Final Project Report for CS 175, Winter 2018

Project Title: Box Office Prediction based on Initial Social Media Reaction
Project Number: 21

Student Name(s)
Kevin Yin, 29211757, yink3@uci.edu
Jennifer Jisoo Kim, 72238150, jisook5@uci.edu
Justin Fu, 80167602, fujs@uci.edu

1. Introduction and Problem Statement (1 or 2 paragraphs)
The goal of this project is to predict the box office revenue of American movies released from 2006 - 2017 based on initial social media language and movie metadata such as genre, budget, director, and lead actor net worth. As for the social media, we will use content from Reddit, IMDB user reviews.

2. Related Work: (1 or 2 paragraphs)
For our text analysis on IMDB movie reviews and Reddit comments, we used both a bag-of-words and a dictionary of unique movie ids associated with a string that was the concatenation of all its reviews or comments. This approach was selected as a way to mine text data as suggested by *Aggarwal and Zhai (2012)* . We referenced multiple abstracts found online that used text data to predict box office revenue in order to help us generate our metrics for prediction; some metrics we took were beyond the text such as volume of social media presence (*Early Prediction of Movie Box Office Success*, Box office prediction based on microblog).

3. Data Sets [at least 1 page]
IMDB review data

IMDB provides a public reposity of all movies, TV shows, actresses, actors, directors and TV episodes in different TSV files. The one we used had movie and TV show episde metadata with ~7 million rows of data. We used the TSV file to gather the IMDB ID (tconst), movie title and release year into our own CSV file called tconst_title_year.csv. This gave us an initial ~700,000 movies.
Source: *https://datasets.imdbws.com*

Then, we used the OMDB API to gather the genre, director, main actor, and box office revenue for each movie. For a movie to be included into our dataset, it had to contain a box office revenue. This resulted in 5,000 usable movies.
Source: *http://www.omdbapi.com/*

In order to get the budget for these movies, we used a combination of IMDB scraping and TMDB API to populate our data.
Python library: *https://pypi.python.org/pypi/tmdbsimple*
For the API key: *https://www.themoviedb.org/?language=en*

To get the reviews for each movie, we had to write a python script that dynamically viewed each movie's review pages until all pages were exhausted.
Example user review page for a movie that we scraped:
http://www.imdb.com/title/tt0468569/reviews?ref_=tt_ql_3

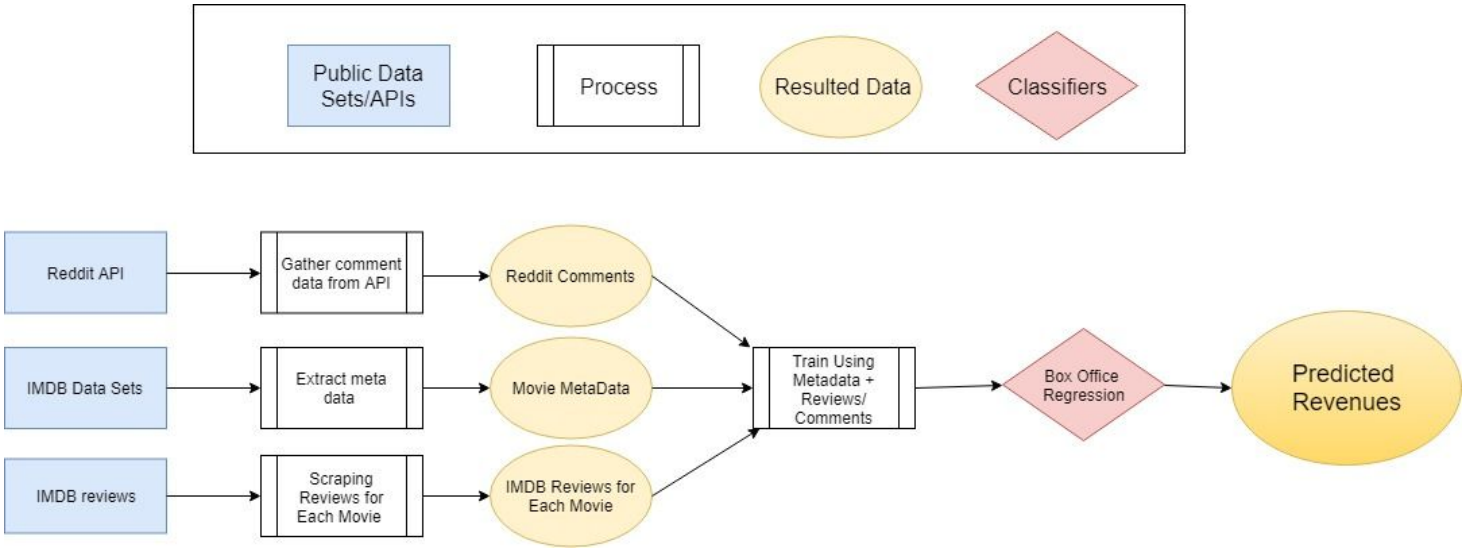
We used an online corpus of polarizing emotion for our learner.
Source: *https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html#lexicon*

Data Pipeline:

Initial CSV from IMDB (including actors, tv shows, etc)	**Movies from IMDB, after 2006**	**Movies with a box office obtainable from OMDB**	**Movies with reviews within the first 3 weeks**
~7 million rows	~700,000	~5,000	~4,000

Reddit comments
Scraped a total of 11 million comments from Reddit **r/Movies** and **r/askreddit**. Simplified it down to 6.5 million comment that had a movie title in the body of the text. From here, we had 3619 unique movie, from which 13 were excluded because they appeared in over 100,000 comments each. The average movie title was mentioned in 1,113 different comments. Movie titles with only 10 or few comments were kept because their often unique titles meant the comment was significantly more relevant than those will many comments. A final version of 3 million comments split into 30 subsets of randomly selected and unrepeated comments.

4. Description of Technical Approach [at least 1 page]



The first steps in our project was gathering our data. Originally, we wanted to use Twitter, but historical data is difficult to access online without paying ridiculous amounts of money. So, we pivoted to utilizing IMDB user reviews, and supplementing that with user comments from places like Reddit.

First, we figured out that there is no dataset online with IMDB review data, that has metadata like date and corresponding movie, and wasn't purely made for sentiment analysis. So, we needed to scrape the data. We first found a dataset online from IMDB that gave us a list of all their data, about 7 million rows in the form of a CSV, which we filtered down to about 700,000 movies after 2006. From these, we ran a python script checking each imdb id with the OMDB API and recorded all the movies that had a box office value, which we decided was the minimum criteria for a movie eligible for our project. We ended up with only ~5,000 movies.

Next, we created our own python scraper with BeautifulSoup4 that searched for reviews and budgets for a movie using its IMDB id. Since IMDB altered their page structure to only display additional movie reviews dynamically through user input, we figured out we could "load" more reviews through continuously simulated Ajax calls on each page. We also scraped the main movie landing page for the budget value.

We wanted our dataset to only contain movies with at least 1 review so this reduced our final dataset to 3747 movies.

To begin, we decided to split up the learning process into two separate categories: individual movie feature learners and combined learners. For each feature, we used two regression models: linear regression and random forests ensemble. We created learners for attributes like budget, genre, review count, review score average, and review text, then analyzed their results to decide which features would be included in our multiple regression learners. We also used a

10 fold cross validation to split our data into training/testing sets because our dataset was smaller than we can initially anticipated.

Because our review count, budget and review text learners did the best, we decided to use those in our multiple regression learner.

For the metadata, we tried a couple of different regression algorithms, including linear regression, cross validation and random forest algorithms. We also tried a multilinear regression involving budget and review count. For our text analysis learners, we utilized linear regression and random forests with both Tfidf and Count vectors.

The reddit data was the most difficult to collect and organize. Premature crashing forced us to restart the data collection since there was no way to identify repeated comments without exponentially increasing the time of collection. A lot of data was sacrificed (not collected) in order to have some set of workable data in time for presentations. After collection, the comments were given an unique ID and associated with a unique IMDB Id in a many-to-one relationship which increased our database size to 5 gb because of repeated comments. 13 movies were excluded because they included 100,000 or more comments suggesting that their title was too ambiguous for our filter to detect. A total of 30 sets of 100,000 comments was finalized and used to train and test a linear regression model. Through time constraints, this model is seperate from the movie metadata model and the IMDB reviews model. Having many different sets, with various amount of unique movies allowed for an interesting range of prediction accuracies.

4. Software

We wrote:

Regression Learners	*Regression_learners.ipynb* Contains all the code we wrote for create, train and test all of our learners.
Comment Scraper	*Reddit_scraper2018.py* and *RedditAPI_27.py* were scripts we wrote that used PRAW to retrieve comments.
Reddit Database	*Reddit_scraper2018.py*, *sql3_test.py, sort_comments.py, sort_comments2.py* and *random_sets.py* are all programs we wrote used to store filtered comments into a SQLite3 database.
IMDB meta-data organizer	Downloaded a .csv file from IMDB and cross referenced it with OMDB and TMDB to get our final result of movie IMDB ID's and

	budget. fetch_imdb_metadata.py fetch_reviews.py

We used:

SciKit Learn	Scikit learn provided us with the basic structure of our Learnings: Linear Regression and Random Forest Ensemble.
NumPy	Gave us the tools to manipulate and translate our data into matrices that could be inserted into the learners. Also allowed for us to calculate our results and graph them.

5. Experiments and Evaluation [at least 1 page, preferably 2 or 3]

Movie Metadata Count Statistics

First, we ran some tests on how many reviews we were able to scrape for each movie. Here are some statistics:

Number of movies with more more than 0 reviews out of 5000: 4555
Mean: 158.514
Median: 57
Standard Deviation: 293.35
Minimum = 1
Maximum = 5147
Number of movies w/ more than 150 reviews: 1316
Number of movies w/ more than 20 reviews: 3187
Number of movies w/ at least 1 review: 3747
Movies with at least 1 review, budget and box office: 3747

Single Feature Regression Learners

For each individual movie metadata feature, we created respective lists to store all the features in the same movie order so that when they were transformed, they would all maintain the same order.

Variables with large values such as budget and box office revenue were normalized by taking the log 10 of each value.

Once we had the data ready, we ran each feature with linear regression and random forests.

Variance of Features

	Genre	Budget	Review Score Avg	Review Counts
Linear Regression	2%	62%	0.4%	18%
Random Forest	18%	63%	15%	59%

The review counts and budget were our strongest individual features which is why we decided to use these in our multiple regression learners.

IMDB Text Analysis Learners

We wanted to see how many reviews each movie had, and see if it had any effect on the data. We ran a few tests and decided that although our project is about initial movie data (the first three weeks), it wouldn't hurt to try and test it with movie review data from throughout history. In this way, we would instead be looking at whether or not the overall language used in the reviews in discussion had an effect on box office, and not just initial social media reaction.
All tests are conducted with encoding normalization. There is minimal difference with choosing between 1, 2, and 5 minimum document frequency.

After attempting SVN, linear regression, decision tree, and random forests, we chose linear and random forests as the learners we wanted to use. Linear regression was an easy to understand algorithm that we could look closely and understand its data more easily, while an ensemble learner like random forest would provide us with the best possible variance score.

Throughout our data analysis, the random forests almost always ended up with better results, so we will be referencing that score when we talk about variance. First, we tried by running with all the reviews within the first 3 weeks, and resulted in about an 70% variance. Next, I tried with ALL the reviews from all time, which resulted in an almost 30% decrease. Then, limiting it to just movies with over 20 reviews. We concluded that running our learner with reviews from the first 3 weeks or all movies with reviews over 20 resulted in a similar 70% variance. This means that by including ALL of the movies and their reviews, we were including a lot of noise, and perhaps a lot of skewed predictions from movies with only a couple of reviews. So, from all testing from here on forward, we used all the movies and their reviews from the first 3 weeks.

Below are a number of tests we conducted. I tried a number of different parameters. First, I ran the following tests with the tfidf vector, with just removed stop words. I got a decent value, around 73%. I then tested with a range of different vectorization parameters, with their results listed below.

After our results from the initial metadata learners, we figured out that review score is a poor indicator of box office success, meaning that the user's opinionated analysis is potentially

unuseful. From this, we thought that maybe, if we remove all of the polarizing words, we could clean up some of the noise and result in a better learner. I found an online corpus of polarizing positive and negative words, totaling about 6,000. I added these as additional “stopwords”, excluding them from our Count Vectorizer.

Linear Regression

	With All Reviews	All movies, 20+ Reviews	First 3 weeks:	With 2-Grams	With stopwords	No polarizing words	With lemmatization	Default (No stopwords)
MSE	881563.407	319812.7161		496327.4122	545704	612784.36	556632.252	612784.36
Variance	43%	55%		58%	52%	49%	52%	52%

Random Forest

	With All Reviews	All movies, 20+ Reviews	First 3 weeks:	With 2-Grams	With stopwords	No polarizing words	With lemmatization	Default (No stopwords)
MSE	582865.804	319812.716		219562.882	326068.4	359890.58	401799.25	359890.58
Variance	62%	70%		71%	72%	70%	65%	73%

In conclusion, many of the strategies we tried did not improve our learners by that much. It was a mostly minor or marginal increase, across the board.

Reddit Comment Text Analysis

Linear Regression Average

	Vector Counts	**TF-IDF**
MSE	550.0338	3.1637
Variance	-430.611	-0.565

Random Forest Average

	Vector Counts	**TF-IDF**
MSE	2.009	2.677
Variance	0.005	0.025

To avoid overfitting ad to try cross-validation, I split the database of 11 million comments into multiple sets of 100,000 comments. Each set averages around 2000 unique movie titles with an average of 970 comments per movie. I ran the Vector Counts, TF-IDF on each subset. From the last previous presentation, by taking the average of multiple smaller sets, the TF-IDF did not do any better or worst than the original set we ran it on. The biggest mistake was in the early scraping of comments when the decision was made to sacrifice context for quantity of data. The current set is too noisy to extract any usable information.

Multiple Regression Learners

We performed 2 multiple regressions:

For our first regression, we used the budget and review counts as the independent variables to predict box office revenue. To do this, we created a new test data set where each value was a list of the budget and review count respectively. We fed this new matrix into both linear regression and random forest learners.

For our second regression, we combined the TFIDF bag-of-words vector with review count and budget. To create the a dataset for this, we appended the normalized values for review count and budget to the TFIDF matrix. The values were normalized so that they would uphold similarity to the TFIDF values. We fed this into our linear regression and random forest learners in two forms -- one with all the words (except stop words) and only including words from the top 50 TFIDF value list.

Variance

	Budget & Count	**Budget, Count & TFIDF**	**Budget, Count & TFIDF (top 50)**
Linear Regression	60%	50%	22%
Random Forest	70%	66%	53%

6. Discussion and Conclusion [at least ½ a page]

This project gave us a lot of insight and hand-on knowledge on how powerful data can be if processed and used utilized with the right knowledge.
Our goal was to be able to predict the box office revenue of a movie based on movie data that could essentially be accessed by anyone on the internet. We were able to achieve a maximum of 70% accuracy with our budget & count multiple random forest regression learner which is roughly what we expected, though we were optimistically aiming for an 80% accuracy.

A major limitation we did not expect was how noisy our data turned out to be, especially with our Reddit comments. Though it initially seemed like a bountiful resource of initial human reactions to movies, the Reddit data proved extremely difficult to clean. Many movie titles were common colloquial sayings which skewed the accuracy of our data resulting in us losing a great chunk of comment we were relying on.

Looking at the results, what was surprising was how poorly the user score average performed in its individual feature learner, coming in at a 15% accuracy at best with random forests. This goes to show that what the judgement of an average user does not correlate to a movie's box office success.

Overall, our learners turned out to perform with a decent accuracy and if we were able to spend more time fine tuning our parameters we would have tried stemming all our words for the bag of words countvectorizer. Nonetheless, we are satisfied with the outcome and with the learning experience we gained from this project.