Topic Modeling and Cosine Similarity-Based Content Recommendation Engine for the Criterion Channel

Aaron Tan, Prashant Mourya

Abstract

A recommendation engine filters data using different algorithms and recommends the most relevant items to users. It first captures the past behavior of a customer and, based on that, recommends products which the customer might be likely to consume. If a completely new user visits an e-commerce site, that site will not have any past history of that user. So how does the site go about recommending products to the user in such a scenario? One possible solution could be to recommend the best-selling products, i.e., the products which are highest in demand. Another possible solution could be to recommend the products which are the most similar to a target product that the user inputs. We developed two types of content-based recommendation engines: The first uses Latent Dirichlet Allocation (LDA) topic modeling computed over film descriptions; this method takes a target film and then outputs a list of similar films within the same topic ranked by IMDB rating. The second uses cosine similarity computed over TF-IDF matrices of film descriptions; this method takes a target film and then outputs a list of the closest films based on cosine similarity. These two recommendation engines should give viewers two options to play with in order to find the next film that they would likely enjoy.

Motivation

For our project, we used film metadata from the Criterion Channel (<https://www.criterionchannel.com>). The Criterion Channel describes itself as “A Movie Lover’s Dream – Classics and discoveries from around the world, thematically programmed with special features, on a streaming service brought to you by the Criterion Collection.” It is a streaming service for films in the Criterion Collection, but unlike services such as Netflix and Hulu, it does not rely on a recommendation engine to suggest films for viewers to watch. Instead, it relies on a weekly hand-curated newsletter delivered by email that alerts viewers of new or relevant content. For this reason, we decided to build a recommendation engine for the Criterion Channel – one that would allow a viewer to input a film they recently watched and enjoyed, and then receive a list of similar films.

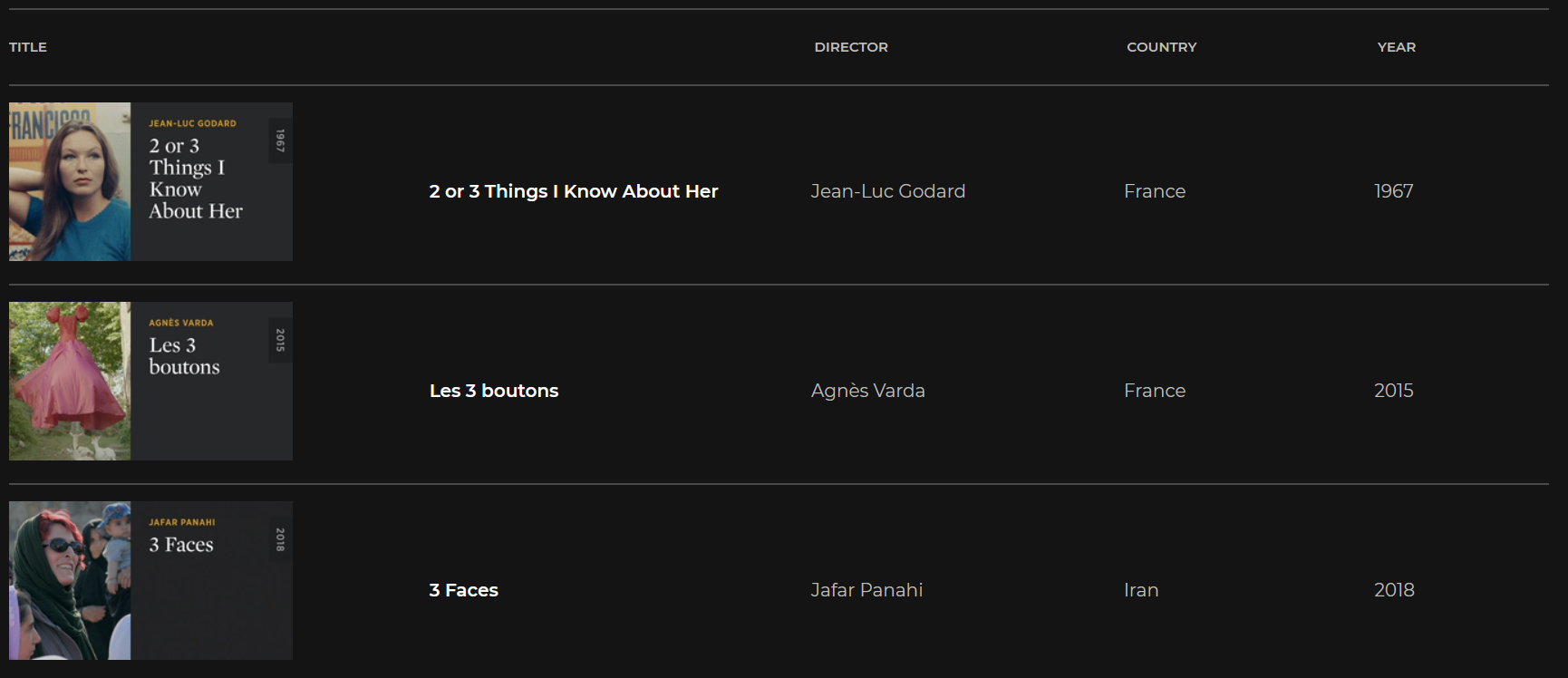


Figure 1. The Criterion Channel, a very basic, non-algorithmic film streaming service.

Data Collection

We used Beautiful Soup to scrape the Criterion Channel’s film metadata. This resulted in a dataset of 1,621 films with the columns: Title, Director, Country, Year, Decade, Duration, Description, Total Hours, and Url.

We merged the Criterion Channel dataset with up-to-date IMDB ratings from <https://www.imdb.com/interfaces>. Merging faced some issues, since film titles can be represented slightly differently between the two datasets (e.g., capitalization, accents, etc.). Due to the size of the datasets (~1,000 rows for the Criterion Channel dataset and ~1,000,000 rows for the IMDB dataset), computing pairwise distance metrics such as Levenshtein Distance via FuzzyWuzzy was computationally intractable. To solve this problem, we used String Grouper (<https://github.com/Bergvca/string_grouper>), “a library that makes finding groups of similar strings within a single, or multiple, lists of strings easy — and fast. string\_grouper uses tf-idf to calculate cosine similarities within a single list or between two lists of strings. The full process is described in the blog Super Fast String Matching in Python.” Comparing a title string from one dataset using cosine similarity to an TD-IDF matrix that fully captures a second dataset was immensely faster. After merging, there was some data loss, as some films were in one dataset but not the other, and we were left with 1,436 films in the merged dataset. This dataset contains all the metadata from the Criterion Channel in addition to data on average IMDB user rating and number of ratings.

Why Content-Based Recommendation

There are two main approaches to recommendation engines, content-based and collaborative. Content-based recommendation engines recommend items based on some relation between a target item/items and other items in the dataset (e.g., if you bought products A and B, you are recommended similar products C and D). Collaborative recommendation engines recommend items based on the similar behavior of users (e.g., if you liked products A and B, and another user also liked those products plus product C, you are recommended product C).

Content-based recommendation addresses the ‘cold start’ problem associated with collaborative filtering, where certain items do not have any rating information and hence the corresponding item vectors consist of all zeroes (we use zeroes to represent missing ratings in the rating matrix). Given that the Criterion Channel does not have any user rating data, and given that our results would be arbitrary and uninterpretable if we generated our own rating data, we chose this approach.

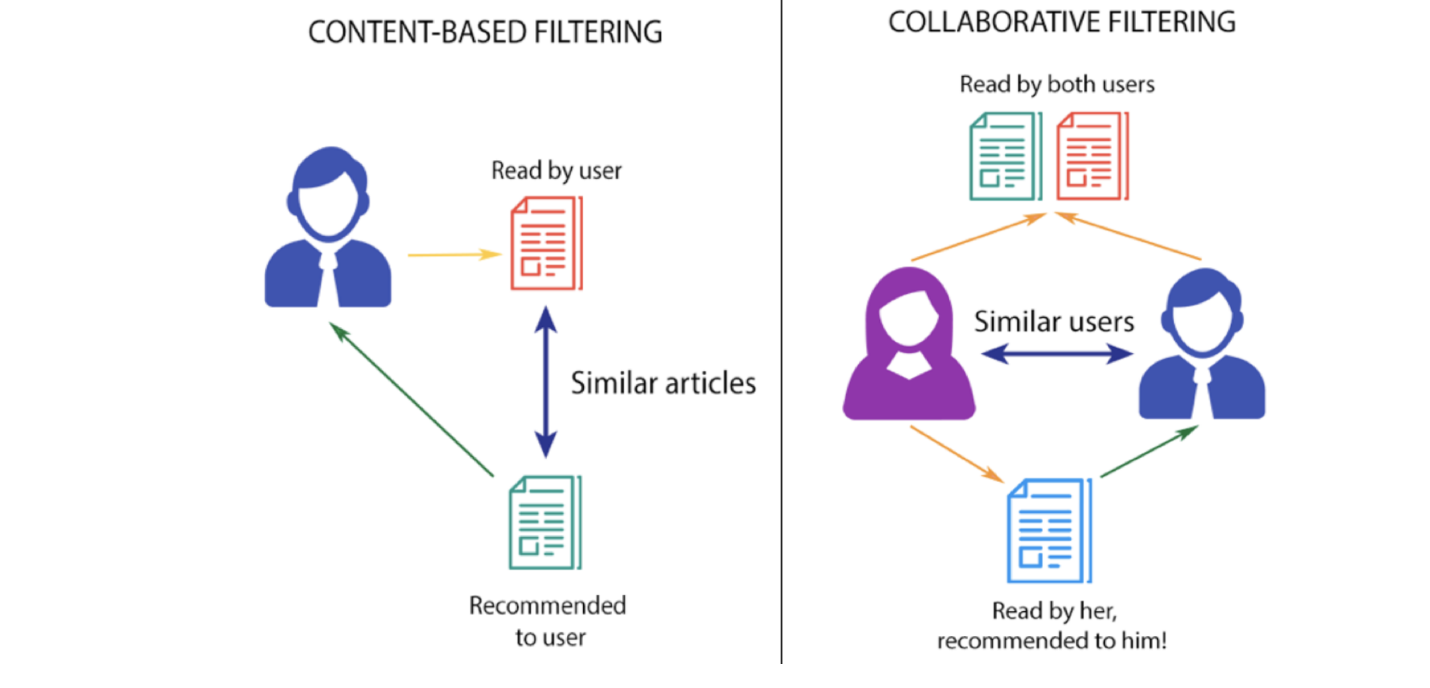


Figure 2. Content-based vs. collaborative filtering.

Text Preprocessing

For LDA topic modeling, the film descriptions need to be preprocessed. Natural Language Toolkit (NLTK) was used to clean, tokenize, and lemmatize the descriptions. In NLP, a corpus is a collection of text data, used for verifying hypothesis about language such as extracting features from text or finding patterns of word usage. For the movie review data, we collected the text data and performed the following preprocessing stages. During preprocessing, irrelevant words such as {of, and, or} are removed using a common English stopword list. Next, NLTK’s default lemmatizer is used for lemmatization. It uses WordNet Database to look up lemmas. A lemmatizer reduces all derivationally related forms of a word to a common base form. For example, the word “cars” is reduced to “car”. This allows us to keep the base word and remove other forms of same word in a corpus. Ultimately, the purpose of preprocessing is to remove noise from the data so that algorithms such as LDA or cosine similarity have more relevant and salient information to work with.

Topic Modeling

One approach for a content-based recommendation engine is to use topic modeling of plot summaries to identify latent themes/topics. We can learn topic representations for each item (a vector of topic proportions) and add them to the item vectors in the latent-factor model. Such topic representations of movie items are also useful outside the domain of movie recommendation. Interpretability of topics may help in explaining recommendations to users, effective content programming, and ad targeting based on user profiles. In summary, what LDA accomplishes is to create a generative statistical model that assumes that each document is a mixture of “topics,” each topic being a list of tokens. Each document – in this case, each film description – then has a “dominant topic,” which is the topic that captures the greatest amount of variance in the document within the mixture of topics. For example, one topic contains the tokens “filmmaker, life, journey, acclaimed,” and a film for which this is the dominant topic is Federico Fellini’s 8½, a film about the life and journey of a fictional filmmaker.

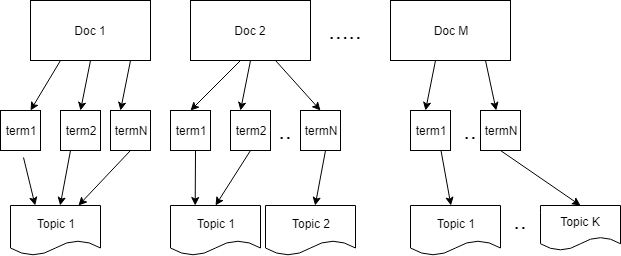


Figure 3: LDA topic modeling. Documents are represented as a mixture of topics that are mixtures of tokens.

Gensim was used to create a unigram + bigram model. In paragraphs, certain words always tend to occur in pairs (bigram) or in groups of threes (trigram) because the two or three words combine together form the actual entity. For example: The word ‘French’ refers the language or region and the word ‘revolution’ can refer to planetary revolution. But by combining them, ‘French Revolution,’ refers to something completely different.

This model was then transformed into a bag of words corpus, an object that contains the word ID and its frequency in each document. You can think of it as Gensim’s equivalent of a document-term matrix. To create Gensim’s bag of words corpus, all you need to do is to pass the tokenized list of words to the dictionary.doc2bow().

For LDA topic modeling, we used the Mallet version of Gensim, described as a topic modeling package that “includes an extremely fast and highly scalable implementation of Gibbs sampling, efficient methods for document-topic hyperparameter optimization, and tools for inferring topics for new documents given trained models” (<http://mallet.cs.umass.edu/topics.php>).

We created 100 topic models ranging from 1 to 100 topics. We also computed and plotted coherence scores for each number of topics. For this dataset, coherence continues to climb up to 100 topics, probably due to the very heterogeneous nature of film descriptions. We are currently using the 100-topic model, but will consider using fewer topics, because as the number of topics increase, the number of films within each topic decrease, potentially reducing the power of the recommendation engine. After topic modeling, our dataset includes labeled topics and topic descriptions for each of the films in the Criterion Channel.

Cosine Similarity

Another approach for a content-based recommendation engine is to use cosine similarity to identify films with similar descriptions to a target film. For cosine similarity, TF-IDF matrices were constructed scikit-learn’s TfidfVectorizer. Cosine similarity is a metric used to measure how similar two items are. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. The output value ranges from 0–1. 0 means no similarity, whereas 1 means that both items are 100% similar. sklearn’s linear\_kernel computes similarity as the normalized dot product of input samples X and Y (TD-IDF matrices) – this is the cos θ for the two vectors in the count matrix, in other words, the cosine similarity.

Results

The recommendation engine itself was implemented using ipywidgets and Voila. Functions were written to implement the two engines as described above:

“The first uses Latent Dirichlet Allocation (LDA) topic modeling computed over film descriptions; this method takes a target film and then outputs a list of similar films within the same topic ranked by IMDB rating. The second uses cosine similarity computed over TF-IDF matrices of film descriptions; this method takes a target film and then outputs a list of the closest films based on cosine similarity. These two recommendation engines give viewers two options to play with in order to find the next film that they would likely enjoy.”

For example, taking a target film – say, Akira Kurosawa’s Seven Samurai – we can input this film into the topic modeling recommendation engine. The dominant topic of this film contains the tokens, “samurai, godzilla, monster, series, make, earth, action, clan.” Within this topic, the film with the highest IMDB rating is Harakiri, another iconic samurai film, and a very suitable suggestion for a film to watch next. If the target film Seven Samurai is input into the cosine similarity recommendation engine, the engine outputs the film with the greatest cosine similarity to the target film – in this case, the film Drunken Angel, another thrilling Kurosawa film with the same star as the Seven Samurai. Using these two recommendation engines, you can browse through the first, second, third, etc. recommendation, and very likely find a film to watch next that fits your current mood and viewing trajectory.

References

Ghosh, S., & Gunning, D. (2019). Natural Language Processing Fundamentals.

Lane, H., Howard, C., & Hapke, H. M. (2019). Natural Language Processing in Action: Understanding, analyzing, and generating text with Python.

Sarkar, D. (2019). Text Analytics with Python: A Practitioner’s Guide to Natural Language Processing.

<https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05>

<https://ai.intelligentonlinetools.com/ml/category/topic-modeling/>