Eindhoven University of Technology

Title goes here

Stefan Patelski

November 10, 2013

1 Introduction

1.1 Crawler : S.R. Patra

1.1.1 Motivation

The very first task was to collect data for our project. We decided to implement a focused crawler for Rotten tomatoes website to collect information about different movies. Large collection of movies with credible reviews made Rotten Tomatoes a suitable candidate for us to get our data from. The goal was to start with a seed URL and extract all the yet unseen URLs on that HTML page along with the metadata about the movie such as movie name, director, producers,reviews etc. The process needs to be repeated for all the collected links.

1.1.2 Approach

For each page, the method *CollectData* in the crawler read the HTML content line by line, only collecting the relevant information and disregarding all the other content that were not relevant to us. This was done with the help of an HTML parser which used *Jsoup* library functions. Whenever relevant data were encountered, they were appended to an XML document. To find the movie URLs , Java's pattern matching functionality was being used which looks for a particular pattern of anchor texts which are specific to movie pages. The crawler architecture is shown in fig.1.

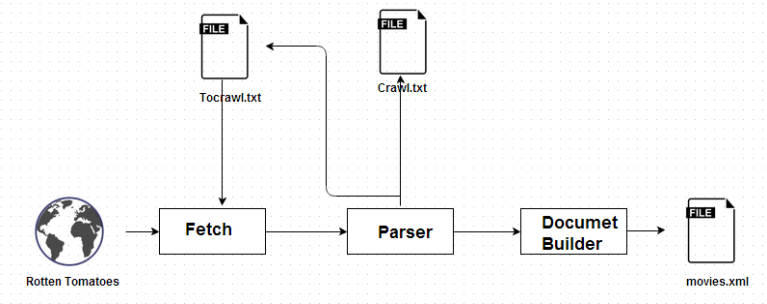

Figure 1: Crawler Architecture

1.1.3 Challenges

The main challenge was to encounter the non uniformity of the content in Rotten Tomatoes web pages. Some of the web pages contained all the information about the movies while in others, a few of the information fields were missing. The crawler was designed to scrap the HTML data in a way such that maximum amount of information can be retrieved. Another challenge was to make sure that the crawler is not blocked by the site. This

was handled by building the crawler in a flexible way so that, during an iteration, the crawler will only crawl through a specific number of pages specified to it as a parameter. After each iteration, the URLs which are already crawlled and the URLs those are yet to be crawlled are written back to *Crawl.txt* and *Tocrawl.txt* respectively which will be used in the next iterations. Fig.2 shows an entry in the *movies.xml* file.

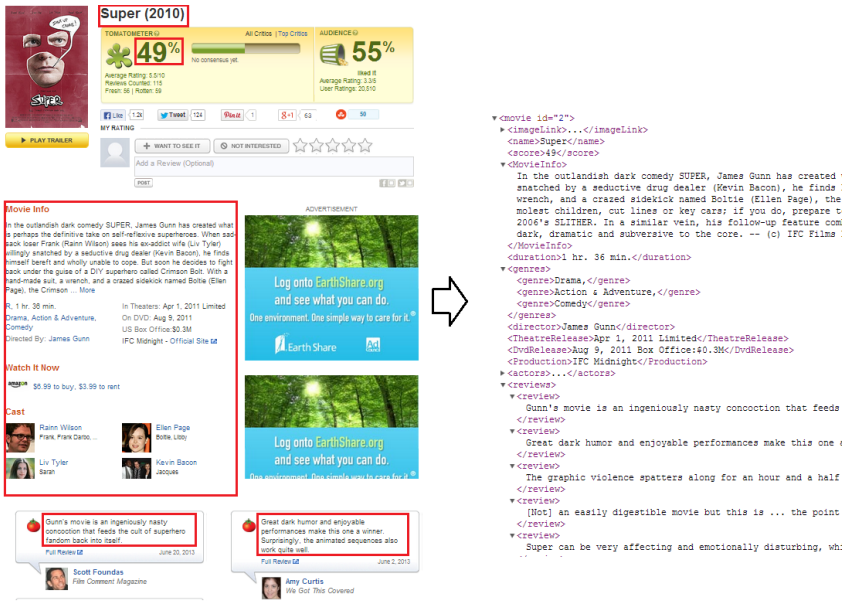

Figure 2: Collecting data from Rotten Tomatoes

1.1.4 Evaluation & Summary

After several iterations, we collected 6227 movies each containing movie metadata and the reviews. We collected only top critics reviews because we feel that those are the most representative of all the reviews. As focused crawler parse all the content in documents and filter only the relevant content for the application, crawling took more time than a normal crawler would take. But the flexibility in the design of the crawler allowed us to run it in iterations so we did not face any issue regarding the time for crawling. We took 20 random samples from the file and compared the contents with the content in the site. In all the cases, we were able to capture all the relevant data needed for the search and sentiment analysis functionality.

1.2 Lucene Search, Wildcard Queries & Autocomplete : S.R. Patra

1.2.1 Motivation

Lucene provides very rich library functions to implement indexing and search. So we implemented lucene search as a metric to evaluate our boolean and vector space model performances.

1.2.2 Approach

Lucene.net library was used for generating the index and for getting the search results from the processed query. We limited our search results to be 10 top relevant results. Index was created for different fields to allow the user to search based on any criteria they want i.e. based on actor, director, genres etc. We also implemented the *Autocomplete* feature to help the user with the query. This feature was developed in *JQuery* and was implemented as a web service which queries the database about possible movie names after each input letter in the text box.

Wild card queries were used to provide query suggestions to the user when the user only knows parts of the terms in the query. The speech to text query assistance feature was also included using the HTML5 Webkit speech. All the features discussed above are shown in fig.3.

Figure 3: Searching functionalities

1.2.3 Evaluation & Summary

After trying a number of sample queries for different search criterias, we feel that the search provided quite relevant results in all the cases. The wildcard queries were also accurate in their suggestions and the users were able to retrieve relevant movie names based on parts of the search query terms.

1.3 Paragraph Level Sentiment Analysis : S.R. Patra

1.3.1 Motivation

Performing sentiment analysis on movie reviews have lot of advantages. Most of the users take opinion from others before going for a movie. If they want to research about a movie online, they have to spend time in reading the reviews or watching reviews in videos. In this context, a sentiment analyzer can help in saving time as it gives the users a score which represents the overall opinion of others about the movie. We also wanted to perform sentiment analysis on YouTube videos of different movie reviews because a large number of people also prefer to watch video reviews.

1.3.2 Approach

We perform sentiment analysis task based on a naive Bayes classifier. It was implemented using a web service *uClassify*[1] where the classifier *ReviewClassifier* was trained with a dataset containing 1000 positive and 1000 negative review text files[2]. After the training, the classifier had 703414 negative features (words) whereof 37640 were unique and 798382 features whereof 40173 were unique. For each of these words, the classifier calculates polarity values which is basically a combination of a *positive* and a *negative* score. The probability of a document being in a class, which is basically the product of the probabilities of all the terms in the document being in that class, is calculated using the *Bayes theorem*. It can be better understood using the following example. Let us assume that a test document D has 3 terms *d1, d2 and d3*. The classifier calculates the probabilities of the document being in both the classes *Positive* and *Negative* and classifies them to be of the class having the higher probability score.

$$P(D|Positive) = P(d1|Positive) * P(d2|Positive) * P(d3|Positive)$$

$$P(D|Negative) = P(d1|Negative) * P(d2|Negative) * P(d3|Negative)$$

If $P(D|Positive) > P(D|Negative)$, then the document is classified as *Positive* else it is considered *Negative*.

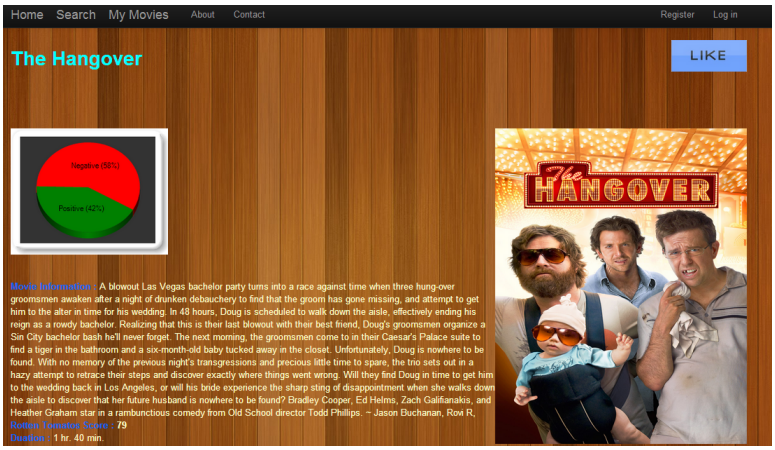

Figure 4: Positive and Negative scores of the movie

We performed sentiment analysis on all the 6227 movies and probability scores are displayed in the page for each movie as shown in fig.4. To make our project more interactive, we also provide a functionality where the user can write a review of his own and get the polarity scores for the review. Additionally he has the option to *submit* the review to the database as shown in fig.5.

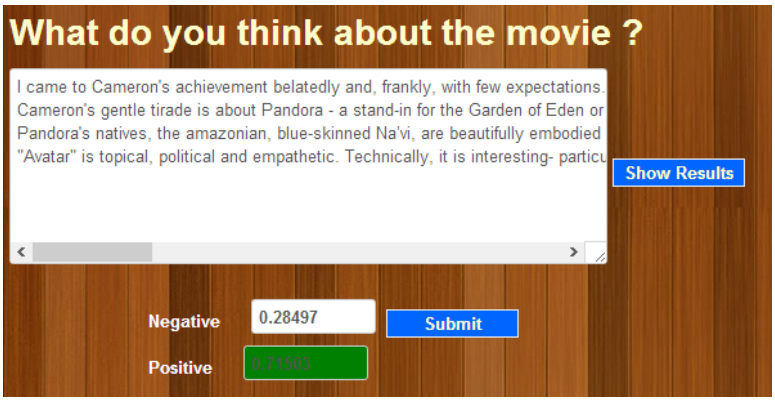

Figure 5: User Review with option to submit

We did not find any good dataset containing video reviews,so in our application, we decided to use YouTube API to display top 6 video reviews of the movie. To analyze the content of these videos, we take the help of YouTube Closed Caption feature. Integrating *Google2SRT* [3] helped us get the closed captions of the video and perform sentiment analysis on them. Fig6 displays a sample showing how the captions from the video are converted to text and how the scores are displayed post analysis.

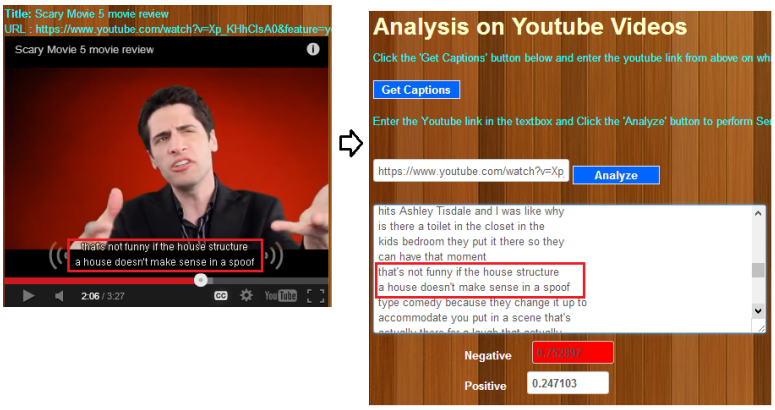

Figure 6: Sentiment analysis on youtube videos

1.3.3 Challenges

Performing sentiment analysis on YouTube videos itself is a really challenging task. because Closed caption is a new feature in YouTube and most of the older review videos are simply transcribed based on Speech to Text recognition technologies which are not always so accurate. Also, in videos, people generally tend to compare the movie with other movies which really affects the score. For example, if a user did not like previous X-Men movies but liked the new one and for more than half part of the video, he criticizes the previous parts, the video will be classified negative which should not be the case.

1.3.4 Evaluation & Summary

For evaluation, we chose a metric where movies with positive score greater than 65% were considered *Positive* and less than 65% were considered *Negative*. From the Rotten Tomatoes website, we collected the percentage of positive reviews in the attribute *score* and used the same metric for these scores. We were able to achieve 77.69% Accuracy. Details are shown in fig.7. Considering the fact that we only collected the representative set of top critics reviews and not the complete set of reviews, we believe the results are quite satisfactory. Evaluation on YouTube videos were performed manually. We watched 10 random reviews and classified those videos. After comparing them with the scores from the classifier, 7 of them were correctly classified whereas 3 were classified in the wrong class.

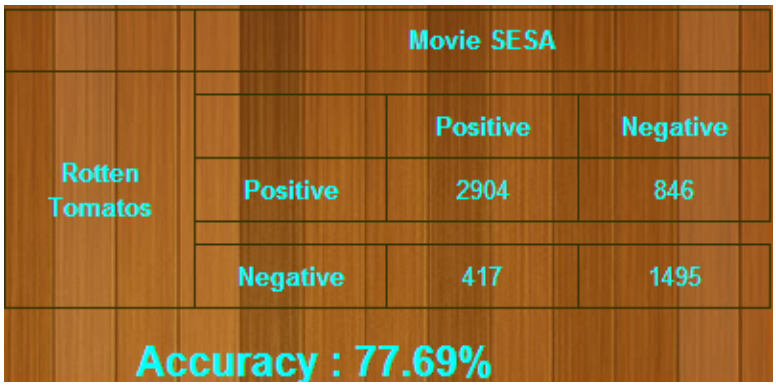

Figure 7: Confusion Matrix of the classifier

We also thought it would be interesting to analyze how popular are movies from different *genres* . Figure 8 shows the results of our analysis.It can be seen that Drama movies are really popular with the highest number of positive reviews.

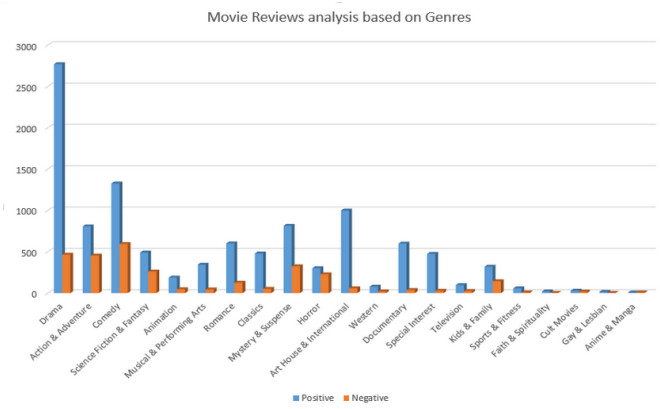

Figure 8: Movie reviews analysis based on genres

Figure 9 shows the percentage of positive and negative reviews per genre.

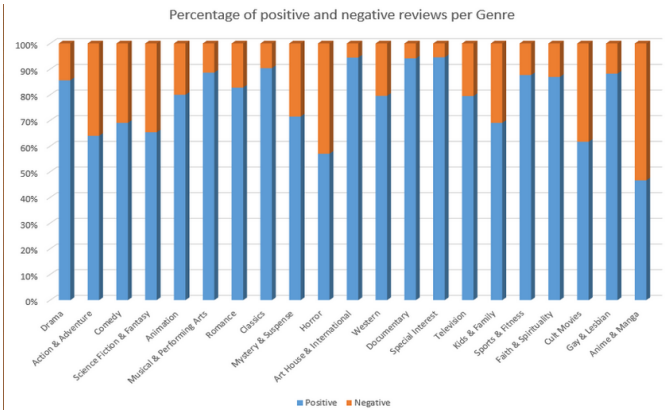

Figure 9: Percentage of Positive and Negative reviews per genre

1.4 User Modeling & Personalized Recommendations : S.R. Patra

1.4.1 Motivation

User modeling has become an integral part of web information systems these days. Personalized recommendations has great impact in attracting new users to the site, because personalized approach has the best chance of predicting user's likes and dislikes and provide relevant recommendations. This motivated us to add a personalized recommendation component in our project.

1.4.2 Approach

Our approach is to obtain explicit feedback from the user about his fondness of the movies. We do this by providing a *Like* button on each movie's page and by clicking this button , the user lets us know that (s)he likes the movie. For each user we store this information in a database table *UserLikeMovies* and the user can always see the list of movies that he has liked in a separate page called *My Movies*. Figure10 shows how the component was integrated in our project. As can be seen from the figure, we use pie chart representation to display the percentage distribution of the movies user liked from different genres. The genre that the user is most inclined towards is considered to be his category. We recommend the best 20 movies (based on the sentiment analysis scores) in that genre to the user. If the user is inclined towards 2 genres, he will get recommendations for both the genres. Also,gradually, if the user likes other kind of movies, his category will change and he will get recommendations from the new category.

Figure 10: Personalized recommendations

1.4.3 Summary

Our method could easily be modified to accommodate *Fixed Training Window* approach to handle concept drift. Instead of performing the analysis on all the movies the user has liked, if we perform analysis on only last few movies (s)he liked and recommend movies based on that analysis, we can detect a concept drift and change user's category immediately to recommend movies based on new category.

References

[1] http://www.uclassify.com/

[2] http://www.cs.cornell.edu/people/pabo/movie-review-data/

[3] http://www.techreviewsdaily.com/how-to/google2srt-download-subtitles-from-youtube-videos-to-subrip-srt/