Movie Recommendation System

Rehas Sachdeva, Tanmay Chaudhari, Ramkrishna Maheta, Ayush Jain
Project Mentor: Vishal Gupta

I. PROBLEM STATEMENT

A recommendation system is a type of information filtering system which attempts to predict the preferences of a user, and make suggestions based on these preferences. Websites such as IMDB, Amazon, Wikipedia have a number of reviews about various movies. Given general details of a movie such as genre, ratings, plot summary, director, stars, era etc along with user reviews on IMDB, Amazon and critique on Wikipedia, the goal is to design a movie recommendation system, in which a user describes the kind of movie he/she is interested in watching. The users query can be arbitrarily long and textually rich. The recommendation system should return top k results for the query.

Example Queries with recommendations

- Movies that start with recruitment scenes? Think along the lines of Seven Samurai or Ocean's Eleven. Movies that start with a group slowly being put together based on each member's unique skills and abilities.

  Recommendations: Ronin, The Sting, 13 Assassins, Armageddon, The Dirty Dozen, The Italian Job, The Magnificent Seven.

- Are there any big budget movies about the Bolshevik Revolution? I don't know much about the implementation of communism in russia as a moment in history and think it would make a fascinating topic for a movie, just like how the pianist or schindler's list were a window into the holocaust. Think about it, there was this whole new ideology with profound societal implications that only existed in theory and some guys just got together and forced it to work through a revolution and pretty much mob rule. There is so much content in this area of history! Does anyone have any recommendations for me?

  Recommendations: S. Eisenstein, Reds, Anastasia, Doctor Zhivago, Battleship Potemkin, Nicholas & Alexandra, The Last Station.

II. Previous Approaches

Two main approaches are widely used for recommender systems. One is content-based filtering, where we try to profile the users interests using information collected, and recommend items based on that profile. The other is collaborative filtering, where we try to group similar users together and use information about the group to make recommendations to the user. A brief about two approaches based on collaborative filtering can be found below -

1) *Nearest Neighbors Collaborative Filtering*: This approach relies on the idea that users who have similar rating behaviors so far, share the same tastes and will likely exhibit similar rating behaviors going forward. The algorithm first computes the similarity between users by using the row vector in the ratings matrix corresponding to a user as a representation for that user. The similarity is computed by using either cosine similarity or Pearson Correlation. In order to predict the rating for a particular user for a given movie j, we find the top k similar users to this particular user and then take a weighted average of the ratings of the k similar users with the weights being the similarity values.

2) *Latent Factor Methods*: The latent factor algorithm looks to decompose the ratings matrix R into two tall and thin matrices Q and P, with matrix Q having dimensions numusers k and P having the dimensions num$_i$items k where k is the number of latent factors. The decomposition of R into Q and P is such that

$$R = Q.P^T$$

Any rating r_{ij} in the ratings matrix can be computed by taking the dot product of row q_i of matrix Q and p_j of matrix P. The matrices Q and P are initialized randomly or by performing SVD on the ratings matrix .

III. Motivation

Movie recommendation is a popular problem. There are a lot of implementations available which are based on characteristics of movies and/or use collaborative filtering method to recommend movies to a user based on the movies watched or liked by other similar users.

A large amount of information exists in reviews written by users. This source of information has been ignored by most of the current recommender systems while it can potentially alleviate the sparsity problem and improve the quality of recommendations. Our problem is a novel one. We go a step further to analyze the user reviews for a movie. Moreover, the user while specifying a query, can be as specific as they want to, our results would cater to that requirement.

Currently we have various Question Answering threads such as on Reddit and StackExchange etc. These include relevant movies as answers or comments by humans. Hence this the action that we want to automate. The idea behind using reviews is that reviews depict what a user feels about a movie. When asked a query like, this is the kind of feeling Im looking for in a movie, the suggestions will be based on the reviews.

IV. Dataset Creation

A. IMDB

We crawled data for 3500 movies from IMDB. Movies on IMDB are categorised into 23 genres namely Action, Adventure, Animation, Biography, Comedy, Crime, Documentary, Drama, Family, Fantasy, Film-Noir, History, Horror, Music, Musical, Mystery, Romance, Sci-Fi, Short, Sport, Thriller, War and Western. We crawled around top 150 movies for each genre, sorted by number of votes. Number of votes was chosen as sorting order because usually more votes means more reviews. This also eliminated movies which havent been released yet and hence carry no reviews. Hence for each movie, we have the following fields:

- Genre
- Plot
- Directors
- Writers
- Stars
- Rating
- Similar movies
- Plot keywords

For each movie, top 50 reviews sorted by most helpful, were collected. Only about 10% of the total movies had less than 50 reviews, but still had at least 20 reviews. Each review is written as a title, text pair.

Unique id to title mapping	id-to-title.json
Generic fields or details for each movie id	id-to-details.json
50 reviews data for each movie id	id-to-reviews.json

Table I: IMDB data files

B. Amazon product data for Movies and TV

This is a publically available dataset. From this we extract product id and reviews for each product. We map product id to Unique movie ids collected from IMDB. The reviews are a tuple of following fields:

- Review Text
- Summary
- Overall Rating.

C. Wikipedia Dataset

wiki_data.json We crawled data from the wikipedia page of each of the 3500 movies in our database from IMDB. For each movie, we organised the data into the following fields:

- Cinematography
- Country
- Critical Response
- Directed by
- Distributed by
- Language
- Produced by
- Release date
- Running time
- Plot

D. Reddit Data

Reddit has a lot of threads where user describe the kind of movies they are interseted in watching, followed by a lot of comments as answers. The crawled reddit data is organised as following fields:

- Title of post.
- Link to the post.
- Description of query.
- List of comments.

E. StackExchange Data

StackExchange Data was also obtained in the same way as reddit data. Only posts with positive votes were considered while crawling.

- Title of post.
- Link to the post.
- Votes on comment followed by comment.

F. QRel data creation

To build a set of *QUERY RELEVANCE SET*(qrels) for query q we generate a small number m of query aspects. Using a single IR system, we run each query aspect against the documents.
The union of the top k documents retrieved for each aspect constitutes a list of pseudo-qrels for q. From this point, evaluation proceeds in the usual fashion, merely substituting pseudo-qrels for human-judged qrels. The core idea is that the set of query aspects articulates different facets of the underlying information need. Running each aspect as a query against D allows us to cast a wide net, collecting what is hopefully a variety of relevant documents. We assigned relevance judgement in 3 broader categories.

- A 0 relevance indicates that for a given movie, the recommendation is totally irrelevant for all category, details etc.
- A relevance of 1 indicates that the movie is partially relevant to the recommendation. Example either the details or the genre is similar.
- A relevance of 2 indicates that the movie is totally relevant to the answers of recommendations.

Total movies in Imdb dataset	3508
Total movies with details in Imdb dataset	3508
Total reviews in Imdb dataset	162325
Total movies with details in Wikipedia dataset	3505
Total query asked in Reddit data	1828
Total query answered in Reddit data	1793
Total suggestions in Reddit data	17795
Total query-suggestion in movie.stackexchange.com dataset	3543

Table II: Dataset insight

V. Implementation of baseline approach

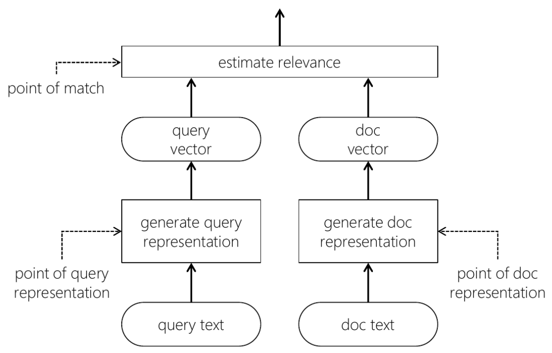
Our baseline approach is based on creating an index for the movies based on the content in their reviews. We have used Apache Solr for indexing and retrieving the Top 40 results. All the reviews from IMDB, Amazon and WIkipedia were aggregated into a document for each movie. Solr interface can then take a user query and match it to the indexed documents, returning a list of top 40 movies for the query.

VI. Deep learning model

We implement a deep learning based model to learn movie properties from the review text. Experimental results demonstrate that DL approach significantly outperforms all baseline approaches.

We reduce our problem to neural document ranking, considering all the content of a movie as a document. The goal then is to retrieve the most relevant documents for the given query.

At the most abstract level, document ranking is described as,



estimate relevance
point of match
query vector
doc vector
generate query representation
generate doc representation
point of query representation
point of doc representation
query text
doc text

Figure 1: Document ranking typically involves a query and a document representation steps, followed by a matching stage. Neural models can be useful either for generating good representations or in estimating relevance, or both.

Neural document ranking differs in whether neural model is applied at representation stage or matching stage or both.

- The first step is to learn vector representations of every term in our vocabulary. When vector representations are dense, small and learnt from data, they are called embeddings.
- Generating the embeddings is done using below methods,
  - LSA - *Latent Semantic Analysis (LSA)* involves performing *singular value decomposition* (SVD) on a term-document (or term-passage) matrix X to obtain its low-rank approximation. SVD on X involves finding a solution to $X = U\Sigma V^T$, where U and V are orthogonal matrices and Σ is a diagonal matrix. While LSA operate on a term-document matrix, matrix factorization based approaches can also be applied to term-term matrices
  - Word2vec - For word2vec the features for a term are made up of its neighbours within a fixed size window over the text from the training corpus.
    * The *skip-gram* architecture is a simple one hidden layer neural network. Both the input and the output of the model is in the form of one-hot vectors and the loss function is as follows.

$$\mathcal{L}_{skip-gram} = \frac{1}{|\mathcal{S}|}\sum_{i=1}^{|\mathcal{S}|}\sum_{c\leq j\leq +c, j\neq 0}\log(p(t_{i+j}|t_i))$$

    where,

$$p(t_{i+j}|t_i) = \frac{\exp(W_{out}\vec{\boldsymbol{v}}_{t_{i+j}})^T(W_{in}\vec{\boldsymbol{v}}_{t_i})}{\sum_{k=1}^{|\mathcal{T}|}\exp(W_{out}\vec{\boldsymbol{v}}_{t_k})^T(W_{in}\vec{\boldsymbol{v}}_{t_k})}$$

    $\mathcal{S}$ is the set of all windows over the training text and c is the number of neighbours we need to predict on either side of the term t_i. The denominator for the softmax function for computing $p(t_{i+j}|t_i)$ sums over all the words in the vocabulary.
    * The *continuous bag-of-words (CBOW) architecture* is similar to the skip-gram model, except that the task is to predict the middle term given the sum of the one-hot vectors of the neighbouring terms in the window. Given a middle term t_i and the set of its neigbours { t_{i-c}, . . . , t_{i-1}, $t_{i+1}, ..., t_{i+c}$ }, , the CBOW model creates a single training sample with the sum of the one-hot vectors of all the neighbouring terms as input and the one-hot vector $\vec{\boldsymbol{v}}_{t_i}$, corresponding to the middle term, as the expected output.

$$\mathcal{L}_{CBOW} = -\frac{1}{|\mathcal{S}|}\log(p(t_i|\sum_{c\leq j\leq +c, j\neq 0} t_{i+j}))$$

  - GloVe - GloVe is trained using AdaGrad. Similar to word2vec, GloVe also generates two different (IN and OUT) embeddings, but unlike word2vec it generally uses the sum of the IN and the OUT vectors as the embedding for each term in the vocabulary.

$$\mathcal{L}_{GLOVE} = \sum_{i=1}^{|\mathcal{T}|}\sum_{j=1}^{|\mathcal{T}|} f(x_{ij})(\log(x_{ij} - \vec{\boldsymbol{v}}_{w_i}^T)\vec{\boldsymbol{v}}_{w_j})^2$$

- For every text in our queries + documents corpus, we aggregate its term embeddings using the below methods. Lets call the final representation from below as R.
  - Linear combination: Average term embeddings: Take coordinate-wise mean of all the embeddings.
  - Max method: Take coordinate-wise max of all the embeddings.
  - Min method: Take coordinate-wise min of all the embeddings.
  - Non-Linear combinations: Fisher Kernel framework: First compute the fisher kernel for each pair of embeddings in the text. Convert the sequence of embeddings in the text to FK values for every consecutive pair.

- Now given a query, compute its $R(q)$. For every document in the corpus compute $R(d)$. Rank the documents using cosine similarity.

A. DSSM: Compute Similarity in Semantic Space

It consists of 5 layers.

- Input as a sequence of terms of texts X and Y.
- Word hashing: use sub-word unit (e.g.,letter n-gram) as raw input to handle very large vocabulary.
- Convolutional and Max-pooling layer: identify key words/concepts in X and Y.
- Representation: use DNN to extract abstract semantic representations.
- Learning: maximize the similarity between X (source) and Y (target).

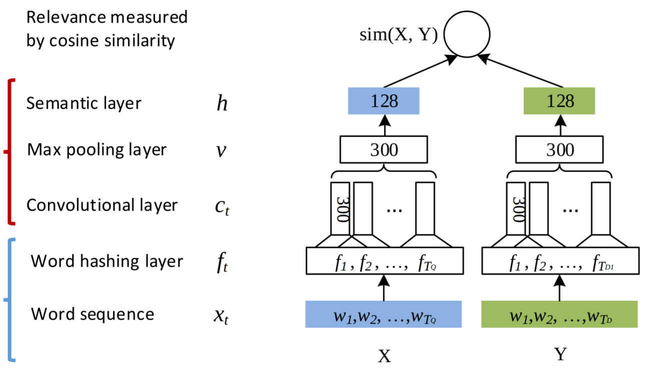

Relevance measured by cosine similarity
sim(X, Y)
Semantic layer h
Max pooling layer v
Convolutional layer c_t
Word hashing layer f_t
Word sequence x_t
128
128
300
300
300
300
$f_1, f_2, \dots, f_{T_Q}$
$f_1, f_2, \dots, f_{T_{D1}}$
$w_1, w_2, \dots, w_{T_Q}$
$w_1, w_2, \dots, w_{T_D}$
X
Y

Figure 2: DSSM Architecture

1) Word hashing layer: We use Letter-trigram Representation to control the dimensionality of the input space as only about 50K letter-trigrams are there in English. Example: e.g., cat is represented as # cat # which is represented as # -c-a, c-a-t, a-t- #. This also captures sub-word semantics (e.g., prefix & suffix). We also observe that Collision Rate: different words with same letter-trigram representation is very low, as has been empirically found.

2) Convolutional layer: Convolutional layer extracts local features. For Example, it extracts information such as: {w1, w2, w3} represents some topic 1. {w2, w3, w4} represents some topic 2, and so on.

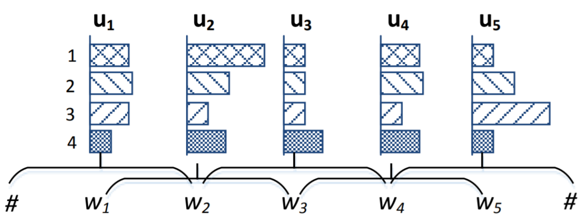


Figure 3: Convolution layer. Ui's denote how much percentage of a particular topic is present in a given word subset.

3) Max Pooling layer: Generate global features using max-pooling. This is done by identifying key or more important topics of the text, as well as key or more important words in the text.

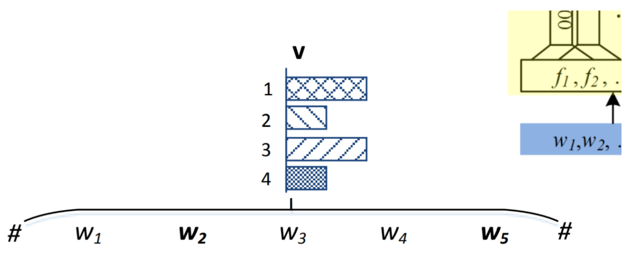


Figure 4: Result of max pool layer

The final results after identifying the key topics and the words that represent them, from a text, look somewhat like this.

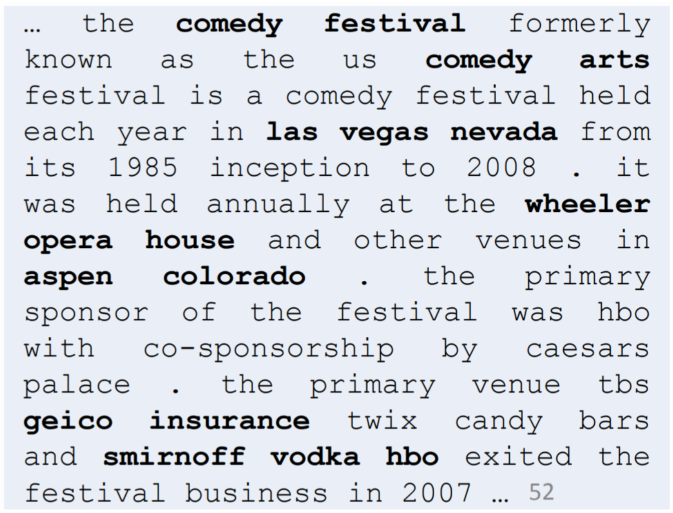

… the **comedy festival** formerly known as the us **comedy arts** festival is a comedy festival held each year in **las vegas nevada** from its 1985 inception to 2008 . it was held annually at the **wheeler opera house** and other venues in **aspen colorado** . the primary sponsor of the festival was hbo with co-sponsorship by caesars palace . the primary venue tbs **geico insurance** twix candy bars and **smirnoff vodka hbo** exited the festival business in 2007 … 52

Figure 5: Highlighted key topics and words that represent them.

4) Intent matching via convolutional-pooling: The learning phase does semantic matching of query and document as follows,

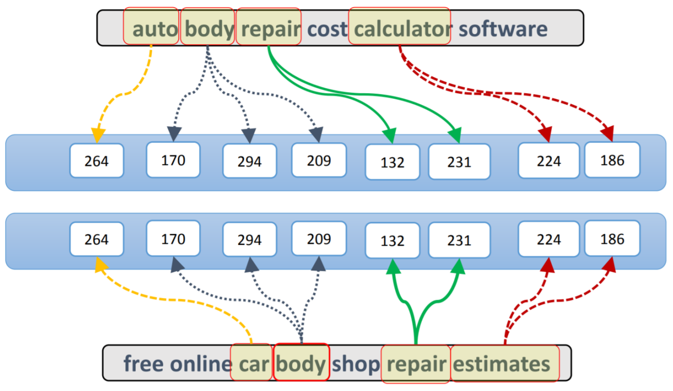

auto body repair cost calculator software
264
170
294
209
132
231
224
186
264
170
294
209
132
231
224
186
free online car body shop repair estimates

Figure 6: Represents most active neurons at the max-pooling layer.

VII. Evaluation Metrics

- Precision: Fraction of retrieved documents that are relevant.
- Recall: Fraction of relevant documents that are retrieved.
- NDCG:
  - For a query q, let relevance level of document ranked j wrt q be $r_q(j)$.
  - $r_q(j)$ means totally irrelevant.
  - Response list is inspected up to rank L.
  - Discounted cumulative gain for query q is

$$NDCG_q = Z_q \sum_{j=1}^{L} \frac{2^{r_q(j)} - 1}{log(1+j)} \quad (1)$$

  - Z_q is a normalization factor that ensures the perfect ordering has $NDCG_q = 1$.
  - Overall $NDCG$ is average of $NDCGq$ over all q

VIII. Results

The results for a test query: **Query:** Show me a spy movie starring tom cruise and is an action thriller.

Truth Value for this query is:

- Mission: Impossible
- Mission: Impossible - Rogue Nation
- Mission: Impossible II
- Mission: Impossible - Ghost Protocol
- Mission: Impossible III
- Jack Reacher
- Spy Game
- Salt

The Results by the system were:

- Mission: Impossible
- Mission: Impossible - Rogue Nation
- Mission: Impossible II
- Knight and Day
- Mission: Impossible - Ghost Protocol
- Mission: Impossible III
- Enigma
- Salt
- Jack Reacher
- The Firm

Precision: 0.7 **Recall:** 0.875

IX. Challenges

- The project inherits all the challenges of text analysis such as language intricacies, and semantic analysis to derive a sufficiently rich representation to capture the relationship between the objects or concepts.
- Handling long semantic queries is computationally expensive.
- All the data was crawled.
- QRel data creation.

X. Conclusion

Movie recommendation is a popular problem. There are a lot of implementations available which are based on characteristics of movies and/or use collaborative filtering method to recommend movies to a user based on the movies watched or liked by other similar users. Here we have used a novel approach by using data from reviews as well and also leveraged the state-of-the art research in document ranking. We have come up with a question answer based interface for movie recommendation which is more flexible and allows the user to be as specific as they want to. Currently we have search and recommendations by keywords and on a much broader level.

References

[1] Jianfeng Gao, *Deep Learning for Web Search and Natural Language Processing*, Microsoft Research, Redmond, USA

[2] Bhaskar Mitra, Nick Craswell, *Neural Models for Information Retrieval*, arXiv:1705.01509v1 3 May 2017.

[3] Michael A. Alcorn, *Deep Semantic Similarity Model*

[4] Prateek Sappadla, Yash Sadhwani, Pranit Arora, *Movie Recommender System.*