Movie Plot Based Recommender System

Group 7: Eyoel Armede, Andrew Cummings, Brad Fetes, Ian Ustanik

IST 664 Final Project Report

Project Rationale

Traditional recommender systems use a mix of ratings systems and clustering based on movies watched and similar movie genres or length. In this system, if a person enjoys a movie in the action or horror genres, they would be recommended another movie within those rigid definitions. Oftentimes, movie watchers are not satisfied with suggestions that rely on these parameters. Movie watchers not only want to be recommended things familiar to them but often want films that feel new. Our solution to this need is to create an addition to these recommender systems that uses NLP techniques to match users with relevant movies based on processing the plots of movies they have seen. We believe that recommending movies based on the content of a film's plot will not only recommend movies that are similar to a user's likes and interests (what current recommenders do) but also expand their interest across genres and types of movies they may typically watch.

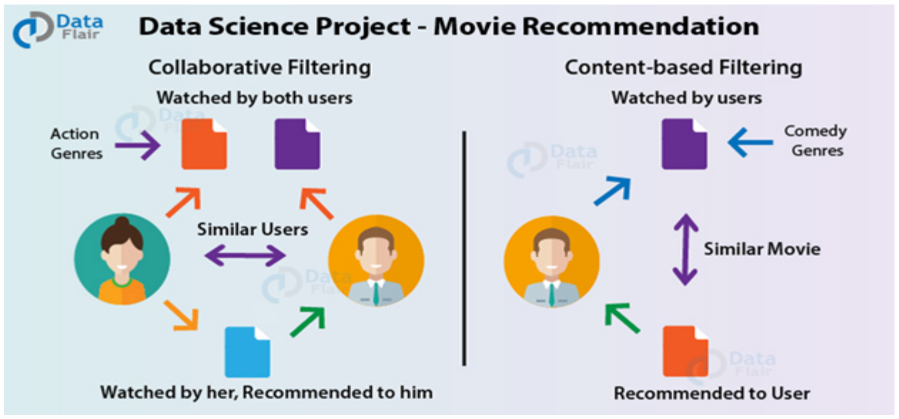


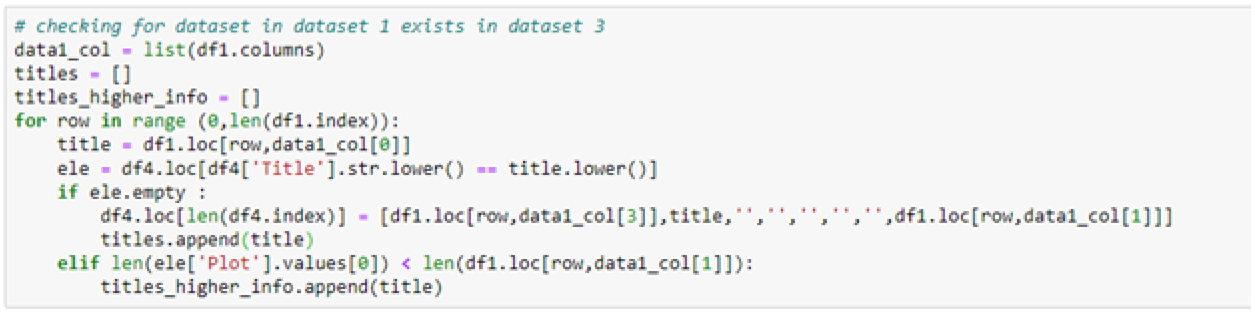
Figure 1: Example of how traditional movie recommendation systems work

Dataset Description

The datasets that were used for this analysis were from Kaggle. The information in these datasets came from one of two sources, IMDB or Wikipedia. Each dataset included the title of the film, release year, plot summary, and a genre categorization. Two of the datasets added additional elements such as name of the director, prominent actors/actresses, Wikipedia page, and origin of the film. Combined, the filtered dataset contained around 90,000+ movie records. The team decided to only analyze movies that were released after 1980 to add a recency aspect for the analysis. Additionally, only movies that had an origin of America or Britain were used to return mainly English-speaking plots for a more consistent analysis.

Pre Processing

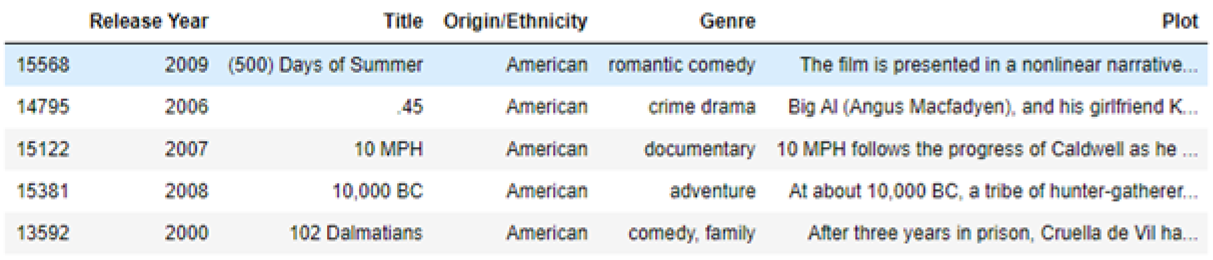
The dataset that is used for the project is a combination of 3 different datasets from kaggle. The datasets are diverse in that it doesn't always contain the same column information. The common column information is movie title and plot summary. The data has been filtered and merged together as a single dataset. The datasets combined together have features such as Movie title, release year, Origin, Genre and Plot. The datasets were combined in such a way that if the movie is available across all the data sets then the entry with the longest plot summary is taken as an entry. The movie is also filtered based on origin of the movies as the final datasets are American and British in origin. The code in Figure 2 below shows checking between two datasets that have the same movie title and release year and take the data entry with the longest. Summary plot.



```
# checking for dataset in dataset 1 exists in dataset 3
data1_col = list(df1.columns)
titles = []
titles_higher_info = []
for row in range (0,len(df1.index)):
    title = df1.loc[row,data1_col[0]]
    ele = df4.loc[df4['Title'].str.lower() == title.lower()]
    if ele.empty :
        df4.loc[len(df4.index)] = [df1.loc[row,data1_col[3]],title,'','','','','',df1.loc[row,data1_col[1]]]
        titles.append(title)
    elif len(ele['Plot'].values[0]) < len(df1.loc[row,data1_col[1]]):
        titles_higher_info.append(title)
```

Figure 2: The code to merge multiple datasets to one

The figure below shows the final merged dataset from the 3 original datasets with release year, title, origin, genre and plot summary fields.



	Release Year	Title	Origin/Ethnicity	Genre	Plot
15568	2009	(500) Days of Summer	American	romantic comedy	The film is presented in a nonlinear narrative...
14795	2006	.45	American	crime drama	Big Al (Angus Macfadyen), and his girlfriend K...
15122	2007	10 MPH	American	documentary	10 MPH follows the progress of Caldwell as he ...
15381	2008	10,000 BC	American	adventure	At about 10,000 BC, a tribe of hunter-gatherer...
13592	2000	102 Dalmatians	American	comedy, family	After three years in prison, Cruella de Vil ha...

Figure 3: Final merged dataset

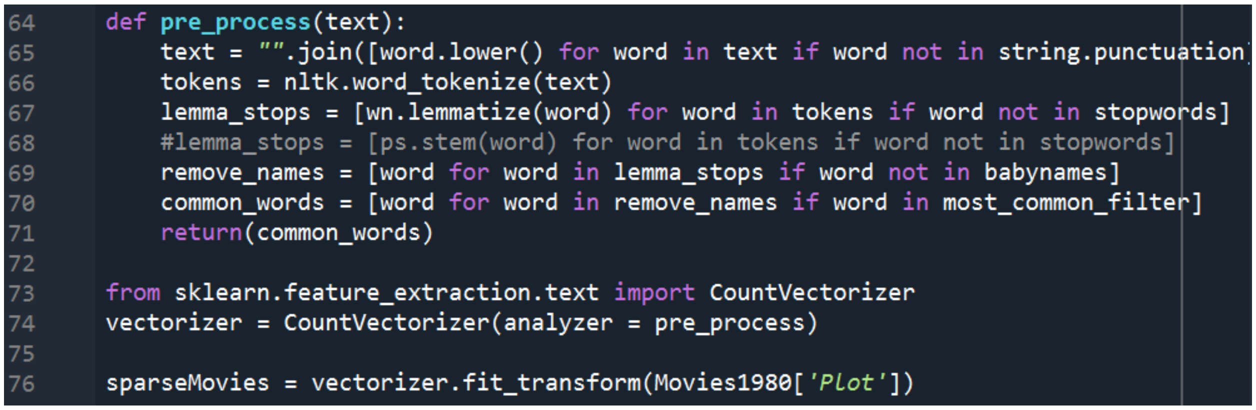
Feature Selection

The feature selected for processing is the plot summary text only. The project wants to recommend a system based on plot summary because of that other features weren't used as features. To extract features first the plot summary has been tokenized and then the stop words are removed from the data sets. Further peoples name has been removed from the text before finding the most common words in the datasets.

The other features are obtained by using the pos tagger and tagging each statement to find the adjectives, adverbs and verbs in all the text. After compiling this the most common adjective, adverbs and verbs were used as feature sets. Finally by using the VADER sentiment analysis the average sentiments of each plot summary has been calculated and the four values positive, negative and neutral and compound have been used as feature sets.

Model 1

One option for a recommendation engine based on movie plots would be to create features based on the most common words in the corpus. Our corpus contained over 60,000 words and this model utilized the top 2% of words (approx 1200 words) in the corpus to look for relationships between movie plots. To process the data we created a function that pre-processed the data to word-lemma form. Additionally, this function removed stopwords, person/character names, punctuation, and limited the data to the most common words based on the entire corpus. This function was then passed to SK-learn's ***countvectorizer***. Countvectorizer creates a sparse matrix of tokens and counts of tokens within each movie plot. The countvectorizer also weights the frequency of each word within the plot of a film. If a film had the term "cars" or "death" multiple times, those terms would be weighted more heavily and ideally paired with other movies in which these terms also more frequently appear.



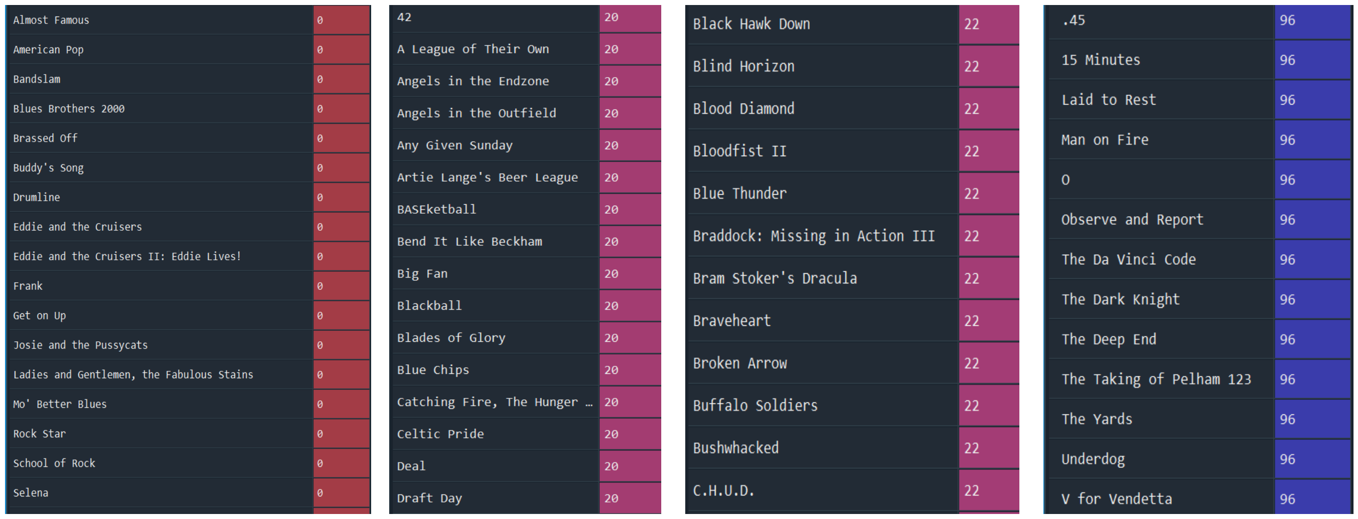
```python
def pre_process(text):
    text = "".join([word.lower() for word in text if word not in string.punctuation
    tokens = nltk.word_tokenize(text)
    lemma_stops = [wn.lemmatize(word) for word in tokens if word not in stopwords]
    #lemma_stops = [ps.stem(word) for word in tokens if word not in stopwords]
    remove_names = [word for word in lemma_stops if word not in babynames]
    common_words = [word for word in remove_names if word in most_common_filter]
    return(common_words)

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(analyzer = pre_process)

sparseMovies = vectorizer.fit_transform(Movies1980['Plot'])
```

Figure 4: The code to to clean and vectorize the data

After passing the sparse matrix to a Kmeans analysis of 100 clusters we looked at the movies in each cluster to see how accurate the cluster was. We found that our model produced a number of useful clusters. These clusters would not only recommend movies within a viewer's "comfort range" but also expand their horizons into different styles and genres of what they are traditionally interested in.



Almost Famous	0
American Pop	0
Bandslam	0
Blues Brothers 2000	0
Brassed Off	0
Buddy's Song	0
Drumline	0
Eddie and the Cruisers	0
Eddie and the Cruisers II: Eddie Lives!	0
Frank	0
Get on Up	0
Josie and the Pussycats	0
Ladies and Gentlemen, the Fabulous Stains	0
Mo' Better Blues	0
Rock Star	0
School of Rock	0
Selena	0

42	20
A League of Their Own	20
Angels in the Endzone	20
Angels in the Outfield	20
Any Given Sunday	20
Artie Lange's Beer League	20
BASEketball	20
Bend It Like Beckham	20
Big Fan	20
Blackball	20
Blades of Glory	20
Blue Chips	20
Catching Fire, The Hunger …	20
Celtic Pride	20
Deal	20
Draft Day	20

Black Hawk Down	22
Blind Horizon	22
Blood Diamond	22
Bloodfist II	22
Blue Thunder	22
Braddock: Missing in Action III	22
Bram Stoker's Dracula	22
Braveheart	22
Broken Arrow	22
Buffalo Soldiers	22
Bushwhacked	22
C.H.U.D.	22

.45	96
15 Minutes	96
Laid to Rest	96
Man on Fire	96
O	96
Observe and Report	96
The Da Vinci Code	96
The Dark Knight	96
The Deep End	96
The Taking of Pelham 123	96
The Yards	96
Underdog	96
V for Vendetta	96

Figure 5: Various clustering examples

As successful as this method was there are a few downsides to this potential model. First, by basing the features on the most popular words in the whole corpus, movies with unique plots may be left empty or without any defining features. Secondly and relatedly, picking the appropriate number of words to base features on is important. If too few word features are picked, distinctions between movies will be difficult but having too many words as features may render some clusters with only one or two films. Lastly, the difference in the lengths of the movie's plots is going to be a concern using this method. For example, Legally Blonde has 147 terms in which to cluster around, while the 1987 film Cyclone only has seven. The most common words model provided our group with some success but realizing that it had numerous shortfalls we decided to try other methods to see if we could more accurately group movies based on their plots.

<u>Model 2 (TF-IDF)</u>

For the second model, the team decided to use TF-IDF (Term Frequency – Inverse Document Frequency) as a method of feature engineering for creating distinct movie recommendation clusters. TF-IDF produces a sparse matrix made up of vectors where words common to all documents get discounted by the inverse of their frequency. See the below formula where f_i is the number of documents that contain the word i:

$$\text{tf-idf}_{ij} = f_{ij} \log \frac{|D| + 1}{f_i + 1}$$

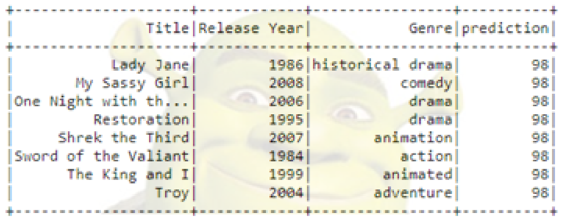
Figure 6: TF-IDF formula

In addition to the sparse matrix output from the TF-IDF method, the team also chose to consider sentiment for each movie plot before making a cluster prediction. For this task, NLTK’s VADER package was used, where an average was taken across all sentence compound sentiment scores in a movie plot.

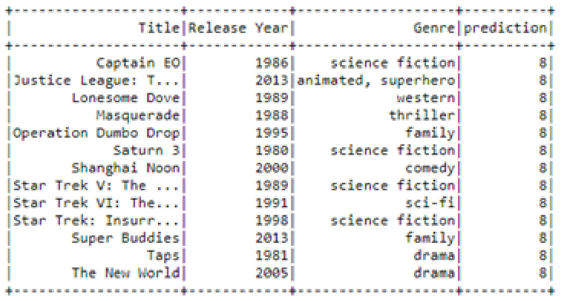
Due to the size of the movie plots dataset, the team decided to take advantage of PySpark, which is the python release of Apache Spark, used for distributed datasets in computing. The team utilized what is known as a PySpark machine learning pipeline to produce clusters from the TF-IDF and sentiment analysis. The pipeline consisted of seven different functions that would take some sort of input and produce an output that was usually used as the input for the next step of the pipeline. Tokenizer() was the function used to tokenize the text in the movie plot. It received ‘Plot’ as input and produced a column ‘words’ as output. StopWordsRemover() used a University of Glasgow stopwords corpus to remove stopwords from the text. It received ‘words’ as input and returned the column ‘filtered’ as output. CountVectorizer() produced a word count vector, using all words in the movie plots corpus. It took ‘filtered’ as the input column and produced ‘tf’ as the output. This is where the term frequency aspect of TF-IDF comes into play.

IDF() computed the resulting sparse matrix using the TF-IDF formula to account for inverse document frequency. It received ‘tf’ as input and returned an output column of ‘tfidf’, using minimum document frequency set to 100 as a parameter. VectorAssembler() takes all of the features needed to be input into a machine learning algorithm and combines them into a single feature vector. This is necessary for PySpark logic and it took ‘tfidf’ and ‘sentiment’ (derived from the use of NLTK’s VADER package) as the input columns and produced a single output column of ‘features0’. StandardScaler() was then used to scale all input features so that they could be compared on the same level. It took ‘features0’ as an input and returned ‘features’ as the output column.

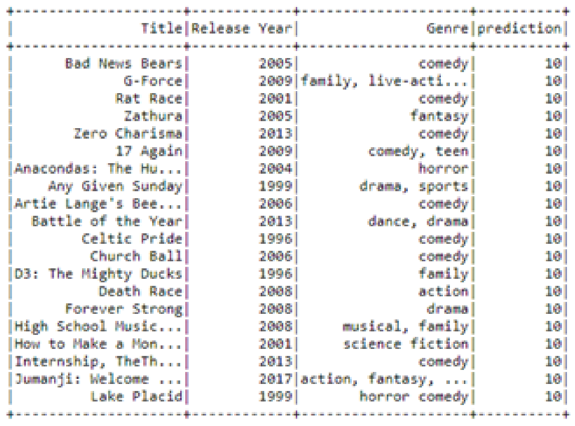
Finally, Kmeans() could now be used to produce distinct movie recommendation clusters. Kmeans is the PySpark formula for the k-means clustering algorithm. K was set to 100 and the ‘features’ column was used as input for creating the cluster predictions. The team returned a mix of a few very large clusters where there was too little plot data and therefore insufficient distinction amongst individual movies, in addition to a few very small clusters that appeared to be outliers. Around forty very strong candidates for good clusters were generated which produced interesting insights and results that could be used to make a movie recommendation:



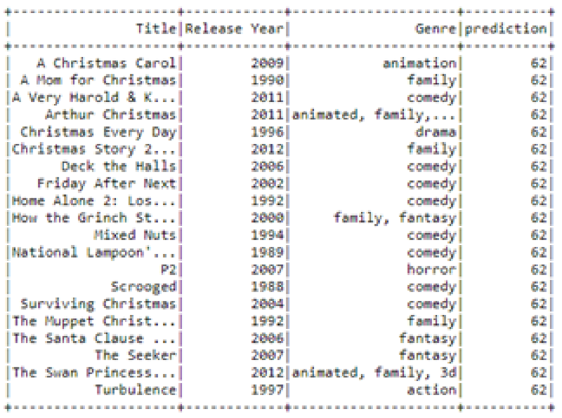
```
+--------------------+------------+----------------+----------+
|               Title|Release Year|           Genre|prediction|
+--------------------+------------+----------------+----------+
|          Lady Jane|        1986|historical drama|        98|
|       My Sassy Girl|        2008|          comedy|        98|
|One Night with th...|        2006|           drama|        98|
|         Restoration|        1995|           drama|        98|
|     Shrek the Third|        2007|       animation|        98|
|Sword of the Valiant|        1984|          action|        98|
|      The King and I|        1999|        animated|        98|
|                Troy|        2004|       adventure|        98|
+--------------------+------------+----------------+----------+
```



```
+--------------------+------------+-------------------+----------+
|               Title|Release Year|              Genre|prediction|
+--------------------+------------+-------------------+----------+
|          Captain EO|        1986|    science fiction|         8|
|Justice League: T...|        2013|animated, superhero|         8|
|       Lonesome Dove|        1989|            western|         8|
|          Masquerade|        1988|           thriller|         8|
|Operation Dumbo Drop|        1995|             family|         8|
|            Saturn 3|        1980|    science fiction|         8|
|       Shanghai Noon|        2000|             comedy|         8|
|Star Trek V: The ...|        1989|    science fiction|         8|
|Star Trek VI: The...|        1991|             sci-fi|         8|
|Star Trek: Insurr...|        1998|    science fiction|         8|
|       Super Buddies|        2013|             family|         8|
|                Taps|        1981|              drama|         8|
|       The New World|        2005|              drama|         8|
+--------------------+------------+-------------------+----------+
```



```
+--------------------+------------+--------------------+----------+
|               Title|Release Year|               Genre|prediction|
+--------------------+------------+--------------------+----------+
|      Bad News Bears|        2005|              comedy|        10|
|             G-Force|        2009|family, live-acti...|        10|
|            Rat Race|        2001|              comedy|        10|
|             Zathura|        2005|             fantasy|        10|
|       Zero Charisma|        2013|              comedy|        10|
|            17 Again|        2009|        comedy, teen|        10|
|Anacondas: The Hu...|        2004|              horror|        10|
|    Any Given Sunday|        1999|       drama, sports|        10|
|Artie Lange's Bee...|        2006|              comedy|        10|
|  Battle of the Year|        2013|        dance, drama|        10|
|         Celtic Pride|        1996|              comedy|        10|
|          Church Ball|        2006|              comedy|        10|
|D3: The Mighty Ducks|        1996|              family|        10|
|          Death Race|        2008|              action|        10|
|      Forever Strong|        2008|               drama|        10|
|High School Music...|        2008|     musical, family|        10|
|How to Make a Mon...|        2001|     science fiction|        10|
|Internship, TheTh...|        2013|              comedy|        10|
|Jumanji: Welcome ...|        2017|action, fantasy, ...|        10|
|         Lake Placid|        1999|       horror comedy|        10|
+--------------------+------------+--------------------+----------+
```



```
+--------------------+------------+--------------------+----------+
|               Title|Release Year|               Genre|prediction|
+--------------------+------------+--------------------+----------+
|   A Christmas Carol|        2009|           animation|        62|
|  A Mom for Christmas|        1990|              family|        62|
|A Very Harold & K...|        2011|              comedy|        62|
|    Arthur Christmas|        2011|animated, family,...|        62|
| Christmas Every Day|        1996|               drama|        62|
|Christmas Story 2...|        2012|              family|        62|
|      Deck the Halls|        2006|              comedy|        62|
|   Friday After Next|        2002|              comedy|        62|
|Home Alone 2: Los...|        1992|              comedy|        62|
|How the Grinch St...|        2000|     family, fantasy|        62|
|          Mixed Nuts|        1994|              comedy|        62|
|National Lampoon'...|        1989|              comedy|        62|
|                  P2|        2007|              horror|        62|
|            Scrooged|        1988|              comedy|        62|
| Surviving Christmas|        2004|              comedy|        62|
|The Muppet Christ...|        1992|              family|        62|
|The Santa Clause ...|        2006|             fantasy|        62|
|          The Seeker|        2007|             fantasy|        62|
|The Swan Princess...|        2012|animated, family, 3d|        62|
|          Turbulence|        1997|              action|        62|
+--------------------+------------+--------------------+----------+
```

Figure 7: Various TF-IDF clustering examples

<u>Model 3 (word2vec Recommender System)</u>

For the third and final model, the team used a word2vec pipeline that embedded the filtered words from each movie and generated a three dimensional vector for each movie as a result, which was then used in k-means clustering. The pipeline consisted of a tokenizer, stop word remover, the word2vec main model, a vector assembler, and a k-means model. Similar to the second model, this process was done exclusively in Pyspark, with data being visualized using the pandas library.

This model was used because of its frequency in commercial recommender systems such as in Netflix or Spotify, and is usually trained using various Neural Networks. Behind the scenes, the word2vec method relies on two nlp models; the skip-gram and the continuous bag of words(CBOW) model. The main difference between these two methods is that the skip-gram assumes that a word will be generated based on the context words surrounding it, with CBOW being the opposite; the context words will be generated based on the current word. Below is a look at computing the word vectors to find the word love using CBOW.

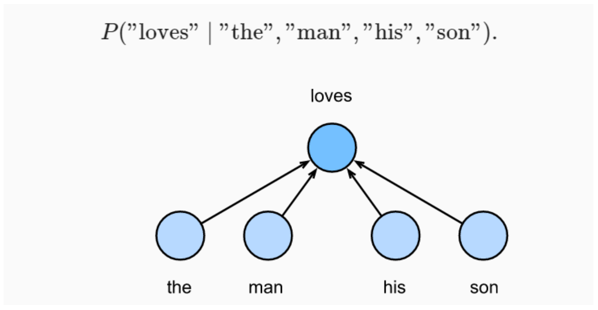

$P(\text{"loves"} \mid \text{"the"}, \text{"man"}, \text{"his"}, \text{"son"}).$
loves
the
man
his
son

Figure 8: CBOW model

Overall, this model was found useful because of its high interpretability; the final result was a simple three dimensional scaled vector matching to every movie. Below is an example of 2 of the 3 dimensions being graphed for 1000 of the ~8000 movies used, with three movies being labeled:

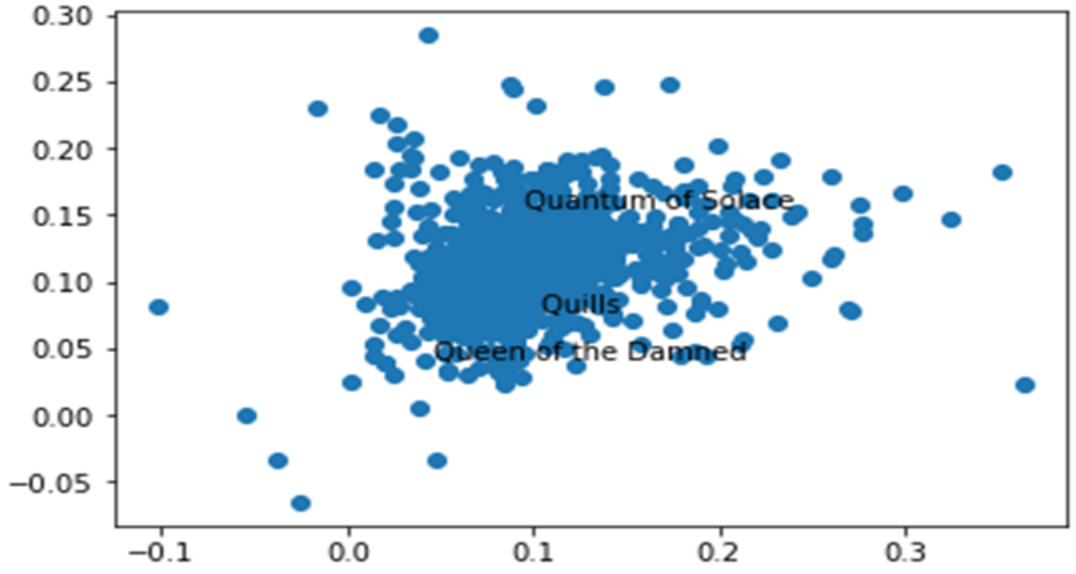


Figure 9: movie clustering

Although easy to interpret this model did have its drawbacks as well. As you can see above, the majority of the movies fell in the same area, making the clustering somewhat arbitrary. Below is a graph that shows the size of each cluster when 300 clusters were implemented. With any fewer clusters, there was a larger cluster with over 300 movies in it every time.

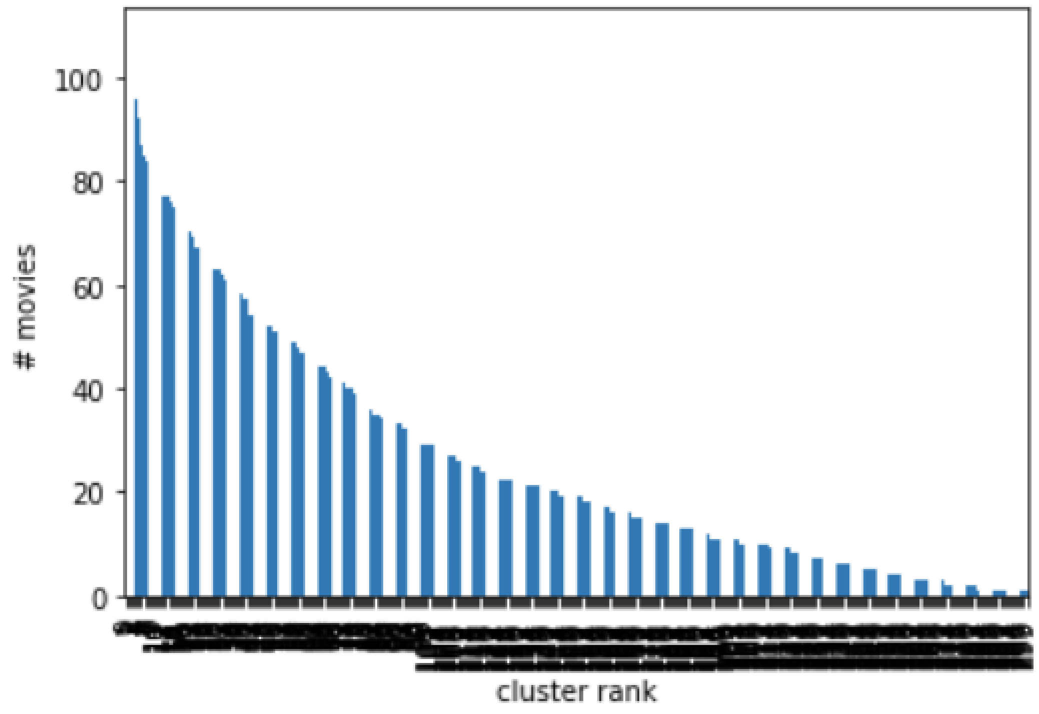


Figure 10: 300 clusters of various sizes

<u>Comparison</u>

For the comparison portion, the team took a step back to observe each of the methods and outputs generated from all three models from both a features standpoint and an outcomes standpoint. We started with a comparison of the features.

In model 1, the most common words were used as features to generate the k-means clustering. This method contrasts the tfidf method used in model 2 because the tfidf method penalizes the model for overlearning the most common and the most infrequent words. As a result, these models are producing different features that are then plugged into k-means models. Both models had stop words removed, with tfidf removing them automatically by weights and the first model by filtering them out. The third model, in contrast to both, does not rely on frequency of words, but rather all the words are used to create vector values for each movie, which is then treated as the feature for the k-means model. Stop words were filtered out before the word2vec model was used.

Overall, each model contained 100 clusters (third model was modified), with all three models having one larger cluster and many smaller clusters that contained outlier movies. The clusters each had movies that matched genres but also movies that strayed from this metric. This was surprising because each set of features were different, yet the k-means models were so similar.

Interpretation

From the beginning, the teams' goal was to utilize the plot to find movie recommendations that bypassed feature-heavy traditional genre, user review-based models, like those seen on Netflix or other streaming services. The rationale behind this decision followed the idea that people want to explore genres and themes and should be suggested movies that allow this to happen. The team also wanted to make recommendations with a simple, less computationally heavy method; oftentimes in recommender systems the movie content is one small part of a larger more complex system. The third goal was realism; the team wanted a recommender system that was realistic and could be used effectively in a commercial system.

The models achieved what the team wanted to achieve, they produced non-traditional clusters that not only took minimal work computationally, but could also easily be applied to a commercial setting. The clusters were made up of classic genre matched movies, but also movies that one would not expect to see in the same bin. Further steps would be to see how these recommendations perform with a user audience, based on their ratings of the suggested movies.

Reference

[1] https://www.kaggle.com/mananjhaveri/imdb-movies-data/tasks?taskId=2170

[2] https://www.kaggle.com/jrobischon/wikipedia-movie-plots

[3] https://www.kaggle.com/cryptexcode/mpst-movie-plot-synopses-with-tags