Movie Tagging System

Jiatao Fan
University of Michigan - Ann Arbor
Ann Arbor, Michigan 48105
jiataof@umich.edu

Kaifeng Chen
University of Michigan - Ann Arbor
Ann Arbor, Michigan 48104
chenkf@umich.edu

Emily Zhou
University of Michigan - Ann Arbor
Ann Arbor, Michigan 48105
mingyuan@umich.edu

ABSTRACT

When choosing a movie based off of online information, people rely on existing catalog system to filter movies by objective attributes such as year of release, genre, and etc. However, people may look for a movie with specific elements that cannot be discovered by the aforementioned filter. We propose a movie tagging system to extract, store and present keyphrases in movie reviews, as well as use them for indexing a movie when a keyphrase is entered into our system.

CCS CONCEPTS

• **Information Systems → Information Retrieval**; • **Computing methodologies** → *Artificial intelligence*; *Machine learning*; • **Human-centered computing** → Human computer interaction (HCI);

KEYWORDS

natural language processing, text mining, text summarization, automatic keyword extraction, unsupervised learning

ACM Reference format:
Jiatao Fan, Kaifeng Chen, and Emily Zhou. 2017. Movie Tagging System. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 4 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The current movie catalog system adopted by most of the movie information websites uses only a limited number of structured attributes such as year of release, genre, and rating to categorize movies. However, it is observed that when looking for or assessing a movie, most users actually require more concrete categorizations which reflect subjective opinions of viewers. For instance, a user may want to look for movies that feature landscapes in Europe, have beautiful soundtracks, or depict a melancholy romantic story, which are difficult to achieve with a traditional movie catalog system. In such cases, tagging the movies will effectively help users find the movies that match their needs. In this project, we have designed and implemented a movie tagging system based on reviews. Our system aimed to generate tags which can disclose viewers' subject opinions towards movies, so we chose to employ movie reviews as our data resources since reviews contain a large number of subjective comments. By analyzing reviews on the same movie, the system labels movies with phrases which can be descriptions of the content, character or effect of the movie, the performance of the actors and filming technique, or a reflection of the reviewers' sentiment and aesthetics. Through data acquisition, keyphrase extraction model construction and web user interface building, we hope to improve user experience in online movie searching.

2 RELATED WORKS

Automatic keyword extraction algorithms can be summarized into 4 main categories: simple statistical approach, linguistics approach, machine learning approach, and hybrid approach.

Under the class of simple statistical approach, researchers employ term frequency (Tf) [6], term frequency-inverse document frequency (Tf-Idf) [9], word co-occurrences [7] or PAT-tree [2]. For these methods, the frequency of occurrence determines if a word is a keyword or not. They are simple to compute but they generate sub-optimal results. Linguistics approach exploits the linguistic features of the natural language, and including the lexical analysis [1], syntactic analysis [5], discourse analysis [10], etc. Lexical analysis incorporates resources such as WordNet, n-grams, POS pattern, etc, and syntactic analysis incorporates noun phrases, chunks, etc. Machine learning approach for keyword extraction can be categorized into supervised and unsupervised learning. For supervised learning, annotated documents with manually labeled keywords are used for training, and documents whose keywords are to be extracted used for prediction tasks. Some used algorithms include Naive Bayes (NB) [4], support vector machine (SVM) [13], bagging, Hidden Markov model [3], etc. An advantage of the supervised learning approach is that they are easy to evaluate with true labels, but the annotation of labels is costly. As to the unsupervised learning approach, one noteworthy algorithm is the keyword extraction algorithm (KEA) [12]. This algorithm converts text into a graph where each node represents a word and each edge connecting two nodes represents the co-occurrence of two words in the same sentence. With graph-based clustering algorithm, we can conduct clustering using the number of edges between nodes and eventually extract the cluster heads as keywords. Hybrid approach, as its name suggests, combines two or more of the above approaches and exploits their corresponding advantages.

3 METHODS

3.1 System Overview

The overall structure of the Movie Tagging System is illustrated in Figure 1. First, we constructed our movie information database by collecting movie titles, IDs, and images from IMDB's Top Rated Movies list, which contains the 250 most popular movies among the public. For each of the movies, we also fetched their reviews on IMDB. All types of the information above are collected by the web crawler we built with the Python package Beautiful Soup. Second, we built the Keyphrase Extraction Model to obtain "tags" for each movie from reviews. Through pre-processing steps such as tokenization, normalization and lemmatization, unnecessary words from the review texts are filtered out and only words matching subjective syntactic patterns are retained as candidate keywords. Once the candidate list was generated, we ranked the words using

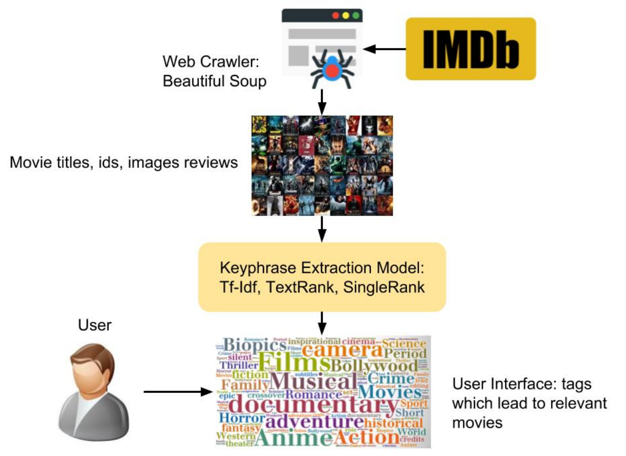
Web Crawler:
Beautiful Soup
IMDb
Movie titles, ids, images reviews
Keyphrase Extraction Model:
Tf-Idf, TextRank, SingleRank
User
User Interface: tags which lead to relevant movies

Figure 1: Movie Tagging System Architecture

4 different algorithms. We implemented both tf-idf based statistical model and graph-based unsupervised learning model in order to compare the performances. The result keyphrases, typically a sequence of nouns and adjectives, are formed by selecting candidate phrases which include one or more of the top-ranked candidate words. Finally, we build a web-based movie search engine using the "movie-tags" dictionary generated in the previous step.

3.2 Data Extraction

Data collection was accomplished using the Beautiful Soup Python package. IMDB lists the 250 top rated movies in a single page. Thus we retrived the list page by its URL and located DOM objects with findAll() function to extract the titles, IDs, and images for the movies and stored them in a csv file. With each of the movie IDs, we formed a URL towards each movie's review page on IMDB. After obtaining each review page, we collected all the review entries in the DOM objects and stored each entry as a single line in a text file. If a movie's reviews are divided into multiple webpages, we also iterated through the page numbers to acquire all the reviews.

3.3 Keyphrase Extraction Model

3.3.1 Candidate Words Selection.

Pre-process: Tokenization, Normalization, Lemmatization In order to only focus on the important words, we performed pre-procesing steps towards the texts with the Python package Natural Language Toolkit. Since our basic lexical unit to analyze is word, we first tokenized the review by segmenting the texts into sentences with nltk.sent_tokenize() and then dividing the sentences into tokens with nltk.word_tokenize(). In order to suppress the noise of commonly used words, we also normalized the texts by removing stopwords. Besides, to eliminate the influence of different forms of a word, we also lemmatized the tokens with NLTK's WordNetLemmatizer. For later processes, we also rendered the words with position of speech tags with NLTK's pos_tag()

Select tokens matching specific POS pattern Given computational costs and the fact that not all words are equally likely to convey its content, we applied two heuristics to further filter out unnecessary words and identify a smaller subset of better candidates. Due to our goal to extract subjective keyphrases, our first heuristic is to limit the words to only noun phrases matching the POS pattern {(<JJ>* <NN.*>+ <IN>)? <JJ>* <NN.*>+}, which matches any number of adjectives followed by at least one noun that may be joined by a preposition to one other adjective(s)+noun(s) sequence. As a comparison, we raised an alternate approach to extract all unigram nouns and adjectives as candidate words.

3.3.2 Candidate Words Ranking (Model Building).

Tf-Idf Tf-Idf, assigns a score to each term in a document by the formula:

$$tfidf(t,d,D) = tf(t,d) \times idf(t,D) = tf(t,d) \times log(\frac{D}{D_t}) \quad (1)$$

where D is number of documents, and D_t is number of documents that contain term t. To compute the Tf-Idf score of each candidate word, we employed the Python package Gensim. We first formed the bag-of-word corpus for the candidate tokens with gensim.copora.Dictionary() and gensim.copora.Dictionary.doc2bow(). Then we directly called gensim.models.TfidfModel() on the corpus, from which we can easily get the Tf-Idf score for each of the candidate word.

TextRank Mihalcea and Tarau [8] proposed the TextRank algorithm which represents a text as a graph. On the graph, each vertex corresponds to a term and the edge between two vertices corresponds to the co-occurrence of two terms. A weight is assigned to an edge by the number of co-occurrences of the two corresponding terms within a specific window size in the text context. For TextRank, the algorithm performs the best when the window size is 2 (meaning edges exist only between adjacent words). For each vertex v_i, the score $S(v_i)$ is initialized as a default value and is computed in an iterative manner until convergence using this recursive formula:

$$S(v_i) = (1-d) + d \times \sum_{v_j \in Adj(v_j)} \frac{w_{ji}}{\sum_{v_k \in Adj(v_j)} w_{jk}} S(v_j) \quad (2)$$

where $Adj(v_i)$ denotes v_i's neighbors. After convergence of the algorithm, we select top-scored terms as keywords. Adjacent keywords are then rallied and output as a keyphrase. According to Mihalcea and Tarau [8], TextRankâĂŹs best performance is achieved when only nouns and adjectives are used to create a uniformly weighted graph for the text under consideration, where an edge connects two word types only if they co-occur within a window of two words. This setting is coherent with our purpose to extract subjective "adjective noun" keyphrases, thus we followed this configuration in our implementation.

SingleRank Wan and Xiao [11] proposed the SingleRank algorithm that is a variation of TextRank. It differs from TextRank in 3 major aspects. First, while each edge in a TextRank graph has the same weight, each edge in a SingleRank graph has a weight equal to the number of times the two corresponding word co-occur in

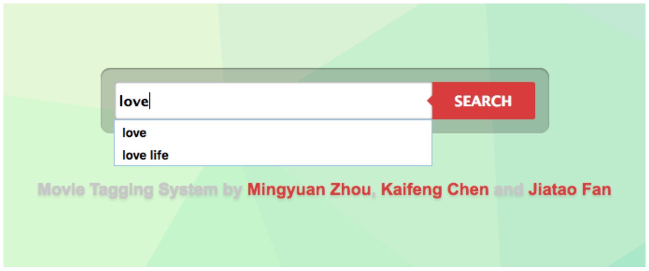
love
SEARCH
love
love life
Movie Tagging System by Mingyuan Zhou, Kaifeng Chen and Jiatao Fan

Figure 2: Search Page

a window. Second, in TextRank only the words with top ranking scores from the ranking formula are kept to form keyphrases. On the other hand, in SingleRank, it does not filter out any low-scored vertices. Rather, it (1) forms candidate keyphrases by assembling any longest-matching sequence of nouns and adjectives in the text and calculate the score of a candidate by summing the scores of its constituent word types obtained from the SingleRank graph. Besides, it eventually (2) outputs the N highest-scored candidates as the keyphrases for the text. Finally, SingleRank employs a window size of 10 rather than 2.

3.3.3 Keyphrase formation.

In Tf-Idf, we extract all the longest n-grams consisting of candidate keywords. Then we score each n-gram by summing Tf-Idf scores of its constituent unigrams. Finally we select top N n-grams as keyphrases. In TextRank, we extract n-grams similarly, and form keyphrases if the n-grams contain one or more of the selected keywords. In SingleRank, we sum the PageRank scores of the n-grams' constituent keywords, and select the top N n-grams as keyphrases.

3.4 Web Interface

We constructed a web-based search engine that allows users to search for movies with a text query of related keywords.

3.4.1 system design. Our system is composed of three parts: the inverted index file, the index server and the Flask web app server.

We used the movie tagging result to create an inverted index file of keyphrases and movies containing the keyphrases. The index server will load the inverted index and use it to process the search queries. A movie is included in the returned list if its keyphrases are a superset of the queried keywords, which are extracted from a space-deliminated query string. The result are returned to our web app via the network and the Python-Flask API.

3.4.2 retrieval example. We enter the keyword "love" into our search bar (Figure 2), which returns a list of movies with their names and posters (Figure 3).

4 EVALUATION AND RESULTS

In order to gain an insight into the 3 algorithms, We report their performance in terms of precision and the number of relevant movies among retrieved movies. We have considered using recall but it can be very hard to determine the ground truth of the collection of relevant movies without intensive annotation labor.

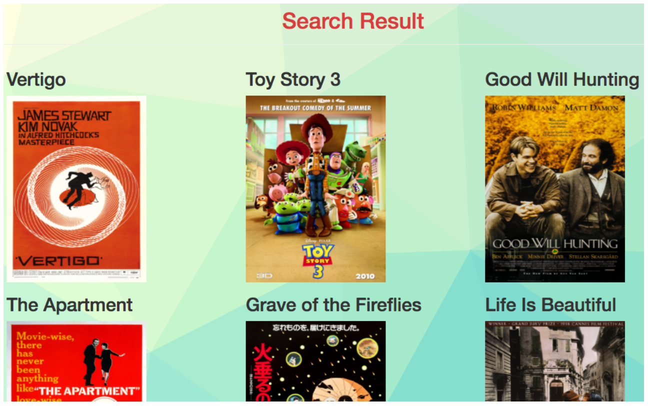

Figure 3: Search Result Example

We use a set of keywords as input to feed into our search engine, and evaluate the precision by computing the ratio of the number of relevant movies to the number of retrieved movies. The set of keywords we use is: love, horror movie, dance, sad, French. The evaluation results are as follows:

Tf-Idf		
query	#relevant	#retrieved
horror movie	2	4
romantic	1	3
dance	2	4
sad	3	4
French	2	2
Total	10	17

TextRank		
query	#relevant	#retrieved
horror movie	2	2
romantic	0	0
dance	2	2
sad	0	0
French	3	3
Total	5	5

SingleRank		
query	#relevant	#retrieved
horror movie	5	5
romantic	14	20
dance	6	10
sad	20	24
French	11	13
Total	51	67

Summary		
Algorithm	precision	#relevant retrieved documents
Tf-Idf	0.59	10
TextRank	1.00	5
SingleRank	0.76	51

TextRank has the the highest precision, which is 100%; SingRank has the highest number of relevant retrieved documents; Tf-Idf is in the middle in terms of the two metrics. However, we should be cautious that TextRank has 0 retrieved result for the two queries we chose, implying 0 recall.

5 DISCUSSION

Our deliverables include implementation and evaluation of three algorithms(Tf-Idf, TextRank, and SingleRank), and a search engine for users to search movies with keywords as input.

The obstacles we encountered throughout the project includes how to effectively extract and assign importance to target words in the review corpus and how to detect termination of a sentence in our keyphrase formation step. Also, learning to use the deep learning toolkits and web development frameworks was a huge challege. Our deliverables include the keyphrase extraction and formation model and the inverted index movie search engine. The code are included in our github repository.[1]

In the future, an idea to improve the current algorithms would be to prevent key phrase formation if keywords span across two independent clauses. Our current algorithm identifies dots in chunking so that key phrases would form until meeting a non-keyword or a dot sign. While we could treat commas in a similar fashion, we should identify the situations where commas are used to separate different words, rather than separate independent clauses, which can be a challenge. We would also like to implement ExpandRank algorithm, an extension of the TextRank algorithm, which exploits neighborhood knowledge and can extract common phrases from topically similar documents. For this algorithm, we would need to extract many more reviews for a movie, and treat each review as a single document rather than combining them like what we did for our implemented algorithms.

In addition to the improvement of methodology, we also need more refined evaluation criteria. We think of employing Amazon Mechanic Turk to annotate our data. We would train workers to label keywords to each of the 250 movies after going through the movie reviews. From there we will be able to use customers' labels as our ground truth to evaluate recall, and F-score for our tasks. For this goal, we raised 2 intuitive strategies. The first is to let workers directly give movie reviews any label they think of, and the second one is to ask them to choose one preset labels we provide for a review.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Qiaozhu Mei and PhD student Wei Ai for providing guidance and support to our project.

REFERENCES

[1] Regina Barzilay and Michael Elhadad. 1997. Using lexical chains for text summarization. In *Proceedings of the ACL/EACL 1997 Workshop on Intelligent Scalable Text Summarization*. 10–17. http://research.microsoft.com/en-us/um/people/cyl/download/papers/lexical-chains.pdf

[2] Lee-Feng Chien. 1997. PAT-tree-based Keyword Extraction for Chinese Information Retrieval. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '97)*. ACM, New York, NY, USA, 50–58. https://doi.org/10.1145/258525.258534

[3] John M. Conroy and Dianne P. O'leary. 2001. Text Summarization via Hidden Markov Models. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '01)*. ACM, New York, NY, USA, 406–407. https://doi.org/10.1145/383952.384042

[4] Eibe Frank, Gordon W. Paynter, Ian H. Witten, Carl Gutwin, and Craig G. Nevill-Manning. 1999. Domain-Specific Keyphrase Extraction. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI '99)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 668–673. http://dl.acm.org/citation.cfm?id=646307.687591

[5] Anette Hulth. 2003. Improved Automatic Keyword Extraction Given More Linguistic Knowledge. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing (EMNLP '03)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 216–223. https://doi.org/10.3115/1119355.1119383

[6] H. P. Luhn. 1958. The Automatic Creation of Literature Abstracts. *IBM J. Res. Dev.* 2, 2 (April 1958), 159–165. https://doi.org/10.1147/rd.22.0159

[7] Y. Matsuo and M. Ishizuka. 2004. Keyword Extraction From A Single Document Using Word Co-Occurrence Statistical Information. *International Journal on Artificial Intelligence Tools* 13 (2004), 2004.

[8] Rada Mihalcea and Paul Tarau. 2004. TextRank: Bringing Order into Texts. In *Proceedings of EMNLP 2004*, Dekang Lin and Dekai Wu (Eds.). Association for Computational Linguistics, Barcelona, Spain, 404–411. http://www.aclweb.org/anthology/W04-3252

[9] Juan Ramos. 1999. Using TF-IDF to Determine Word Relevance in Document Queries. (1999).

[10] Gerard Salton, Amit Singhal, Mandar Mitra, and Chris Buckley. 1997. Automatic Text Structuring and Summarization. *Inf. Process. Manage.* 33, 2 (March 1997), 193–207. https://doi.org/10.1016/S0306-4573(96)00062-3

[11] Xiaojun Wan and Jianguo Xiao. 2008. Single Document Keyphrase Extraction Using Neighborhood Knowledge. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2 (AAAI'08)*. AAAI Press, 855–860. http://dl.acm.org/citation.cfm?id=1620163.1620205

[12] Ian H. Witten, Gordon W. Paynter, Eibe Frank, Carl Gutwin, and Craig G. Nevill-Manning. 1999. KEA: Practical Automatic Keyphrase Extraction. In *Proceedings of the Fourth ACM Conference on Digital Libraries (DL '99)*. ACM, New York, NY, USA, 254–255. https://doi.org/10.1145/313238.313437

[13] Kuo Zhang, Hui Xu, Jie Tang, and Juanzi Li. 2006. Keyword Extraction Using Support Vector Machine. In *Proceedings of the 7th International Conference on Advances in Web-Age Information Management (WAIM '06)*. Springer-Verlag, Berlin, Heidelberg, 85–96. https://doi.org/10.1007/11775300_8

[1]https://github.com/mingyuanz/eecs549_final_project