the *book title*, *author name*, *average rating*, *number of ratings*, *user-generated tags*, and a *textual description*.

These diverse fields provide both structured and unstructured information, allowing the construction of a feature-rich content-based recommendation engine. Our system leverages these features to compare and recommend books based on their similarity.

4.2 Preprocessing and Cleaning

Before constructing features, the dataset underwent several preprocessing steps to ensure consistency and quality:

- Removal of duplicate or incomplete entries (e.g., books missing descriptions or titles).
- Standardization of textual data by converting all characters to lowercase.
- Elimination of punctuation, digits, and non-ASCII characters.
- Removal of common English stop words (e.g., “and”, “the”, “is”).
- Optional lemmatization or stemming to reduce word variants to their root forms.

Numerical fields such as average rating and number of reviews were normalized to ensure comparability across different scales if used in later analysis.

4.3 Feature Selection and Construction

The goal of this step was to define a meaningful and discriminative representation of each book. We selected the following features:

- **Title**: The official book title, which can carry thematic cues.
- **Tags**: User-generated tags summarizing book themes (e.g., "fantasy", "science-fiction").
- **Description**: A summary or synopsis of the book content.

These three fields were concatenated into a single text document per book, forming a unified textual representation. This aggregated text served as the input for vector embedding in the subsequent step.

4.4 Feature Representation Overview

The resulting text document for each book was vectorized using the **TF-IDF (Term Frequency-Inverse Document Frequency)** method. This representation captures the relative importance of terms in a given document compared to the entire corpus, helping highlight distinctive keywords.

The final result is a sparse vector for each book, which can be used to compute cosine similarity — a measure of textual closeness — between any pair of books. This process is detailed in Section **??**.

5 Vector Embedding

6 Similarity Scoring (Cosine Similarity Matrix)

7 Visualization and Heatmap

8 Clustering

8.1 What is Clustering?

Suppose you are working with a dataset that includes patient information from a health-care system. The dataset is complex and includes both categorical and numeric features. You want to find patterns and similarities in the dataset. How might you approach this task?

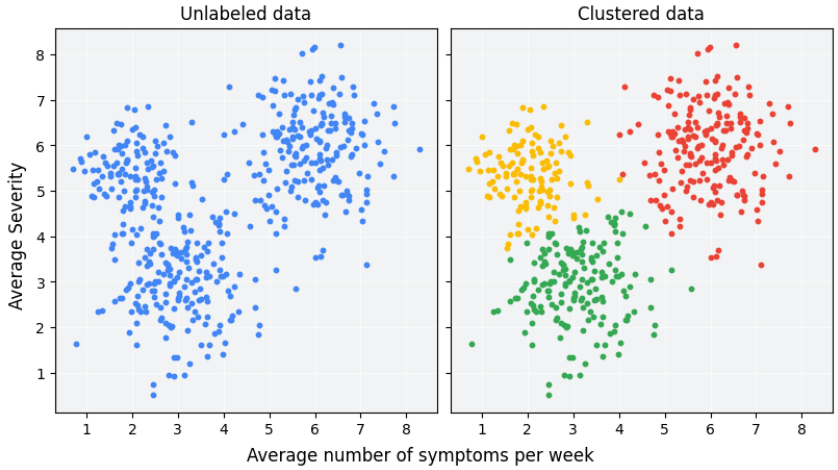

Figure 1

Clustering is an **unsupervised machine learning** technique designed to group unlabeled examples based on their similarity to each other. (If the examples are labeled, this kind of grouping is called *classification*.) Consider a hypothetical patient study designed to evaluate a new treatment protocol. During the study, patients report how many times per week they experience symptoms and the severity of the symptoms. Researchers can use clustering analysis to group patients with similar treatment responses into clusters.The figure demonstrates one possible grouping of simulated data into three clusters.

8.2 Types of clustering

There are many different **clustering** algorithms as there are multiple ways to define a cluster. Different approaches will work well for different types of models depending on the size of the input data, the dimensionality of the data, the rigidity of the categories and the number of clusters within the dataset.

8.2.1 Centroid-based clustering

Centroid-based clustering is a method that divides a data set into similar groups based on the distance between their centroids. The **k-means** clustering algorithm is a commonly used technique, where each data point is assigned to a separate cluster. It assumes that the center of each cluster defines the cluster using a distance measure, typically Euclidean distance. The algorithm initializes the clustering by providing a number of expected clusters, which represents the 'K' in K-means. The optimal k clusters are identified by iteratively minimizing the total distance between each point and its assigned cluster centroid. **K-means** is a hard clustering approach, working well when clusters are of roughly equivalent size and there are no significant outliers or changes

in density across the data. Another approach is K-medoids, which uses individual data points as the medoid or center of the cluster, making it less sensitive to outliers.

8.2.2 Hierarchical-based clustering

Hierarchical clustering, also known as **connectivity-based clustering**, groups data points based on their proximity and connectivity across all dimensions. It creates a **hierarchical graph network** of clusters at each hierarchical level, with each node having one parent node but multiple child nodes. These clusters can be visualized using a **dendrogram** to organize and summarize discovered clusters.

There are two approaches to performing hierarchical cluster analysis: **agglomerative** and **divisive**. **Agglomerative clustering** starts with individual data points and merges clusters by computing the **proximity matrix** of all clusters at the current level of the hierarchy. The algorithm moves to the set of newly created clusters and repeats the process until there is one **root node** at the top of the hierarchical graph.

Divisive hierarchical clustering methods partition data points into a **tree-like structure** using a **top-down approach**. The first step is to split the dataset into clusters using a **flat-clustering method** like **K-Means**. The clusters with the largest **Sum of Squared Errors (SSE)** are then partitioned further using a flat clustering method. The algorithm stops when it reaches individual nodes or some **minimum SSE**.

Divisive hierarchical clustering allows greater flexibility in terms of the **hierarchical structure** and the level of balance in different clusters. It can be faster than **agglomerative hierarchical clustering**, especially when the data doesn't require constructing the tree all the way down to individual data points.

8.2.3 Distribution-based Clustering

Distribution-based clustering, also known as **probabilistic clustering**, is a **model-based approach** that groups data points based on their **probability distribution**. This approach assumes a process generating **normal distributions** for each dimension of the data, creating **cluster centers**. It differs from **centroid-based clustering** in that it doesn't use a **distance metric** like **Euclidean** or **Manhattan distance**. Instead, it looks for a well-defined **distribution** that appears across each dimension. The **cluster means** are the means of the **Gaussian distribution** across each dimension.

One commonly used approach is the **Gaussian Mixture Model (GMM)** through **Expectation-Maximization**. This model assumes that each cluster is defined by a **Gaussian Distribution**, often called a **normal distribution**. For example, a dataset with two distinct clusters, A and B, defined by two different normal distributions, is considered. The GMM uses **Expectation-Maximization**, which starts with a random guess for the distributions along each axis and then improves iteratively by alternating

two steps:
Expectation: assigning each data point to each cluster and computing the probability that it came from **Cluster A** and **Cluster B**;
Maximization: updating the parameters that define each cluster, a **weighted mean location**, and a **variance-covariance matrix** based on the likelihood of each data point being in the cluster.

Even a given point may be **probabilistically associated** with multiple clusters, making it suitable for scenarios like **diverse language preferences**.

8.2.4 Density-based clustering

DBSCAN is a **clustering algorithm** that uses **density-based clustering** to detect clusters of any shape, size, or density in data. This approach is particularly useful when working with datasets with **noise** or **outliers** or when there is no prior knowledge about the **number of clusters** in the data. DBSCAN uses a **density-based spatial clustering** approach to create clusters with a **density** passed in by the user, centered around a **spatial centroid**. The area immediately around the centroid is referred to as a **neighborhood**, and DBSCAN attempts to define neighborhoods of clusters with the specified density. For each cluster, DBSCAN defines three types of data points: **core points**, **border points**, and **outliers**.

A variant of DBSCAN, **HDBSCAN**, does not require any **parameters** to be set, making it even more flexible. HDBSCAN is less sensitive to **noise** and **outliers** in the data and can handle clusters of **varying density** more effectively. This is a primary motivation for HDBSCAN, which handles clusters of varying density more effectively than DBSCAN.

8.3 DBSCAN

As mentioned earlier, DBSCAN is a density based clustering algorithm that divides your entire dataset into dense regions separated by sparse regions.

8.3.1 Advantages of DBSCAN

Robust to outliers : It is robust to outliers as it defines clusters based on dense regions of data, and isolated points are treated as noise.

No need to specify clusters : Unlike some clustering algorithms, DBSCAN does not require the user to specify the number of clusters beforehand, making it more flexible and applicable to a variety of datasets.

No need to specify clusters : DBSCAN can identify clusters with complex shapes and is not constrained by assumptions of cluster shapes, making it suitable for data with irregular structures.

Only 2 hyperparameters to tune : DBSCAN has only two primary hyperparameters to tune: "eps" (distance threshold for defining neighborhood) and "min-samples" (minimum number of points required to form a dense region). This simplicity can make parameter tuning more straightforward.

8.3.2 Disadvantages of DBSCAN

Sensitivity to hyperparameters : The performance of DBSCAN can be sensitive to the choice of its hyperparameters, especially the distance threshold (eps) and the minimum number of points (min-samples). Suboptimal parameter selection may lead to under-segmentation or over-segmentation.

Difficulty with varying density clusters : DBSCAN struggles with clusters of varying densities. It may fail to connect regions with lower point density to the rest of the cluster, leading to suboptimal cluster assignments in datasets with regions of varying densities.

Does not predict : Unlike some clustering algorithms, DBSCAN does not predict the cluster membership of new, unseen data points. Once the model is trained, it is applied to the existing dataset without the ability to generalize to new observations outside the training set.

8.3.3 Comparison with K-means

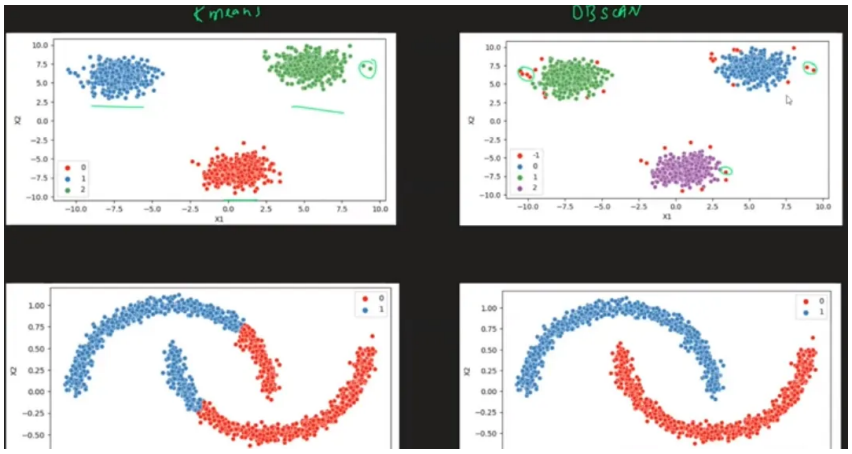

Figure 2: Comparison between K-means clustering and other clustering methods.

9 Discussion

10 Conclusion and Future Work

References