Movie Recommender

Ansh Bordia (ansh979@gmail.com)

1 Introduction

A movie recommender system is a system you come across almost every day. Be it on Netflix, Prime Video or even YouTube. The 'Trending' page on YouTube is a recommender and so is 'Because you watched (#insert movie name)' on Netflix.

A movie recommender system comes in three forms of implementations:

1) Recommendation Based – A simple system based on the most popular videos in a user's region. Yes, you guessed it – The YouTube Trending Page
2) Content Based – Finds similar movies based on different metrics like genre, language, movie summary, actors, director etc.
3) Collaborative Filtering – Based on the watching pattern of user's like you (Yes Google is spying on you!). Netflix uses a combination of both Content based and Collaborative Filtering.

The movie recommender we develop is a simple, yet effective content based one. It uses the concept to Bag of Words + Cosine Similarity (both explained later!).

For those expert recommender system builders wondering why I haven't used the obviously superior Word2Vec/GloVe + Cosine Similarity method, the answer is – my old laptop cannot handle the pre-trained models (The uncompressed Google Word2Vec pretrained model is 3+ GBs!) and similarly training on the entire dataset of a new model is not feasible as well (at least till I get a new one!). Training on a short corpus of, say 1000, movies is possible, but Word2Vec and GloVe perform horribly on such less data – the Google model has word vectors for 3 million words!

Anyway, enough digressing. Let's dive straight into the implementation.

2 Dataset

https://www.kaggle.com/rounakbanik/the-movies-dataset

3 Implementation

The recommender will use the metrics: Title, Overview and Genre. We follow the following pipeline approach to solve the problem at hand:

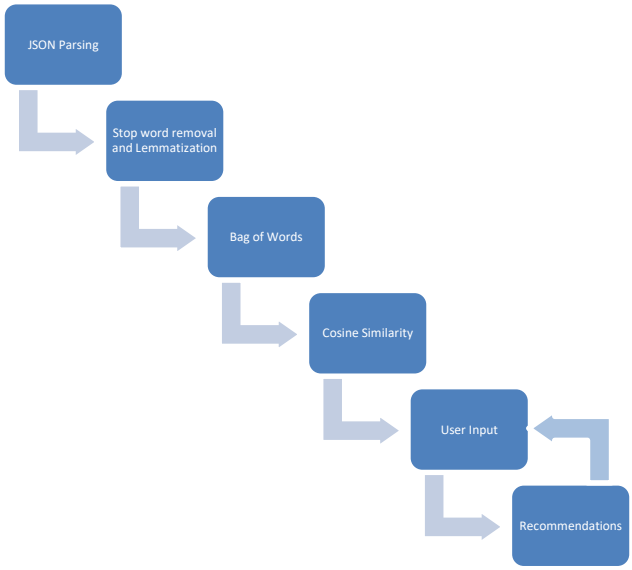

Figure 1

3.1 JSON Parsing

The Genre field has multiple genres of a movie listed. The movie 'Jumanji' has the genres – adventure, fantasy and family. The format at first seems JSON, but it isn't. We first replace all the single quotes with double quotes and then convert it into JSON format. We, subsequently, use that dictionary (JSON object) to extract genres for each movie in a list format, and add a new column to our data with the list representation.

3.2 Stop Word Removal and Lemmatization

Words like 'and', 'moreover' are called stopwords because they do not add meaning to the 'overview' of a movie. We remove them using NLTK stopwords.

Lemmatization of words is the converting a derived form into a root form. In the context of our system the words 'running', 'ran', 'runs' do not give any extra information than the root form 'run'. So, we lemmatize these words.

An advantage of lemmatization over stemming is that we always get words that are English words. Doing this also helps improve speed of the algorithm as the Bag of Words model has to deal with a significant a smaller set of words. Note, lemmatization is not always the solution in all problems. In problems like Authorship Attribution, it may be beneficial to not do lemmatization as certain words help capture author traits.

3.3 Bag Words

Bag of Word is a form of Word Embedding. Word Embedding is representation of text vocabulary and helps capture syntactic and semantic similarity. Enough jargon, basically if your vocabulary has 5 words – {Notebook, Pencil, Paper, Pen, Eraser} and the text you entered was 'Eraser Pencil' then it would look somewhat like:

Word	Count
Notebook	0
Pencil	1
Paper	0
Pen	0
Eraser	1

Table 1

In actual practice there will be tens of thousands of unique words and you will get an extremely sparse matrix. Handling sparse matrices can be computationally expensive. I guess this would be a good time to thank lemmatization for making the sparse matrices less sparse!

3.4 Cosine Similarity

In simple words, cosine similarity is a measure of how similar two pieces of text are. It forms vector representations of the given texts and finds the cosine of the angel between them – 0 for no similarity and 1 for having the same contents. The cosine similarity is advantageous because it considers direction and not magnitude of vectors. So, even if two similar texts are far apart by the Euclidean distance, they may still be oriented close together.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Formula 1

4. Results

We output the top 3 recommendations (first 5000 movies) for a movie:

Movie	Recommendations	Notes
Sudden Death	Never Been Kissed Air Force One War of the Worlds	The last two recommendations fit perfectly well if you check the movies out of Google. While the first movie is clearly misplaced.
Bad Boys	Eddie Hard Target Confessional	The second one is a clear match while the others, not exactly the same, have some similar contexts.

I made an interesting observation which countered my used of lemmatization. Upon removing the lemmatization step, I observed several improvements for multiple movies.

For example, Bad Boys gave the output - Three Fugitives, Tetsuo II: Body Hammer, Real Life which were much better results compared to before. The same was observed for other movies as well. This could perhaps be attributed to derived forms contributing more meaning than expected.

4 Improvements

The Bag of Words model is disadvantageous because it fails to process the meaning of words. Words 'minister' and 'politician' should be termed very similar, but BOW model fails to do so because of no common words. Over here Word2Vec or GloVe embeddings would be very advantageous as they overcome the mentioned weakness. Applying Smooth Inverse Frequency to Word Embedding Method helps us overlook irrelevant words (stop words removal is a very crude method).

Other metrics from other files like Director and Cast could also be used to improve the system for users who choose their movies based on the crew.

5. Conclusion

The BOW + cosine distance is a simple yet effective model for a starter recommender system. Using GloVe or Word2Vec is not a very difficult task and

I will be implementing it once I get a better laptop. The code is available on my Github: github.com/anshbordia

6. References

1) Choi, S.-M., Han, Y.-S.: A content recommendation system based on category correlations. In: The Fifth International Multi-Conference on Computing in the Global Information Technology, pp. 1257–1260 (2010)

2) Ko SK. et al. (2011) A Smart Movie Recommendation System. In: Smith M.J., Salvendy G. (eds) Human Interface and the Management of Information. Interacting with Information. Human Interface 2011. Lecture Notes in Computer Science, vol 6771. Springer, Berlin, Heidelberg

3) Andrew Seig (2018). Text Similarities: Estimating the degree of similarity between two texts.