Movie Recommendation System

Anthony Depace, Vidul Ayakulangara Panickan, Sai Venkata Maneesh Tipirineni

Abstract

This paper presents the results of applying multiple natural language processing algorithms to textual movie data in an attempt to build a unique movie recommendation pipeline. The paper is divided into the three sub-tasks of finding similar movies, classifying movie genre, and predicting movie rating. The data used in our experiments consists of plot summaries, genre tags, and customer reviews for just over 11,000 movies and 680,000 reviews. At the end of the paper we discuss how the best-performing models of each task were integrated to produce meaningful movie recommendations. The resulting pipeline utilizes a combination of Linear Support Vector Machines, K-Nearest-Neighbors, and Multinomial Naive Bayes models to make movie recommendations.

Introduction

The famous Netflix contest (http://www.netflixprize.com/rules.html) awarded 1 million dollars to the team that could improve Netflix's recommendation accuracy by 10% based on what movies a user previously liked/disliked. Their data-set did not take contain movie plot. Because we have user-specific rating information and the plot of the movies, we are interested to see if plot summaries would be a helpful feature.

In most of the online movie databases, tagging a movie with its appropriate genres and rating the movies is a manual process and is subjected to errors. With an automated movie classification and rating system, this process becomes much faster and the error rate can be reduced drastically

We tried to build a system where the user can search for a movie from and our system will generate a rating, assign relevant genres and display other movies with similar plots. For a particular movie, the system will also recommend other movies the user may be interested in.

For building such a system, solving the following three NLP tasks are necessary:

1. Classifying movies into genres based on plot and reviews.

2. Identifying movies with similar plots.

3. Assigning a rating to a movie based on analysis of its reviews.

We used existing data sets and NLP/machine learning libraries to develop our pipelines.

This report will be structured as follows. First we discuss related previous works in the domain and briefly detail their approach to the same problem. Then we explain the methods and models we used to achieve our goal. Later, we look into the results and analyze the models and how they performed on different tasks. Lastly, we discuss how the systems used for different subtasks might be improved in the future.

Related work

1) Ho, Ka-Wing. "Movies' Genres Classification by Synopsis."

The paper describes 4 different methods for automated movie genre classification based on synopsis and one of them employs one-vs-all strategy with SVM. We followed a similar approach and were able to obtain a higher F Measure when we used two classifiers (Linear SVM and Multinomial Naïve Bayes) instead of one.

2) Raschka, Sebastian. "Naive Bayes and Text Classification I-Introduction and Theory." *arXiv preprint arXiv:1410.5329* (2014)

This paper gives an overview on how Naïve Bayes is used for text classification. Section 2.2 of the paper describes how posterior probabilities are used for classification in much detail. This inspired us to use the posterior probabilities for multilabel classification.

3) Vilar, David, María José Castro, and Emilio Sanchis. "Multi-label text classification using multinomial models." *Advances in Natural Language Processing*.

In our multilabel classification model using Naive Bayes, we always had to predefine the number of labels to be predicted. This paper describes a better approach where they use posterior probabilities for multi label classification without any such restriction.

4) Jean Y. and Yuanyuan P. "Predicting Sentiment from Rotten Tomatoes Movie Reviews"

The authors of this paper experimented with different classifiers and machine learning algorithms to predict the sentiment of unseen reviews. They used the rotten tomatoes movie review corpus and used Naïve Bayes and SVM models to classify reviews and also experimented with Neural Networks and Deep Learning models but couldn't achieve better performance than SVM.

5) Soo-Min Kim, Eduard Hovy. "Determining the Sentiment of Opinions"

This paper presents a system that, given a topic, automatically finds the people who hold opinions about that topic and the sentiment of each opinion. It also experiments with various models of classifying and combining sentiment at word and sentence levels. The system uses POS_tagging, NER and WordNet as preprocessing steps and classifies the reviews into different parts of speech and applies different sentiment classifiers based on the POS_tags and later combines them using a sentence sentiment classifier.

6) Wang, S., & Manning, C. (2012). "Baselines and Bigrams: Simple, Good Sentiment and Topic Classification." *In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics.*

This paper analyzes the effect of inclusion of bigram word features on Naïve Bayes and SVM classifier in the context of sentiment analysis and compares their performance on different datasets. The paper presents a novel variant of SVM which uses Naive Bayes log count ratio as features for the SVM and outperforms most state-of-the-art models.

7) Pang, B., & Lee, L. (2005). "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales." *In Annual meeting-association for computational linguistics.*

This paper tries to find a suitable classifier which can classify reviews on a 1-5 scale, just like our Movie rating system. All the other related papers looks at sentiment analysis as a positive vs negative problem and do not consider fine-grained rating systems. The authors experiment with SVM and regression models and in addition consider the problem from a metric labeling system which is a special case of Maximum a posteriori estimation problem for Markov Random Fields.

8) Sriram, Bharath, et al. "Short text classification in twitter to improve information filtering." *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2010.

For genre classification, we used bag of words as features. Feature extraction was done as a part of text preprocessing where tokens less than a particular frequency (10) and less than a particular size (3) were removed. This paper provides a new methodology which uses the domain knowledge of the classes for feature extraction. For example a feature for discriminating an action genre would be the presence of words such as fighting, shooting, chasing, guns etc. This methodology can be incorporated in our system to improve the overall classification accuracy.

9) Michael Steinbach, George Karypis, Vipin Kumar. *"A Comparison of Document Clustering Techniques"*

This paper compares several common algorithms for document clustering. Each plot summary in our data set can be considered a document. Topics discussed in the paper include how to represent a corpus of words as a feature vector, distance metrics, evaluating cluster quality, and how different models compare.

10) Kleedorfer, Florian, Peter Knees, and Tim Pohle. "Oh Oh Oh Whoah! Towards Automatic Topic Detection In Song Lyrics." *ISMIR*. 2008.

This paper discusses an algorithm for grouping song lyrics by topic. Their data consists of song lyrics and 31 genre tags. Similarly, the data we use in our project to find similar movies contains plot summaries with genre tags. Our task of finding similar movies is essentially grouping the movie plots by their topic.

Data

We formed our dataset from two sources, the CMU Movie summary corpus[1] containing movie plot summaries for 42,306 movies and Amazon movie reviews dataset[2] which contain 4,607,047 movie reviews and ratings for around 100k movies. Our dataset was derived from these datasets by extracting movies with plots, genre and review data.

The only feature shared between the two data sets was movie name, however many of the names were not exact matches. A common issue with the Amazon movie reviews is that the movie names would have extra information on them such as 'DVD', 'VHS', 'Blu-ray'. Using simple regular expressions to find and delete these noisy occurrences, we were able to find over 11,000 exact name matches between our two data sets, allowing us to join the two data sets on the movie name feature. We feel that this data should be enough to form a training/testing set for our experiments. If we decide we need more data later we may go back and try more advanced methods of matching movie names (comparing minimum edit distance, radix sorting the movie names & manually inspecting, locality sensitive hashing, etc.)

Our dataset has 11343 movies and has a corpus size of 210MB. Below are the other key details about our dataset.

TOTAL MOVIES: **11343**

MOVIES IN TRAINING SET: **9075**

MOVIES IN TESTING SET: **2268**

TOTAL NUMBER OF REVIEWS: **689152**

REVIEWS IN TRAINING SET: **552866**

REVIEWS IN TESTING SET: **136286**

NUMBER OF GENRES: **319**

NUMBER OF GENRES IN TRAINING SET: **310**

Name	Number of Movies	Name	Number of Movies	Name	Number of Movies
Drama	5791	Action / Adventure	1653	Black-and-white	1081
Comedy	3286	World cinema	1631	Adventure	1055
Thriller	2492	Crime Fiction	1574	Romantic Drama	981
Romance Film	2177	Horror	1487	Science Fiction	888
Action	1982	Indie	1426	Romantic Comedy	824

Table showing most common genres and the number of movies with the corresponding genre.

For the genre classification task we reduced our 310 genres into exactly 22 genre classes. This involved splitting genre labels such as "Romantic Comedy" into the two existing labels "Romance" and "Comedy". This allowed us to achieve better accuracy in genre classification.

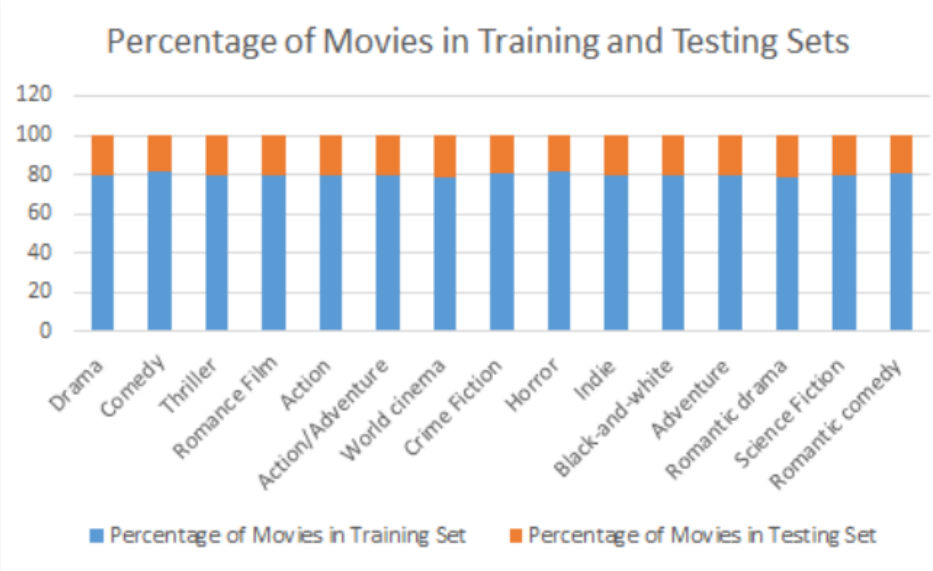
Rating	Number of Reviews

1	49711
2	35137
3	64560
4	141523
5	398221

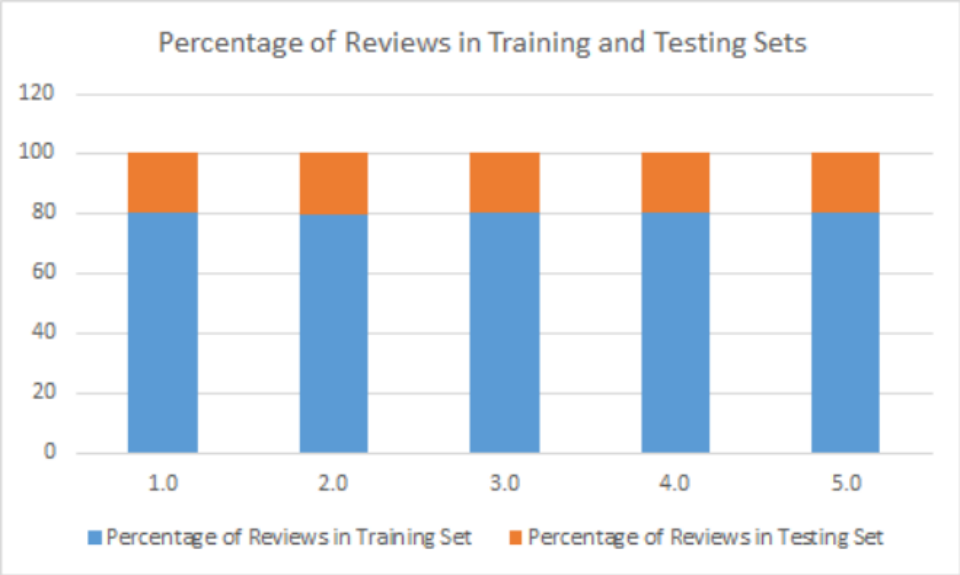
Table showing ratings and the number of reviews with the corresponding rating (both training and testing sets included)

Training/Testing Data Split

The training and testing data was roughly split in the ratio 4:1. The exact details are shared in the dataset section. Other key details regarding the training and testing datasets are displayed below.



Percentage of Movies split into Training and Testing Sets for top 15 popular genres



Percentage of Reviews split into Training and Testing Sets

Method

Genre Classification With Plot Summaries

Movies can be classified based on similarities in their topic, mood, target audience, length, format, type of production etc. Each of these aspects can categorize a single movie into multiple genres. For example, a movie can be a science fiction (topic) war film (setting) shot in widescreen (format) and produced using a big-budget (production type) targeted at teens (target audience).

Generally movies only have a few genres associated with them (typically 2 to 4). This is because, movies are classified only on certain aspects that are prominent (which the viewers really care about such as setting, topic and mood). For this purpose, we generated a new dataset where movies were tagged solely based on these prominent aspects.

Further on manual inspection of genres in our training data, we noticed certain trends:

A number of genres were a combination of two other primary genres (such as *action/adventure*: action, adventure. *romantic drama*: romance, drama etc). We split these compound tags into their constituent genres.

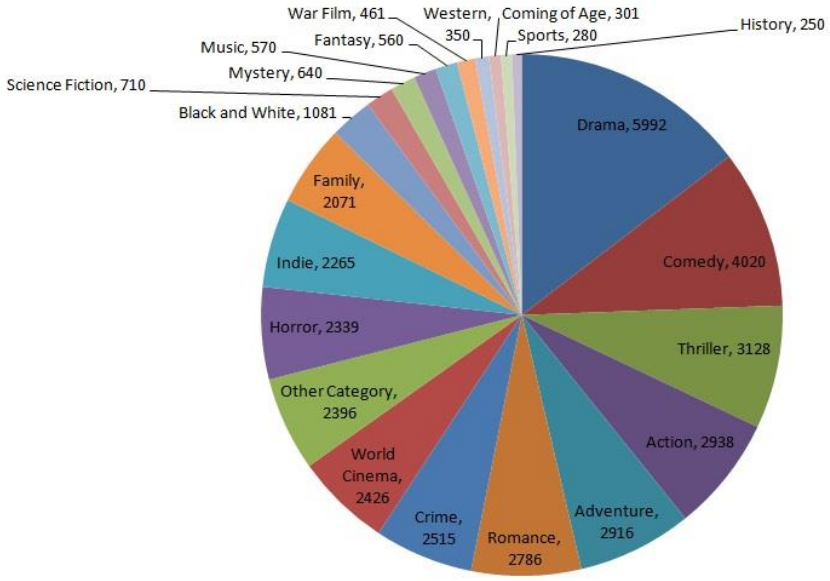
Most genres were found to be subcategories of certain main genres. These sub genres were replaced with their main genre (*'comedy'*: parody, satire, media satire. *'horror'*: slasher, monster '*action'*: martial arts film, swashbuckler films, superhero etc.)

Few other genres were found to occur together frequently. For example movies tagged with *animation* also had the tag *family* in most cases. This is because certain genres share some common elements that make them appear together. So we grouped them under their common genre (*'family'*: animation, children's. *'thriller'*: suspense etc)

Certain genres occurred very rarely (less than 10-15 times). Such low frequency genres were grouped into the 'Other Category'. Due to this grouping, a number of movies were tagged with only the ‘other category’ genre. These movies were removed from training data and were added to the test data.

Thus after splitting, replacing and grouping, the initial 319 genres were reduced to 22 genres.

Genres and the number of movies under each genre



Number of movies under each genre after splitting, replacing and grouping,

After processing the data, we planned to experiment with different classifier models. We built a naïve tokenizer to extract tokens from training data. Then we removed tokens whose length were less than 3 to remove special characters and words that fall under prepositions and determiners (words that won't convey any special meaning like: be,do,a,an,at,to,on). Further we removed words whose frequency was less than 10 to remove infrequent tokens. Further we generated a new list of stopwords by combining MySQL stopwords and Google stop-words and removed them from our vocabulary.

As baseline algorithm, we build a multinomial naïve bayes model for genre classification. Since different movies can have different number of genres this was a multi-lable classification problem. Our first approach was to adopt the one-vs-rest strategy but our model turned out to have a high time complexity hence employing it in a one-vs-rest strategy was not feasible. So we came up with a new approach to do multilabel classification using multinomial Naive Bayes.

Multilabel Classification Using Multinomial Naive Bayes:
In our implementation of naïve bayes model, we generate posterior probabilities for all genres for a movie and select the genre that gives the highest posterior probability. So, for predicting n number of genres for a given movie, we find the top n highest posterior probabilities and select their corresponding genres.

To obtain better results, we tried using several other classifiers that were imported from scikit-learn. Since these classifiers were performing far better in terms of running time, we were able to adopt the one-vs-rest approach.

We implemented the one-vs-rest strategy (training 22 classifiers (one for each genre))with each of the following classifiers and observed their performance on the testing data.

- Perceptron with 100 passes over the training data (epochs)
- SGDClassifier(Linear Support Vector Machine Classifier trained using stochastic gradient descent) with penalty set to elasticnet and 50 passes over training data
- Linear SVC with parameters: loss='l2', penalty='l1', dual=False, tol=1e-3
- K Neighbors classifier with 10 neighbors
- Nearest Centroid classifier
- Multinomial Naïve Bayes with alpha set to 1(found out experimentally)

Finally we compared the Precision and Recall of the different classifiers to select the best classifier for predicting genre.

Finding Similar Movies With Plot Summaries

Given a movie, we tried to predict similar movies based on the movie plot summaries. This task forms the integral part of our movie recommendation system. The main challenge we faced with this task is that there lacks a clear criteria defining the similarity of one movie to another, or to what degree the movies are similar. Our initial evaluation metric of movie similarity was based on the shared genre tags of the movies in question. More formally we evaluated movie X of genre Y as being more similar to movies of genre Y than movies of some other genre. This simple metric is especially attractive because all movies in our dataset have genre tags. Additionally, this allowed for movie genre distributions to be used as an external quality measure for clustering experiments.

By manual inspection of plot summaries and using our own opinions on movie similarity, we observed that very similar movies shared at least 1 genre tag. While this was somewhat meaningful, Intuitively we knew that sharing 1 genre tag was not sufficient information to conclude that a set of movies are very similar. These observations motivated a multi-phase model evaluation procedure which works as follows:

(1) Manually pick a subset of movies from our data that we have seen and are very familiar with.

(2) For each familiar movie, attempt to find similar movies via the model that is being evaluated.

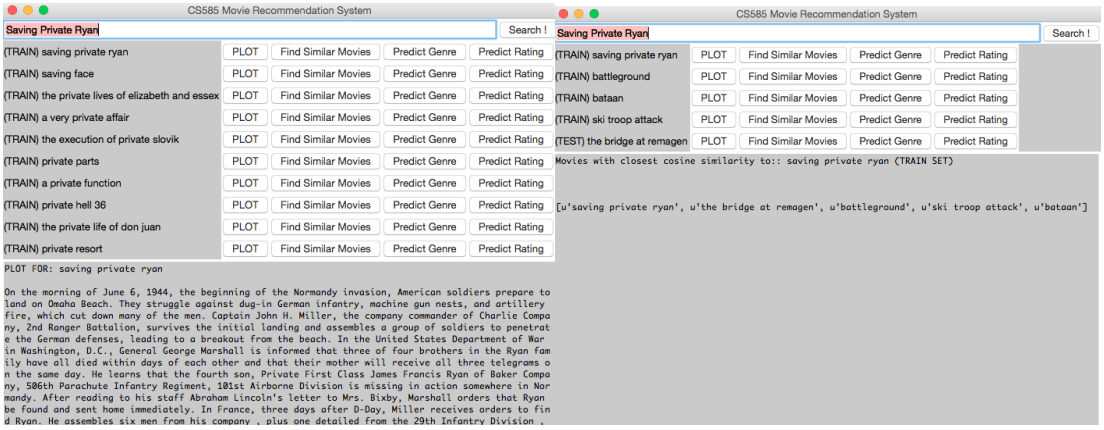
(**3)** For each predicted set of similar movies, if the movies share few or no genre tags with the corresponding familiar movie, reject the model, otherwise continue to step 4.

(4) Manually inspect the plot summaries of the returned movies and using our best human judgment decide if the movies are truly similar to the familiar movie.

(5) After judging how the model worked for each familiar movie, accept or reject the performance of model.

To facilitate the process of manually inspecting the movie data, we developed a simple python application for searching for movies, displaying plot summaries, and finding similar movies with a model.

Data & Model Visualization Application



Search, plot summary, and similar movie prediction for 'Saving Private Ryan'

This application drastically improved the efficiency of our manual evaluations and we also added support for the genre and rating prediction tasks.

The primary models we tested for finding similar movies were K-Means clustering and K-Nearest-Neighbors searching. A study on document clustering found hierarchical clustering to produce higher quality clusters than K-Means, but due to the size of our dataset we decided to avoid models with quadratic runtime or higher.[3]

We chose to represent our plot summaries using normalized term frequency (TF-IDF) vectors. When creating the vectors, we experimented with various length terms (n-grams) and filtered common English words such as 'the' and 'a'. The metric we used for comparison of the plot summary vectors was Cosine Similarity.

Predicting Movie rating

In this section, we try to assign ratings based on the text of a review. The dataset has 80% training data and 20% testing data and has 600,000 reviews overall. We used a Multinomial Naïve Bayes classifier for classifying the reviews into rating classes. As will be evident from the results below, the multinomial NB classifier does not do a great job at classifying the reviews. There is only a minimal improvement over the baseline using Naïve Bayes model.

So we tried many other models and with different preprocessing steps. We experimented with the following classification models:

- Perceptron algorithm (with 10 iterations).
- Naïve Bayes Model with bigram features (top 25,000 bigrams).
- Linear SVM (SGDClassifier in scikit-learn).
- Linear SVM with bigram features(top 25,000 bigrams).

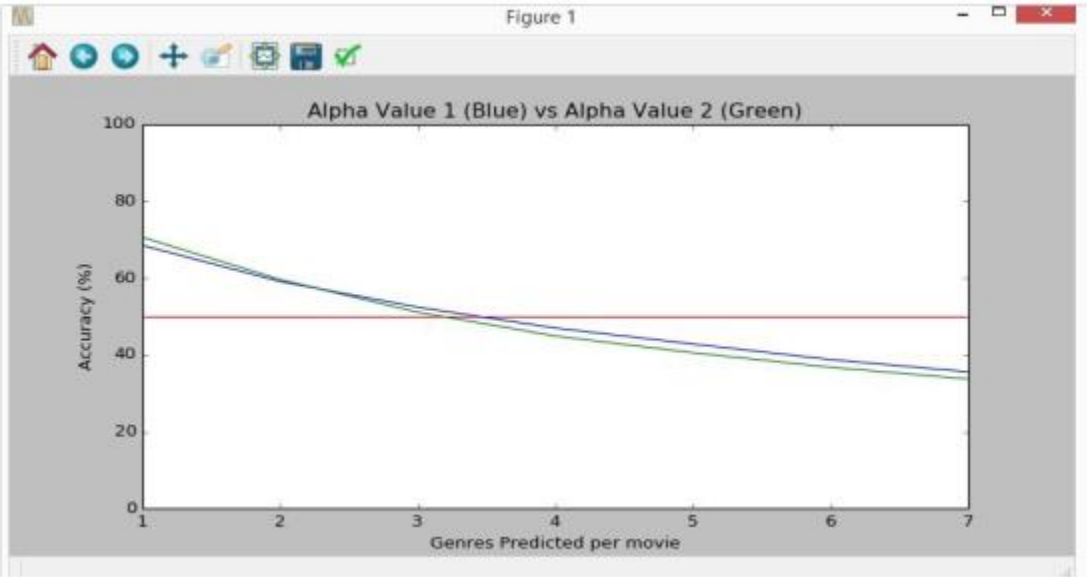
We removed stop-words and stemmed all the words in the review. We found that a lot of noise crept in because of the huge volume of data and tried to reduce it. We only considered only which occurred at least 5 times in the corpus. All low frequency words are basically noise as they are never seen again and are of very little use to the classification model. We tried using Principal Component Analysis to use only the top discriminating features for classification and this improved performance which confirmed our suspicion. All our classification models were imported

from the Scikit-Learn python library. We considered precision as our evaluation metric for measuring the performance of our classifiers.

Results

Genre Classification

Our Naïve Bayes model gave the best accuracy for alpha values 1 and 2. The following figure shows how the accuracy varies with the number of the genres predicted for movies in test data.



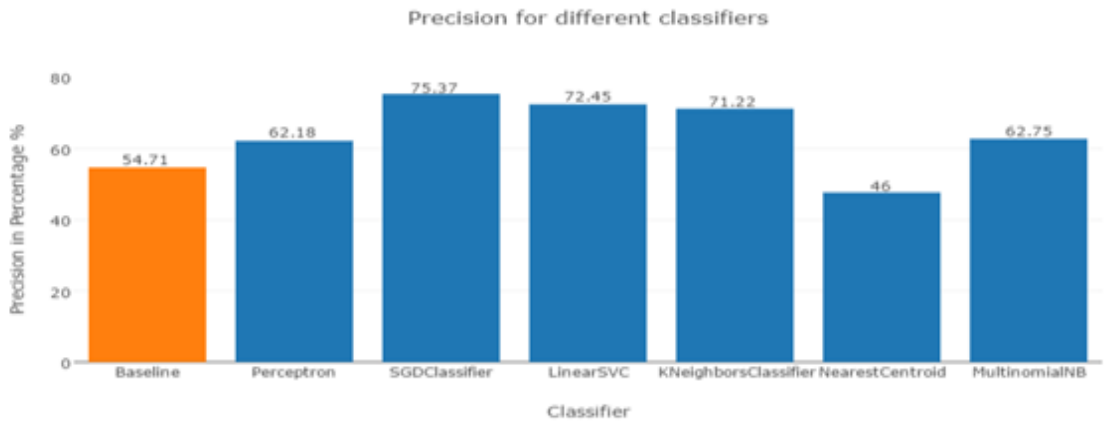
Variation of accuracy with number of genres

It can be observed from the graph that as the number of genres being predicted increases the accuracy decreases. Further for higher number of predicted genres, the model performs better on our test data when alpha is 1.

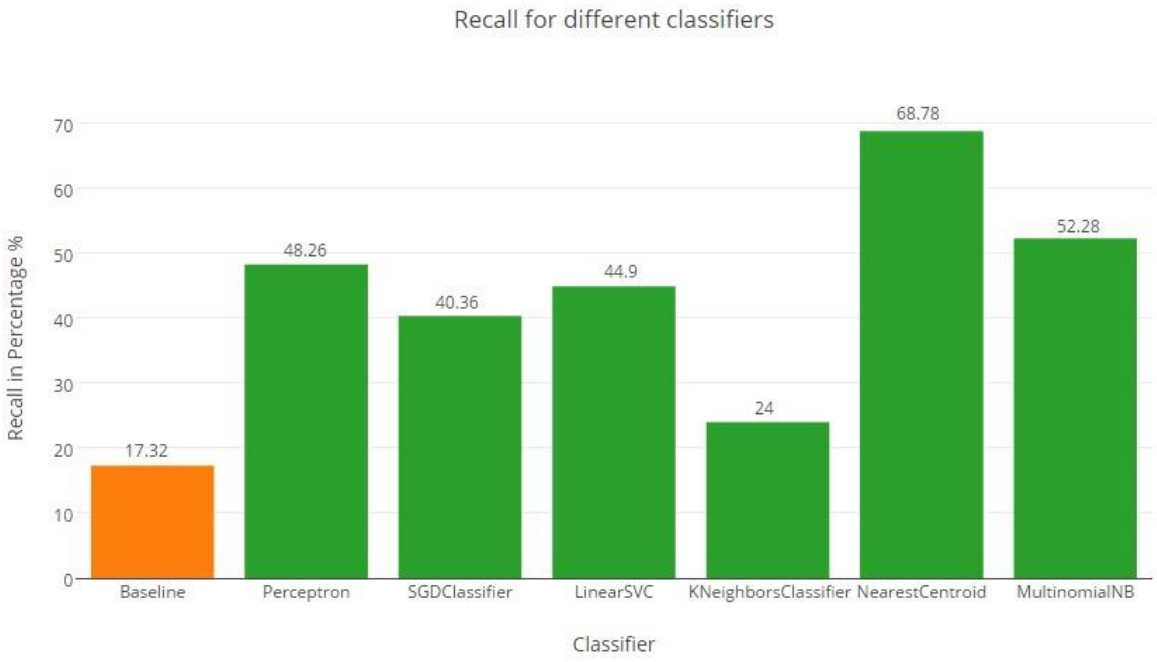
Since the average number of genres per movie in the training data is found to be 4.63. We generated 5 tags for each movie in the test data. The resulting accuracy was around 40%. Due to the low accuracy, we had to discard this model. However we found the optimal value of alpha to be 1 for our data set and we use the same value in the scikit-learn implementation of multinomial naïve bayes.

Experimenting the One-Vs-The-Rest strategy

Baseline: We predicted ‘Drama’ which was the most common genre for every movie in the test data.

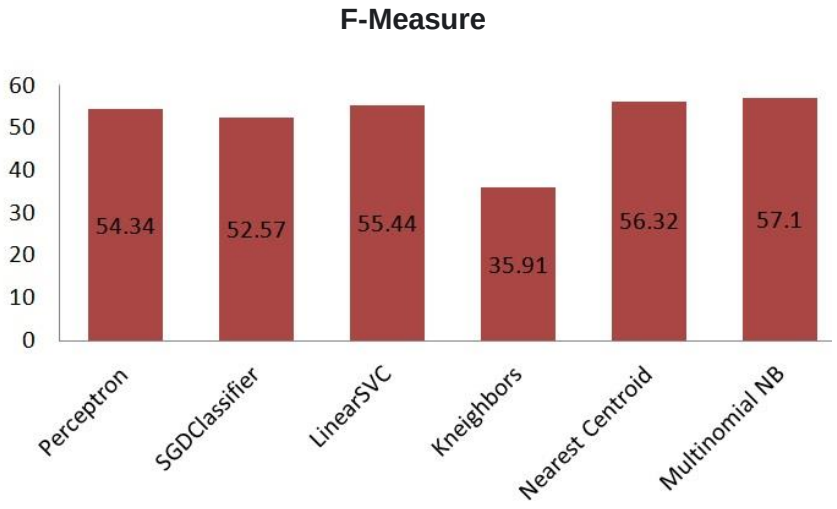


Precision of different classifiers on genre classification



Recall of different classifiers on genre classification

Our goal was to find a classifier that gave high precision at the expense of losing some recall. Hence we didn't rely on F Measure. Our primary candidates were SGDC Classifier, LinearSVC classifier and K Neighbours classifier as they were showing promising results in terms of precision. But their respective recall were comparatively low. Since KNeighbours classifier had the lowest recall, our primary candidates were shortened to SGDClassifier and LinearSVC classifier. We used F Measure to select the best out of the two.

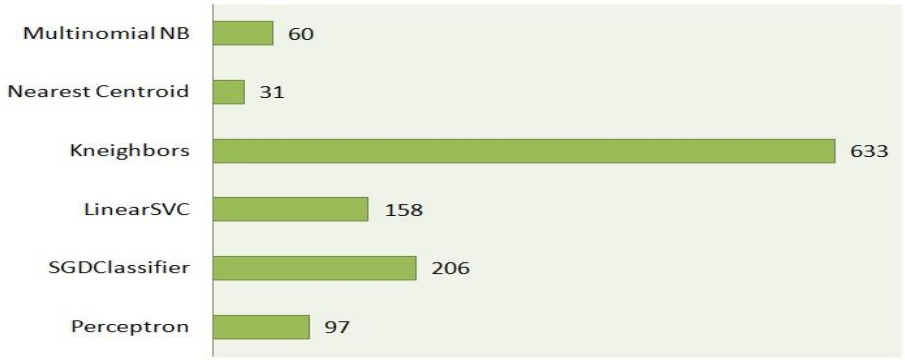


F Measures of different classifiers on genre classification

Linear SVC is observed to have a greater F Measure and hence we selected it as our primary classifier for genre classification.

On sample test run of our model, we noticed that the classifier failed to generate even a single genre for certain movies in the test data. This was because we are using one-vs-rest strategy. If a movie contains elements from a lot of genres, then during classification, each one of the 22 classifiers will always classify it into the 'rest' class. In effect, it will not fall under any of the genres.

Classifier and the number of movies for which no tags were generated



Classifiers long with the number of movies for which no genres were predicted

To overcome this, we made a new model where we predict genres using two classifiers: Linear SVC and Multinomial NB(Since it predicts genres for 97% of movies with 62% precision). When there arise a case when no genres are being generated by LinearSVC(Primary classifier) ,we use the naïve bayes model for prediction (Secondary classifier).

The number of movies with no tags dropped from 158 to 55.
The recall increased from 44.9 to 45.8.
The precision dropped from 72.4 to 71.29.

The F Measure increased from 55.44 to 55.82.

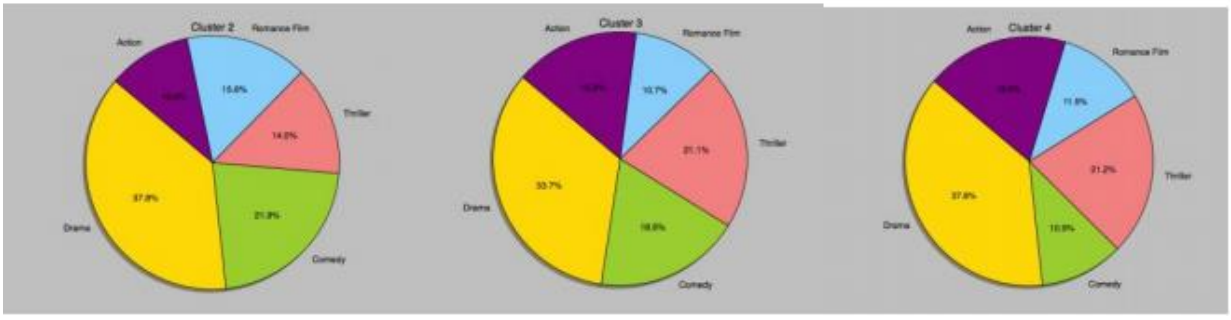
Our training and testing data is only based on Amazon data. So when we predict genres for a movie in test data, we are only comparing it with the genres tagged by Amazon for that particular movie. When we compared the same predictions with genres tagged by IMDB and Wikipedia, there were more number of matches. This implies that the Amazon tagging is incomplete. Hence the above results are only applicable to Amazon data. To know how well the classifier actually performs, we would need to create a new test data where the genre tagging is complete (by scraping genre data from major online movie databases)

Similar Movies

Initially we defined this task in terms of clustering only. We hypothesized that a model that correctly grouped similar movies would also group similar genres. This meant for clustering, we wanted to produce clusters contained movies of similar genre.

We decided we would establish our baseline by observing the relative distribution of the top 5 genres using k-means clustering (k=5, genres = Drama, Comedy, Thriller, Romance, Action). We used the Python Scikit-learn implementation of K-Means clustering to perform our experiment. We defined a somewhat arbitrary 2-dimensional feature vector for each plot summary where the first feature was the number of words in the plot summary and the second feature was the length of the longest word in the summary. The distance metric used by k-means was Euclidean distance.

Baseline Cluster Evaluation

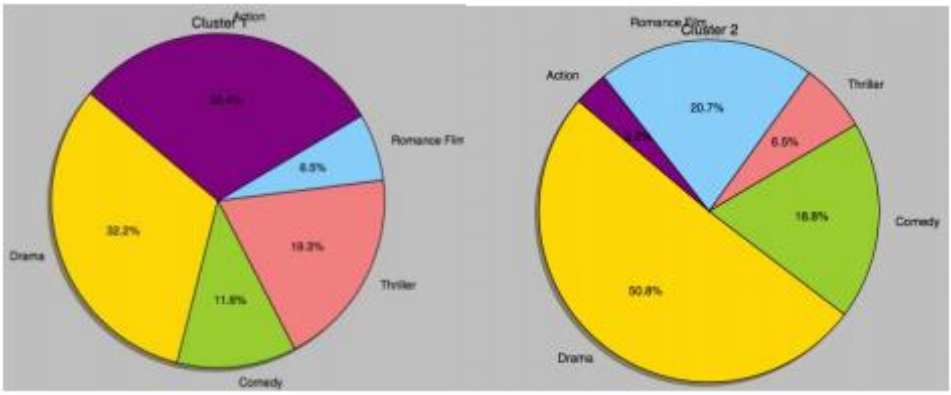


Sample of genre distributions produced during clustering

As shown above, K-Means produced clusters with similar relative genre proportions, which closely resembled the true proportions in the training data.

To improve our results, the first thing we wanted to change was creating a less arbitrary feature vector to represent movie plot summaries. The new feature vectors we used were normalized term frequency vectors (TF-IDF) made from the plot summary data. We got the idea of using TF-IDF vectors from an experimental study of clustering techniques which heavily used k-means.[3] In order to transform our text data into TF-IDF vectors we used the python Scikit-learn 'TfidfVectorizer' before running k-means. With the new feature vectors, we saw a big change in the relative genre distributions of our top 5 genres, but still far from the results we were looking for.

Improved Cluster Evaluation

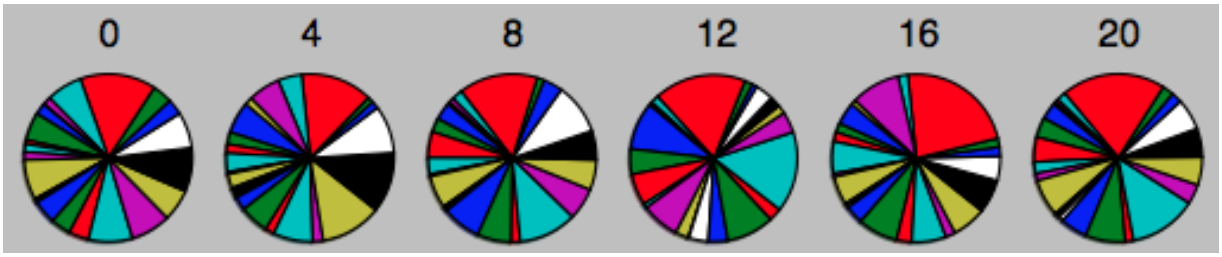


Sample Cluster Genre Distributions Using TF-IDF vectors

Note above the resulting relative distributions of the top 5 genres in the 2nd and 3rd cluster (k=5). As you can see the proportion of Action movies (purple) is drastically different within the clusters, and almost non-existent in the 2nd cluster. This is a large step forward from our baseline to our goal of disjoint clusters of genre.

Next we tried varying the number of clusters produced during k-means, and observed the genre distributions of all 22 genres. We found that even with larger values of k such as 50 and 100 we were not producing clusters with significantly different genre distributions.

Cluster Genre Distribution



Sample Cluster Genre Distributions with k=22

Unsatisfied with our results so far we began experimenting with our feature vectors. Up until this point the TF-IDF vectors were made under a 1-gram model where each term-frequency feature represented 1 word, which clearly does not encapsulate any context of surrounding words. We experimented using 2-gram, 3-gram, and combinations of 1, 2, and 3 gram feature vectors with little positive change in results.

With low success via clustering, we decided to switch gears and try K-Nearest-Neighbors for finding similar movies. The distance metric we used was the cosine similarity of the TF-IDF vectors made from movie plots. For low values of K (1-5) we were starting to see better results.

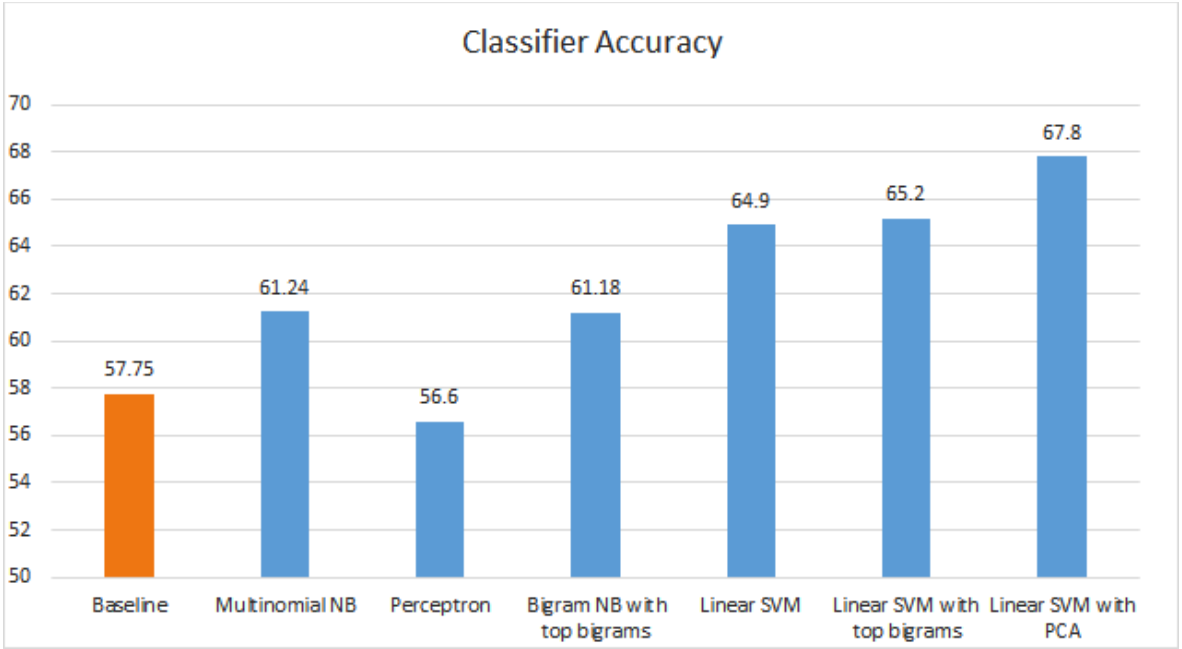
The n-gram model we used for the TF-IDF vectors made a big difference. Using a 1-gram we found that the nearest neighbors of a movie contained the same 1 word in high frequency (as one would expect). When we tried to find similar movies to *Saving Private Ryan*, the 1 gram model returned non-war movies, but all of them had a main character named Ryan. Using a 2-gram model we found the best results on average for returning similar movies. The similar movies returned for *Saving Private Ryan* were all world war 2 movies of the same genre (war-movies). Using a 3 gram model we saw better results than the 1 gram model, but worse results than the 2

gram model. After manually inspecting the returned movies for *Saving Private* Ryan, the dominating genre of the movies were war movies, but not all of them were world war 2 movies.

N-Gram model results for Saving Private Ryan

In the end we accepted KNN as our best model for predicting similar movies, using 2-gram TF-IDF vectors and cosine distance as out metric, with low values for K (20 or less). For some movies a larger value of K still produced movies that seem quite similar, but for movies with a more unique plot (less existing similar movies) low K was better.

Predicting Movie rating



Performance of Different Classifiers on Predicting Ratings based on Reviews

Our naïve bayes model did not perform much better than the baseline. The following is the confusion matrix of our naive bayes model:

	1.0	2.0	3.0	4.0	5.0
1.0	3771	101	847	1140	4153
2.0	829	79	1317	1994	2945
3.0	374	54	1522	5045	5983
4.0	292	53	919	9265	17477
5.0	838	134	743	9366	67045

Confusion Matrix with original class of review on left side and prediction of Multinomial Naïve bayes on the top (for alpha = 1.0)

As we can see the confusion matrix above, the model was classifying the positive reviews correctly but was failing to classify the negative reviews. We thought that considering higher order n-grams might resolve the issue and tried using a bigram Naïve bayes model considering the top 25,000 bigrams (due to the volume of the data, we couldn't afford to consider all bigrams). But even then the confusion matrix of the bigram model exhibited the same phenomenon.

We inferred that this might be happening because most of the reviews had a rating of '5' and due to the naïve bayes models considering the prior probability of a class, the likelihood probability of even negative reviews might have been covered by the prior probability of class '5'. It is clear from the table that most reviews are being classified as '5' irrespective of their original rating.

So we decided to move on to alternative classifiers which did not take the prior probabilities into account. We looked at perceptron algorithm and support vector machines. The results above show that the SVM classifiers have generally fared better than the rest. After removing noise and extracting only the top 10,000 most discriminative features, we find that there is a significant improvement in the performance over the baseline and the naïve bayes models. However, the perceptron algorithm could not learn enough to beat the baseline which came as a surprise. This might be down to it not having good features or due to the high level of noise in the reviews.

	1.0	2.0	3.0	4.0	5.0
1.0	5689	1914	566	229	1614
2.0	1328	1812	1238	992	1794
3.0	628	1182	4756	2636	3776
4.0	332	789	2277	6386	18222
5.0	485	564	808	2413	73856

Confusion Matrix with original class of review on left side and prediction of Linear SVM with PCA on the top

From the confusion matrix of our best performing model, we can see that there is a definite improvement in classification of negative reviews but the bulk of the wrong predictions came from reviews of class '4' being classified as reviews of class '5'. This means that the SVM model was able to predict the tone of the review correctly but failed to capture the magnitude of sentiment. Class '5' still dominates, but the margin of domination has been substantially reduced. It turns out that predicting the magnitude of sentiment is a tougher problem than just saying if the review is 'positive', 'negative' or 'neutral'.

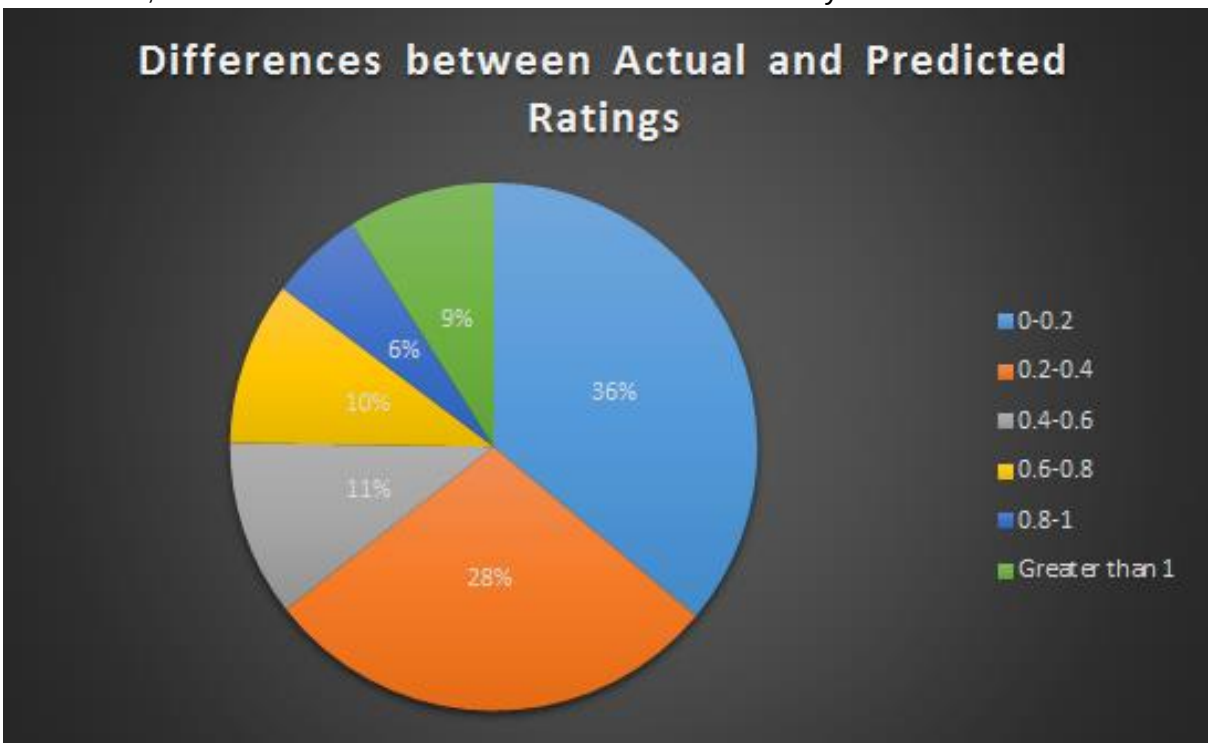
To reinforce our inference and to filter away the possibility that such results might have occurred due to the positive bias in the dataset, we tried to test our model using a toy dataset containing

80,000 training reviews and 10,000 testing reviews evenly split across all ratings (taken randomly).

	1.0	**2.0**	**3.0**	**4.0**	**5.0**
1.0	1814	122	6	7	51
2.0	695	1012	189	22	82
3.0	228	226	857	209	480
4.0	84	180	252	691	793
5.0	3	1	85	164	1747

Confusion Matrix with original class of review on left side and prediction of Linear SVM with PCA on the top on the toy dataset

The results from the above confusion matrix show marked similarities to the one from obtained from the original dataset. So, the problem with the model might not be due to high number of '5' reviews and the performance decreases to 63.2% but this is expected as the baselines is just 20%. Also, we can see a clearer classification of tonality of the review in this case.



Difference between actual movie ratings and predicted ratings of Linear SVM with PCA (on combination of Amazon reviews and Movie plot datasets)

So coming to the crucial part of rating a movie, the rating assigned to a movie was the average of the ratings of all reviews it had. We predicted ratings for all movie reviews in the test dataset using our Linear SVM with PCA model and only considered movies with reviews in the test dataset for this comparison. All reviews in the training dataset were ignored. We can see that the performance of the linear SVM classifier is actually quite good, keeping in mind its shortcomings while classifying single reviews. The classifier was able to predict 36% movie ratings accurately with a difference less than 0.2. 65% of movies had a difference of 0.6 (which can be considered reasonable) between their actual ratings and predicted ratings and 91% movies had a difference

less than 1. So, the severity of the shortcomings decreases and we could say that the model worked reasonably well on the overall task.

Movie Recommendation

After tuning models to tackle the tasks of finding similar movies, genre prediction, and rating prediction, we integrated the three models from each task to recommend movies. Given an input movie the user enjoyed, the recommendation procedure works as follows:

(1) Find the 20 most similar movies to the query movie.

(2) Predict the genres of the 20 movies from (1) and reduce the set of 20 movies to include only movies that share the many genres with the query movie.

(3) Predict the ratings of the remaining movies from (2) and recommend the movie with the highest predicted rating.

Movie Recommendation

```
For query movie:  saving private ryan With genre:  ['drama', 'war film', 'action', 'history']
Recomended movie is:  castle keep
        With predicted genre:  ['action', 'drama', 'war film']
                With predicted rating:  5.0
```

Output of pipeline on sample input movie: 'Saving Private Ryan'

Discussion and Future Work

Movie Rating System: As we can see, the movie rating system has its problems with not being able to predict the magnitude of sentiment in the reviews. It might be down to the irregularities with general user behavior where the user says everything is good but still gives a harsh rating to the movie. But we could explore Pointwise Mutual Exclusion (PMI) based method and compare our review words to very positive words like "excellent" and very negative words like "awful" and use the scores to assign ratings but this would only work if the PMI scores can indeed capture the little nuances in sentiment of the review. We could also try POS tagging the words and trying to correlate the number of adjectives and weights based on PMI scores to try to model the magnitude.

Copyright System: By giving more emphasis on the second part (Finding Similar Movies With Plot Summaries) film directors, producers and writers can research whether their plot for a new movie had already been made into a movie or not. This can help resolve a lot of copyright issues. This kind of a system could also help them improve their plots based on user reviews of similar movies (based on what users liked and what they did not). They could see how such movies fared at the box office and how the cast was chosen for similar plots which can help them with crucial decisions affecting the quality of the movie.

Predicting Box office success: Our system can be used for predicting the box office gross of a movie. Performance of a movie at the box office depends on a lot features that include the cast, director, date of release, genre, synopsis etc. But initially, when the director only has a plot to

start with, he could use our system to find similar movies. And if we have the movie box office collections data, the we could extend our system to find the average of their collections and get a rough estimate of how the movie is going to fare at the box office.

Novel recommendation system: A similar approach can be followed to develop a system that can recommend people what to read next. All we need to do is to edit the list of genres to suit novels instead of movies.

References:

1. Learning Latent Personas of Film Characters David Bamman, Brendan O'Connor, and Noah A. Smith ACL 2013, Sofia, Bulgaria, August 2013.
2. J. McAuley and J. Leskovec. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. WWW, 2013.
3. Michael Steinbach, George Karypis, Vipin Kumar. *"A Comparison of Document Clustering Techniques"*
4. Soo-Min Kim, Eduard Hovy. Determining the Sentiment of Opinions. COLING '04 Proceedings of the 20th international conference on Computational Linguistics