Milestone_1_Submission

April 5, 2017

1 Final Project - Predicting Movie Genres!

Welcome to the final project of CS109b.

The overall theme of the final project is movie data with a focus on movie genre prediction, because it is an area where we all are more or less application domain experts. First, you will explore your data and the challenges of the problem by exploratory data analysis. Use visualizations to find features that correlate with movie genres. These can be extracted from the movie posters, or meta data, or other data you gather, for example plot summaries or even movie transcripts. You will then compare traditional statistical or machine learning methods like generalized additive models, random forest, Bayesian prediction methods, boosting, and SVM, to deep learning models for movie genre prediction.

For this project you will work in teams of 3-4 people and there are weekly milestones to guide you along the way. Even though the milestones are graded, they are mainly in place to make sure you stay in contact with your TF and make progress with the project. Throughout the project you also have room for creativity and to pursue your own ideas. While you need to hand in the milestones at the appropriate due date, there is nothing preventing you from working on a later milestone ahead of time. We suggest that you read through the whole project and all milestones in the beginning to be able to plan ahead. The project is pretty open ended, so you can be creative and let your data science knowledge shine!

For each milestone you will submit a notebook, in raw (`.ipynb`) and PDF formats, containing the deliverables of that week and the extra work you did so far. The notebooks need to contain your code, comments, explanations, thoughts, and visualizations. The final deliverables are a two-minute screencast, a report in paper style for a general data science audience, and all your data and code that you developed throughout the project.

Movie genre header

Below is a description of the data and the milestones with their due dates. All work is due by 11:59PM on the due date unless otherwise specified. We expect you to have the mandatory parts finished by the milestone due dates, and there will be no extensions. However, we strongly encourage you to plan ahead. For example, you need to think about the classification task early on to plan how you want to assemble your training data, and it is beneficial to start the deep learning work as early as possible. There is nothing hindering you to already train a model in the EDA phase to get a better feel for what challenges might lay ahead with the data. You should also see the milestone requirements as a basis for your own creativity, and we expect that most of you will go beyond the mandatory deliverables. For example, if you have a great idea about an interesting question that has to do with movie genre, but cannot be answered with the data from TMDb or IMDb, feel free to gather more data from somewhere else.

We provide a data interface in Python, because it is convenient for IMDb, and we will use Python for the deep learning part. Specifically we will use Keras, a deep learning library that provides a high level interface to Google's Tensorflow framework for deep learning. However, if you feel that you prefer to do some of the work, e.g., visualizations or data cleanup, in R then feel free to use it. You can also use Spark to preprocess your data, especially if you collect large amounts of it from other sources.

Important: Your grade for a milestone will depend on the required deliverables you submit at the due date for that milestone. But every milestone, especially the final project submission, can contain additional cool work you did that goes beyond the deliverables spelled out below.

1.0.1 Logistics

Please adhere to the following guidelines for all submissions: - one submission per team - notebooks should be submitted as PDF and as raw (`.ipynb`) version - all notebooks should be executed so they contain relevant visualizations, and other results - try to make it as easy as possible for the TFs to get all relevant information about your work - do not submit big data sets, please provide a readme file with a link instead - the final report should also be submitted as pdf

1.0.2 Movie Data:

The project is based on two different sources of movie data: IMDb and TMDb. TMDb is great, because it provides the movie posters in addition to the metadata. This is crucial for the deep learning part, in which you will try to predict movie genres from posters. IMDb has more metadata available and will supplement the TMDb data you have.

TMDb provides an easy to use API that allows you to download the data selectively. IMDb does not provide an API, but there is a Python interface available to access the metadata. We will use IMDbPY, which is already installed on the AMI and virtual box images for your convenience.

Important: Please remember to limit your data rate when obtaining the data. Play nicely and do not just spam servers as fast as you can. This will prevent your IP from getting banned. The easiest way to do is is to use the sleep function in Python.

1.0.3 Milestone 1: Getting to know your data, due Wednesday, April 5, 2017

In the beginning you should get acquainted with the data sources and do some EDA. Sign up for the TMDb API, and try to download the poster of your favorite movie from within your notebook. Compare the genre entries of IMDb and TMDb for this movie and see if they are the same. Think about and write down some questions that you would like to answer in the following weeks.

Keep the storytelling aspect of your final report in mind and do some pen and paper sketches about the visualizations you would like to produce. Include photographs of those sketches in your notebook.

Most of the time a data scientist spends on a project is spend on cleaning the data. We are lucky that the data we have is already pretty clean. The Python interface to the IMDb ftp files does a lot of the additional work of cleaning as well. However, you will notice that the genre list for each movie from both databases can have different lengths. This needs to be changed in order to train a model to predict the movie genre. It is up to you to think about possible ways to address this problem and to implement one of them. There is no absolute right answer here. It depends on your interests and which questions you have in mind for the project.

Optionally, you could also scrape additional data sources, such as Wikipedia, to obtain plot summaries. That data may give you additional useful features for genera classification.

To guide your decision process, provide at least one visualization of how often genres are mentioned together in pairs. Your visualization should clearly show if a horror romance is more likely to occur in the data than a drama romance.

The notebook to submit for this milestone needs to at least include:

- API code to access the genre and movie poster path of your favorite movie
- Genre for this movie listed by TMDb and IMDb
- A list of the 10 most popular movies of 2016 from TMDb and their genre obtained via the API
- Comments on what challenges you see for predicting movie genre based on the data you have, and how to address them
- Code to generate the movie genre pairs and a suitable visualization of the result
- Additional visualization sketches and EDA with a focus on movie genres
- A list of questions you could answer with this and related data. Get creative here!

The EDA questions do not necessarily have to tie into the modeling part later on. Think freely about things that might be interesting, like which actors are very specific to a genre? Are action movies more prone to producing sequels than romances? However, as you keep the focus on movie genres, think also about correlations you might discover that can help building features from the metadata for prediction. Is the length of a movie title correlated with genre?

1.1 Milestone 1 Submission

1.1.1 CS109b Project - Group 37 (Alexander Dubitskiy, Keenan Venuti, Timur Zambalayev)

Github repository of the project: https://github.com/adubitskiy/cs109b

First let's do some initial exploration of the the TMDb and IMDb data.

1.1.2 API code to access the genre and movie poster path of your favorite movie

```
In [1]: from IPython.display import Image
        from IPython.core.display import HTML
        import tmdbsimple as tmdb
        import ast
        import operator
        import numpy as np
```

```
import pandas as pd
from imdb import IMDb
```

```
In [2]: tmdb.API_KEY = 'a995a7fe53e021d77d82b99428850ff1'
```

```
In [3]: conf = tmdb.Configuration()
        c_info = conf.info()
```

Now let's look at the list of the poster sizes from TMDb.

```
In [4]: img_conf = c_info['images']
        img_conf['poster_sizes']
```

```
Out[4]: [u'w92', u'w154', u'w185', u'w342', u'w500', u'w780', u'original']
```

What genres do we have in TMDb? How many?

```
In [5]: genres = tmdb.Genres().list()
        tmdb_genres = [g['name'] for g in genres['genres'] ]
        print(len(tmdb_genres))
        print(tmdb_genres)
```

```
19
[u'Action', u'Adventure', u'Animation', u'Comedy', u'Crime', u'Documentary', u'Dran
```

Here's the list of the genres from IMDb (copied from the site). So it's 19 genres in TMDb vs 27 for IMDb.

```
In [6]: imdb_genres = ['Action','Adventure','Animation','Biography','Comedy','Crime
        print(len(imdb_genres))
        print(imdb_genres)
```

```
27
['Action', 'Adventure', 'Animation', 'Biography', 'Comedy', 'Crime', 'Documentary',
```

Our favorite movie will be "Ghost in the Shell". Let's search for it.

```
In [7]: search = tmdb.Search()
        response = search.movie(query="ghost in the shell")
        for s in search.results:
            print(s['title'], s['id'], s['release_date'], s['popularity'])
```

```
(u'Ghost in the Shell', 315837, u'2017-03-29', 53.86811)
(u'Ghost in the Shell', 9323, u'1995-11-18', 6.135032)
(u'Ghost in the Shell 2.0', 14092, u'2008-07-12', 2.703536)
(u'Ghost in the Shell 2: Innocence', 12140, u'2004-03-06', 2.601958)
(u'Ghost in the Shell Arise - Border 5: Pyrophoric Cult', 381519, u'2015-08-26', 2.
(u'Ghost in the Shell: The New Movie', 334376, u'2015-06-20', 2.265999)
```

```
(u'Ghost in the Shell: Stand Alone Complex - Solid State Society', 18874, u'2007-02
(u'Ghost in the Shell: Stand Alone Complex - The Laughing Man', 18839, u'2005-01-01
(u'Ghost In The Shell: The Movie Virtual Reality Diver', 384217, u'2016-02-02', 1.1
(u'Ghost in the Shell Arise - Border 1: Ghost Pain', 196750, u'2013-06-22', 1.46676
(u'Pandora in the Crimson Shell: Ghost Urn', 377885, u'2015-12-05', 1.024687)
(u'Ghost in the Shell Arise - Border 2: Ghost Whispers', 212168, u'2013-11-29', 1.4
(u'Ghost in the Shell Arise - Border 3: Ghost Tears', 240341, u'2014-06-28', 1.5010
(u'Ghost in the Shell Arise - Border 4: Ghost Stands Alone', 279254, u'2014-09-06',
(u'Ghost in the Shell: Stand Alone Complex - Individual Eleven', 111224, u'2006-01-
(u'K\xf4kaku kid\xf4tai Stand Alone Complex - Solid State Society 3D', 446699, u'20
```

```
In [8]: tmdb_movie_id = 315837
        tmdb_movie = tmdb.Movies(tmdb_movie_id)
        info = tmdb_movie.info()
        info['title']

Out[8]: u'Ghost in the Shell'
```

Its genres from TMDB:

```
In [9]: for tmdb_genre in info['genres']:
            print tmdb_genre['name']

Action
Drama
Science Fiction
```

The movie poster image from TMDB:

```
In [10]: Image(url= img_conf['base_url'] + img_conf['poster_sizes'][1] + tmdb_movie

Out[10]: <IPython.core.display.Image object>
```

1.1.3 Genre for this movie listed by TMDb and IMDb

So we found TMDb genres (Action, Drama, Science Fiction) and the poster for our favorite movie.

```
In [11]: imdb_service = IMDb()
         imdb_movie_id = info['imdb_id'][2:]
         imdb_movie = imdb_service.get_movie(imdb_movie_id)
```

The movie genres from IMDB

```
In [12]: for imdb_genres in imdb_movie.data['genres']:
             print imdb_genres

Action
Crime
Drama
Mystery
Sci-Fi
Thriller
```

1.1.4 A list of the 10 most popular movies of 2016 from TMDb and their genre obtained via the API

IMDB has a comprehensive list of genres, we use IMDB as the main source

```
In [13]: def imdbGenres(imdb_movie):
             return imdb_movie.data['genres']

         def tmdbGenres(tmdb_movie):
             tmdb_movie_genres = [g['name'] for g in tmdb_movie['genres']]
             tmdb_movie_genres = [g.replace('Science Fiction', 'Sci-Fi') for g in t
             return tmdb_movie_genres

         def combinedGenres(imdb_movie, tmdb_movie):
             return list(set(imdbGenres(imdb_movie)) | set(tmdbGenres(tmdb_movie)))

In [14]: discover = tmdb.Discover()
         disc_result_2016 = discover.movie(query = 'primary_release_year=2016&sort_
```

A list of the 10 most popular movies of 2016 from TMDb. We can see differences in IMDB and TMDB genres.

```
In [15]: top_movies_2016 = disc_result_2016['results']
         for top_movie in top_movies_2016[:10]:
             tmdb_movie = tmdb.Movies(top_movie['id']).info()
             imdb_movie = imdb_service.get_movie(info['imdb_id'][2:])
             print top_movie['id'], top_movie['title']
             print 'combined: ', combinedGenres(imdb_movie, tmdb_movie)
             print 'imdb: ', imdbGenres(imdb_movie)
             print 'tmdb: ', tmdbGenres(tmdb_movie)
             print '---------------'
```

```
321612 Beauty and the Beast
combined:  [u'Mystery', u'Drama', u'Music', u'Sci-Fi', u'Fantasy', u'Action', u'Ror
imdb:  [u'Action', u'Crime', u'Drama', u'Mystery', u'Sci-Fi', u'Thriller']
tmdb:  [u'Fantasy', u'Music', u'Romance']
---------------
263115 Logan
combined:  [u'Mystery', u'Drama', u'Sci-Fi', u'Action', u'Thriller', u'Crime']
imdb:  [u'Action', u'Crime', u'Drama', u'Mystery', u'Sci-Fi', u'Thriller']
tmdb:  [u'Action', u'Drama', u'Sci-Fi']
---------------
335797 Sing
combined:  [u'Mystery', u'Sci-Fi', u'Family', u'Crime', u'Drama', u'Animation', u'M
imdb:  [u'Action', u'Crime', u'Drama', u'Mystery', u'Sci-Fi', u'Thriller']
tmdb:  [u'Animation', u'Comedy', u'Drama', u'Family', u'Music']
---------------
293167 Kong: Skull Island
combined:  [u'Mystery', u'Drama', u'Sci-Fi', u'Fantasy', u'Action', u'Adventure', u
```

```
imdb:  [u'Action', u'Crime', u'Drama', u'Mystery', u'Sci-Fi', u'Thriller']
tmdb:  [u'Sci-Fi', u'Action', u'Adventure', u'Fantasy']
--------------
315837 Ghost in the Shell
combined:  [u'Mystery', u'Drama', u'Sci-Fi', u'Action', u'Thriller', u'Crime']
imdb:  [u'Action', u'Crime', u'Drama', u'Mystery', u'Sci-Fi', u'Thriller']
tmdb:  [u'Action', u'Drama', u'Sci-Fi']
--------------
135397 Jurassic World
combined:  [u'Mystery', u'Drama', u'Sci-Fi', u'Action', u'Adventure', u'Thriller',
imdb:  [u'Action', u'Crime', u'Drama', u'Mystery', u'Sci-Fi', u'Thriller']
tmdb:  [u'Action', u'Adventure', u'Sci-Fi', u'Thriller']
--------------
259316 Fantastic Beasts and Where to Find Them
combined:  [u'Mystery', u'Drama', u'Sci-Fi', u'Fantasy', u'Action', u'Adventure', u
imdb:  [u'Action', u'Crime', u'Drama', u'Mystery', u'Sci-Fi', u'Thriller']
tmdb:  [u'Adventure', u'Action', u'Fantasy']
--------------
295693 The Boss Baby
combined:  [u'Mystery', u'Drama', u'Animation', u'Sci-Fi', u'Family', u'Action', u'
imdb:  [u'Action', u'Crime', u'Drama', u'Mystery', u'Sci-Fi', u'Thriller']
tmdb:  [u'Animation', u'Comedy', u'Family']
--------------
157336 Interstellar
combined:  [u'Mystery', u'Drama', u'Sci-Fi', u'Action', u'Adventure', u'Thriller',
imdb:  [u'Action', u'Crime', u'Drama', u'Mystery', u'Sci-Fi', u'Thriller']
tmdb:  [u'Adventure', u'Drama', u'Sci-Fi']
--------------
381288 Split
combined:  [u'Mystery', u'Drama', u'Sci-Fi', u'Action', u'Horror', u'Thriller', u'C
imdb:  [u'Action', u'Crime', u'Drama', u'Mystery', u'Sci-Fi', u'Thriller']
tmdb:  [u'Horror', u'Thriller']
--------------
```

1.1.5 Movie genre dataset for exploration

In order to do some of the initial exploration we created movie genre dataset. We used TMDb API to download data for top 11.3k movies. We are building a local IMDB database and we intend to use actors/directors for the further work.

In [16]:
```python
import time
from requests import HTTPError

start_time = time.time()

tmdb.API_KEY = 'a995a7fe53e021d77d82b99428850ff1'
```

```python
discover = tmdb.Discover()

start_page = 1
num_pages = 2 # 1000

for page in xrange(start_page, num_pages + 1):
    if page != start_page:
        time.sleep(7)

    elapsed_mins = (time.time() - start_time) / 60.0

    print 'page: %d, elapsed mins: %.2f' % (page, elapsed_mins)
    discover_result = discover.movie(page=page, sort_by='popularity.desc')

    movies = discover_result['results']

    row_list = []
    for movie in movies:
        movie_id = movie['id']
        print movie_id, movie['title']
        try:
            tmdb_movie = tmdb.Movies(movie_id).info()
            row_list.append(tmdb_movie)
        except HTTPError as e:
            print str(e)

    df = pd.DataFrame(row_list)

    with open('tmdb_movies.csv', 'a') as csv_file:
        df.to_csv(csv_file, header=False, index=False, encoding='utf-8')
```

```
page: 1, elapsed mins: 0.00
321612 Beauty and the Beast
263115 Logan
335797 Sing
293167 Kong: Skull Island
315837 Ghost in the Shell
135397 Jurassic World
259316 Fantastic Beasts and Where to Find Them
295693 The Boss Baby
157336 Interstellar
381288 Split
127380 Finding Dory
76341 Mad Max: Fury Road
395992 Life
118340 Guardians of the Galaxy
293660 Deadpool
330459 Rogue One: A Star Wars Story
```

```
284052 Doctor Strange
329865 Arrival
271110 Captain America: Civil War
245891 John Wick
page: 2, elapsed mins: 0.13
346672 Underworld: Blood Wars
305470 Power Rangers
334543 Lion
15206 The Mother of Tears
22 Pirates of the Caribbean: The Curse of the Black Pearl
140607 Star Wars: The Force Awakens
228150 Fury
122917 The Hobbit: The Battle of the Five Armies
269149 Zootopia
381284 Hidden Figures
324786 Hacksaw Ridge
356305 Why Him?
131631 The Hunger Games: Mockingjay - Part 1
274870 Passengers
246655 X-Men: Apocalypse
121856 Assassin's Creed
278 The Shawshank Redemption
177572 Big Hero 6
24428 The Avengers
209112 Batman v Superman: Dawn of Justice
```

The first page (top 20 popular movies) is what you can see at this url: https://www.themoviedb.org/movie

We loaded first 11291 movies, then we encountered various problems with the data (movies with no data, repeated instances of the same movies, encoding issues). The load took more than 2 hours. We also had to throttle the load (note using sleep function for 7 seconds). It's because TMDb was limiting the number of requests. We believe it's 40 requests per 10 seconds.

Here is the link to this dataset (10Mb): https://github.com/adubitskiy/cs109b/blob/master/Milestone_1/tm

1.1.6 Comments on what challenges you see for predicting movie genre based on the data you have, and how to address them

One problem that we see with predicting movie genres is that in most cases there are multiple different genres attached to each movie.

1) One way to solve this is we can consider one genre at any given moment. For example, we can try to predict for all the movies in the dataset whether they belong to Action genre (yes or no, 1 or 0). We can apply a Logistic regression (for example) and evaluate the accuracy.

2) Another possible approach is to consider all possible genre combinations that we see in the dataset. For example, Action + Fantasy + Adventure. We can treat each unique combination as an indidual separate class. One possible problem with this approach is that we could have

a lot of unique genre combinations. A variation of this approach could be considering only pairs or tuples of three of genres.

Using either methodology, we see class imbalance as a future problem and something to account for. We do not want a supervised trained model to over classify a class to get the best accuracy if that accuracy is not real-world applicable.

Sample multi-label model First approach can be implemented by usoing OneVsRestClassifier. We use text of the movies overviews to predict the genres and we predict multiple lables per observation.

```
In [17]: from sklearn.feature_extraction.text import CountVectorizer
         from sklearn.preprocessing import MultiLabelBinarizer
         from sklearn.multiclass import OneVsRestClassifier
         from sklearn.svm import LinearSVC
         import nltk.data
         from nltk.corpus import stopwords
         %matplotlib inline
         nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to C:\Windows\ServiceProfile
[nltk_data]     s\LocalService\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
Out[17]: True
```

```
In [18]: TMDB_MOVIES_COLUMN_NAMES = [
             'adult', 'backdrop_path', 'belongs_to_collection', 'budget', 'genres',
             'original_language', 'original_title', 'overview', 'popularity', 'post
             'production_countries', 'release_date', 'revenue', 'runtime', 'spoken_
             'video', 'vote_average', 'vote_count',
         ]

         def load_tmdb_movies():
             df = pd.read_csv('Milestone_1/tmdb_movies_11291.csv', header=None, nam
             for column_name in ['genres', 'spoken_languages']:
                 df[column_name] = df[column_name].map(lambda d: ast.literal_eval(d
             return df
```

Load movies and prepare the training set by filtering out movies without genres or without reviews

```
In [19]: tmdb_movies_df = load_tmdb_movies()
         has_overview = ~tmdb_movies_df['overview'].isnull()
         has_genre = ~tmdb_movies_df['genres'].apply(lambda x: not x )
         tmdb_movies_with_overview = tmdb_movies_df[has_overview & has_genre]
```

Convert the overviews to a matrix of token counts

```
In [20]: vectorizer = CountVectorizer(
             stop_words = stopwords.words("english"),
             token_pattern = '[a-zA-Z]+[0-9]*',
             max_df = 0.9,
             min_df = 5,
             dtype=np.float32 )
         X = vectorizer.fit_transform(tmdb_movies_with_overview['overview'].values)
         print 'predictor matrix shape:', X.shape

predictor matrix shape: (11022L, 8707L)
```

```
In [21]: feature_names = np.array(vectorizer.get_feature_names())
         feature_names

Out[21]: array([u'aaron', u'abandon', u'abandoned', ..., u'zone', u'zoo', u'zooey'],
               dtype='<U17')
```

Prepare labels for each movie

```
In [22]: labels = tmdb_movies_with_overview['genres'].apply(lambda x: [g['name'] fo
         mlb = MultiLabelBinarizer()
         y = mlb.fit_transform(labels)
         print 'label matrix shape:', y.shape

label matrix shape: (11022L, 20L)
```

Fit the classifier and predict using the training set

```
In [23]: oneVsResClassifier = OneVsRestClassifier(LinearSVC(random_state=0)).fit(X,
         predict = oneVsResClassifier.predict(X)

In [24]: print 'First observation actual label', y[1]
         print 'First observation predicted label: ', predict[1]
         print 'Accuracy on the training set: ', oneVsResClassifier.score(X, y)

First observation actual label [1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0]
First observation predicted label:  [1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0]
Accuracy on the training set:  0.992741789149
```

We cannot comment on the generalization capabilities of the model but the idea of using OneVsRestClassifier seems to be working out fine. We'll explore the idea further in the following milestones probably using different set of features and with proper cross-validation.

1.1.7 Code to generate the movie genre pairs and a suitable visualization of the result

Let's look at the number of genres per movie.

```
In [131]: import ast
          from collections import Counter

          import matplotlib.pyplot as plt
          import numpy as np
          import operator
          import pandas as pd
          import sys
          from IPython.display import display


          TMDB_MOVIES_COLUMN_NAMES = [
              'adult', 'backdrop_path', 'belongs_to_collection', 'budget', 'genres'
              'original_language', 'original_title', 'overview', 'popularity', 'pos
              'production_countries', 'release_date', 'revenue', 'runtime', 'spoken
              'video', 'vote_average', 'vote_count',
          ]

          def load_tmdb_movies():
              df = pd.read_csv('Milestone_1/tmdb_movies_11291.csv', header=None, na
              for column_name in ['genres', 'spoken_languages']:
                  df[column_name] = df[column_name].map(lambda d: ast.literal_eval
              return df


          tmdb_movies_df = load_tmdb_movies()
          print 'shape:', tmdb_movies_df.shape
          display(tmdb_movies_df.head())
```

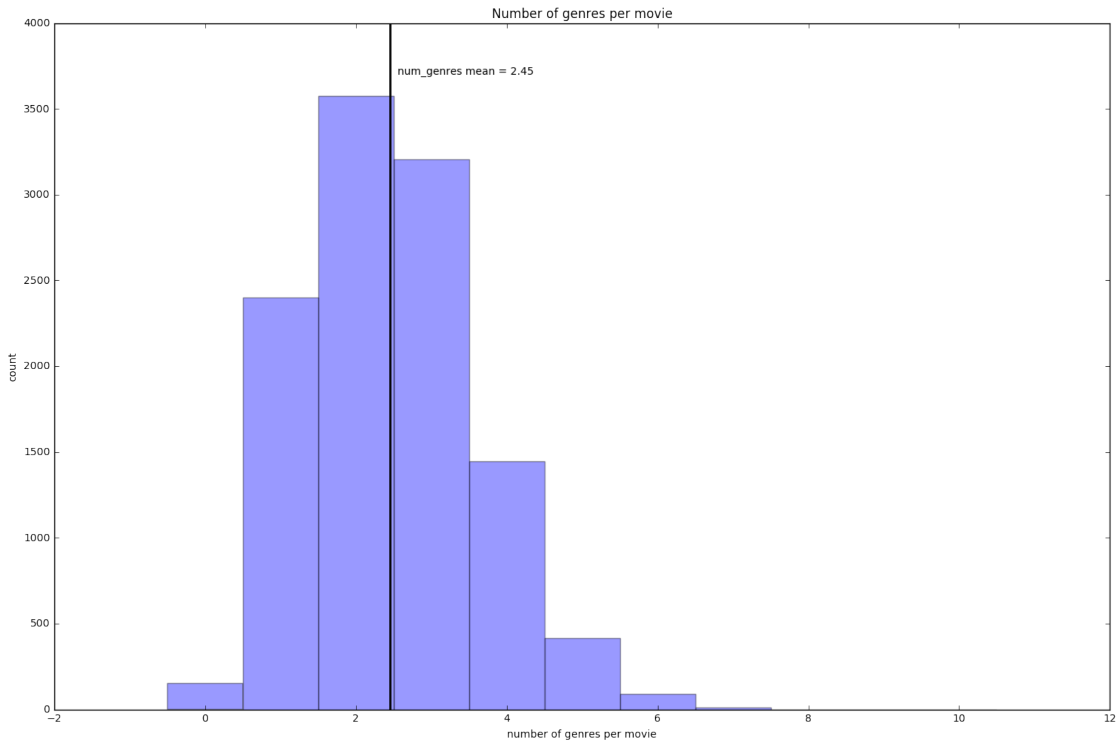
```
shape: (11291, 25)


   adult                    backdrop_path  \
0  False  /6aUWe0GSl69wMTSWWexsorMIvwU.jpg
1  False  /5pAGnkFYSsFJ99ZxDIYnhQbQFXs.jpg
2  False   /fxDXp8un4qNY9b1dLd7SH6CKzC.jpg
3  False  /pGwChWiAY1bdoxL79sXmaFBlYJH.jpg
4  False  /dkMD5qlogeRMiEixC4YNPUvax2T.jpg

                               belongs_to_collection     budget  \
0                                                NaN  160000000
1  {u'backdrop_path': u'/Abnosho2v3bcdvDww6T7Hfec...   97000000
2                                                NaN   75000000
3                                                NaN  190000000
4  {u'backdrop_path': u'/pJjIH9QN0OkHFV9eue6XfRVn...  150000000
```

```
                                              genres  \
0  [{u'id': 14, u'name': u'Fantasy'}, {u'id': 104...
1  [{u'id': 28, u'name': u'Action'}, {u'id': 18, ...
2  [{u'id': 16, u'name': u'Animation'}, {u'id': 3...
3  [{u'id': 878, u'name': u'Science Fiction'}, {u...
4  [{u'id': 28, u'name': u'Action'}, {u'id': 12, ...

                                            homepage      id    imdb_id  \
0  http://movies.disney.com/beauty-and-the-beast-...  321612  tt2771200
1             http://www.foxmovies.com/movies/logan  263115  tt3315342
2                       http://www.singmovie.com/  335797  tt3470600
3                 http://kongskullislandmovie.com/  293167  tt3731562
4                   http://www.jurassicworld.com/  135397  tt0369610

  original_language         original_title    ...     release_date     revenue  \
0                en  Beauty and the Beast    ...       2017-03-15   779548842
1                en                  Logan    ...       2017-02-28   571616411
2                en                   Sing    ...       2016-11-23   601303829
3                en    Kong: Skull Island    ...       2017-03-08   400323204
4                en         Jurassic World    ...       2015-06-09  1513528810

  runtime                                  spoken_languages    status  \
0   123.0       [{u'iso_639_1': u'en', u'name': u'English'}]  Released
1   141.0       [{u'iso_639_1': u'de', u'name': u'Deutsch'}]  Released
2   108.0  [{u'iso_639_1': u'en', u'name': u'English'}, {...  Released
3   118.0       [{u'iso_639_1': u'en', u'name': u'English'}]  Released
4   124.0       [{u'iso_639_1': u'en', u'name': u'English'}]  Released

               tagline                 title  video vote_average vote_count
0         Be our guest.  Beauty and the Beast  False          7.1       1120
1      His Time Has Come                 Logan  False          7.6       2003
2  Auditions begin 2016.                  Sing  False          6.7        982
3      All hail the king    Kong: Skull Island  False          6.1        838
4      The park is open.        Jurassic World  False          6.5       6680

[5 rows x 25 columns]
```

```
In [132]: def get_fig_size(nrows=1):
              return 15, 10 * nrows


          def explore_num_genres_per_movie(tmdb_movies_df):
              print 'total number of movies: %d' % len(tmdb_movies_df)

              genres_rows = tmdb_movies_df['genres']
```

```
    num_genres_per_movie_list = []
    for genres in genres_rows:
        num_genres = len(genres)
        num_genres_per_movie_list.append(num_genres)

    unique_num_genres_per_movie = set(num_genres_per_movie_list)
    print 'unique values of number of genres per movie: %s' % unique_num_

    num_genres_mean = np.mean(num_genres_per_movie_list)
    print 'mean number of genres per movie: %.3f' % num_genres_mean

    _, ax = plt.subplots(1, 1, figsize=get_fig_size())
    ax.hist(num_genres_per_movie_list, bins=np.arange(-0.5, 11.0, step=1.
    ax.axvline(x=num_genres_mean, linewidth=2, color='k')
    plt.text(num_genres_mean + 0.1, 3700, 'num_genres mean = %.2f' % num_
    ax.set_xlabel('number of genres per movie')
    ax.set_ylabel('count')
    ax.set_title('Number of genres per movie')

    plt.tight_layout()
    plt.show()


explore_num_genres_per_movie(tmdb_movies_df)
```

```
total number of movies: 11291
unique values of number of genres per movie: set([0, 1, 2, 3, 4, 5, 6, 7])
mean number of genres per movie: 2.447
```



Our dataset contains top 11,291 TMDb movies by popularity.

We can notice that some movies have zero genres. The mean number of genres per movie is 2.45. The maximum number of genres for one movie is 7.

Let's explore genre counts.

```
In [26]: def explore_genre_counts(tmdb_movies_df):
             num_movies = len(tmdb_movies_df)

             genres_rows = tmdb_movies_df['genres']
             genre_list = []
             for genres in genres_rows:
                 for genre in genres:
                     genre_list.append(genre['name'])

             unique_genres = set(genre_list)
             print 'number of unique genres: %d' % len(unique_genres)

             counter = Counter(genre_list)
             counter_dict = {k: float(v) / num_movies for k, v in dict(counter).ite

             sorted_counter_items = sorted(counter_dict.items(), key=operator.itemg
             for k in sorted_counter_items:
                 print '%s: %.1f%%' % (k[0], k[1] * 100.0)
```

```
explore_genre_counts(tmdb_movies_df)
```

```
number of unique genres: 20
Drama: 46.5%
Comedy: 31.7%
Thriller: 25.9%
Action: 23.9%
Romance: 16.8%
Adventure: 14.8%
Crime: 13.1%
Horror: 12.1%
Science Fiction: 10.9%
Family: 10.0%
Fantasy: 9.0%
Animation: 8.0%
Mystery: 7.2%
History: 3.9%
War: 3.1%
Music: 2.7%
Western: 2.0%
Documentary: 1.7%
TV Movie: 1.1%
Foreign: 0.5%
```

There are 20 unique genres. The most popular is Drama. 46.5% of the movies had Drama as one its genres. The least popular is Foreign (0.5%).

Now we'll look at genre pairs.

```
In [111]: from IPython.display import display

          def get_genre_list(genres_rows):
              return [genre['name'] for genres in genres_rows for genre in genres]


          def get_pair_list(genres_rows):
              pair_list = []
              for genres in genres_rows:
                  for i1 in xrange(len(genres) - 1):
                      g1 = genres[i1]['name']
                      for i2 in xrange(i1 + 1, len(genres)):
                          g2 = genres[i2]['name']
                          k1 = g1 + ':' + g2
                          k2 = g2 + ':' + g1
                          pair_list.append(k1)
```

```
                pair_list.append(k2)
    return pair_list


def get_pair_matrix(pair_counter, sorted_genres):
    num_genres = len(sorted_genres)
    pair_matrix = np.full((num_genres, num_genres), 0, dtype=np.int)
    for i in xrange(len(sorted_genres)):
        g1 = sorted_genres[i]
        for j in xrange(i + 1, len(sorted_genres)):
            g2 = sorted_genres[j]
            count = pair_counter[g1 + ":" + g2]
            pair_matrix[i, j] = count
            pair_matrix[j, i] = count
    return pair_matrix


def get_sorted_genres(genre_list):
    genre_counter = Counter(genre_list)
    sorted_counter_items = sorted(genre_counter.items(), key=operator.ite
    sorted_genres = [i[0] for i in sorted_counter_items]
    return sorted_genres


def explore_genre_pairs(tmdb_movies_df):
    genres_rows = tmdb_movies_df['genres']
    genre_list = get_genre_list(genres_rows)
    pair_list = get_pair_list(genres_rows)

    sorted_genres = get_sorted_genres(genre_list)

    pair_counter = Counter(pair_list)

    print len(pair_list)
    print pair_counter.most_common(10)
    print pair_counter.most_common()[-10:-1]
    print len(pair_counter)

    pair_matrix = get_pair_matrix(pair_counter, sorted_genres)

    pair_df = pd.DataFrame(pair_matrix, columns=sorted_genres, index=sort

    display(pair_df)


explore_genre_pairs(tmdb_movies_df)
```

```
55162
```

```
[(u'Thriller:Drama', 1316), (u'Drama:Thriller', 1316), (u'Drama:Romance', 1277), (u
[(u'Thriller:Documentary', 1), (u'Science Fiction:History', 1), (u'Family:War', 1),
366
```

	Drama	Comedy	Thriller	Action	Romance	Adventure	Crime
Drama	0	1145	1316	839	1277	434	854
Comedy	1145	0	231	590	908	467	331
Thriller	1316	231	0	1135	150	340	887
Action	839	590	1135	0	144	929	617
Romance	1277	908	150	144	0	142	89
Adventure	434	467	340	929	142	0	111
Crime	854	331	887	617	89	111	0
Horror	263	184	717	207	31	58	79
Science Fiction	253	239	417	619	78	390	43
Family	221	553	15	142	105	453	18
Fantasy	260	332	137	308	148	420	24
Animation	94	294	25	193	43	315	15
Mystery	411	77	548	126	60	58	246
History	377	19	47	123	63	75	15
War	278	31	59	129	52	57	7
Music	161	132	11	18	95	17	16
Western	93	39	24	101	29	58	14
Documentary	15	10	1	5	0	4	3
TV Movie	42	31	20	19	18	21	3
Foreign	29	10	14	22	8	4	2

	Horror	Science Fiction	Family	Fantasy	Animation	Mystery
Drama	263	253	221	260	94	411
Comedy	184	239	553	332	294	77
Thriller	717	417	15	137	25	548
Action	207	619	142	308	193	126
Romance	31	78	105	148	43	60
Adventure	58	390	453	420	315	58
Crime	79	43	18	24	15	246
Horror	0	268	2	120	18	206
Science Fiction	268	0	128	257	143	85
Family	2	128	0	335	541	19
Fantasy	120	257	335	0	225	51
Animation	18	143	541	225	0	16
Mystery	206	85	19	51	16	0
History	6	1	3	5	5	6
War	7	5	1	5	3	7
Music	8	8	56	30	34	6
Western	9	5	4	5	2	5
Documentary	4	1	6	2	4	1
TV Movie	10	15	35	19	14	16
Foreign	6	7	3	5	3	1

	History	War	Music	Western	Documentary	TV Movie	Foreign
Drama	377	278	161	93	15	42	29
Comedy	19	31	132	39	10	31	10
Thriller	47	59	11	24	1	20	14
Action	123	129	18	101	5	19	22
Romance	63	52	95	29	0	18	8
Adventure	75	57	17	58	4	21	4
Crime	15	7	16	14	3	3	2
Horror	6	7	8	9	4	10	6
Science Fiction	1	5	8	5	1	15	7
Family	3	1	56	4	6	35	3
Fantasy	5	5	30	5	2	19	5
Animation	5	3	34	2	4	14	3
Mystery	6	7	6	5	1	16	1
History	0	122	8	5	11	10	4
War	122	0	3	8	7	0	0
Music	8	3	0	6	19	7	0
Western	5	8	6	0	0	1	1
Documentary	11	7	19	0	0	1	0
TV Movie	10	0	7	1	1	0	0
Foreign	4	0	0	1	0	0	0

Thriller and Drama is the most popular combination (1316 occurrences). Some combinations happen only once (e.g. Thriller and Documentary). Some combinations of genres never occurred. There are 366 out of possible 400 pairs.

When we look at this matrix of genre pairs, we can see that certain genre pairs are more likely to happen. For example, Thriller and Crime, Family and Adventure.

```
In [113]: def calculate_conditional_probabilities(tmdb_movies_df):
              genres_rows = tmdb_movies_df['genres']
              genre_list = get_genre_list(genres_rows)
              pair_list = get_pair_list(genres_rows)

              sorted_genres = get_sorted_genres(genre_list)

              pair_counter = Counter(pair_list)
              pair_matrix = get_pair_matrix(pair_counter, sorted_genres)

              genre_counter = Counter(genre_list)
              genre_items = [num_genres for genre, num_genres in genre_counter.most
              genre_totals = np.array(genre_items)[:, np.newaxis]

              cond_probs_matrix = 100.0 * pair_matrix / genre_totals
              cond_probs_df = pd.DataFrame(cond_probs_matrix, columns=sorted_genres
              display(cond_probs_df.round(1))
```

```
calculate_conditional_probabilities(tmdb_movies_df)
```

	Drama	Comedy	Thriller	Action	Romance	Adventure	Crime
Drama	0.0	21.8	25.1	16.0	24.3	8.3	16.3
Comedy	31.9	0.0	6.4	16.5	25.3	13.0	9.2
Thriller	45.1	7.9	0.0	38.9	5.1	11.6	30.4
Action	31.1	21.9	42.1	0.0	5.3	34.4	22.9
Romance	67.4	47.9	7.9	7.6	0.0	7.5	4.7
Adventure	26.0	28.0	20.4	55.6	8.5	0.0	6.6
Crime	57.9	22.4	60.1	41.8	6.0	7.5	0.0
Horror	19.2	13.5	52.5	15.1	2.3	4.2	5.8
Science Fiction	20.6	19.4	33.9	50.3	6.3	31.7	3.5
Family	19.6	49.0	1.3	12.6	9.3	40.1	1.6
Fantasy	25.5	32.5	13.4	30.2	14.5	41.2	2.4
Animation	10.4	32.6	2.8	21.4	4.8	35.0	1.7
Mystery	50.8	9.5	67.7	15.6	7.4	7.2	30.4
History	86.3	4.3	10.8	28.1	14.4	17.2	3.4
War	78.8	8.8	16.7	36.5	14.7	16.1	2.0
Music	52.1	42.7	3.6	5.8	30.7	5.5	5.2
Western	41.9	17.6	10.8	45.5	13.1	26.1	6.3
Documentary	7.9	5.2	0.5	2.6	0.0	2.1	1.6
TV Movie	35.0	25.8	16.7	15.8	15.0	17.5	2.5
Foreign	52.7	18.2	25.5	40.0	14.5	7.3	3.6

	Horror	Science Fiction	Family	Fantasy	Animation	Mystery
Drama	5.0	4.8	4.2	5.0	1.8	7.8
Comedy	5.1	6.7	15.4	9.3	8.2	2.1
Thriller	24.6	14.3	0.5	4.7	0.9	18.8
Action	7.7	22.9	5.3	11.4	7.2	4.7
Romance	1.6	4.1	5.5	7.8	2.3	3.2
Adventure	3.5	23.4	27.1	25.1	18.9	3.5
Crime	5.4	2.9	1.2	1.6	1.0	16.7
Horror	0.0	19.6	0.1	8.8	1.3	15.1
Science Fiction	21.8	0.0	10.4	20.9	11.6	6.9
Family	0.2	11.3	0.0	29.7	47.9	1.7
Fantasy	11.8	25.2	32.8	0.0	22.1	5.0
Animation	2.0	15.9	60.0	25.0	0.0	1.8
Mystery	25.5	10.5	2.3	6.3	2.0	0.0
History	1.4	0.2	0.7	1.1	1.1	1.4
War	2.0	1.4	0.3	1.4	0.8	2.0
Music	2.6	2.6	18.1	9.7	11.0	1.9
Western	4.1	2.3	1.8	2.3	0.9	2.3
Documentary	2.1	0.5	3.1	1.0	2.1	0.5
TV Movie	8.3	12.5	29.2	15.8	11.7	13.3
Foreign	10.9	12.7	5.5	9.1	5.5	1.8

	History	War	Music	Western	Documentary	TV Movie	Foreign

Drama	7.2	5.3	3.1	1.8	0.3	0.8	0.6
Comedy	0.5	0.9	3.7	1.1	0.3	0.9	0.3
Thriller	1.6	2.0	0.4	0.8	0.0	0.7	0.5
Action	4.6	4.8	0.7	3.7	0.2	0.7	0.8
Romance	3.3	2.7	5.0	1.5	0.0	0.9	0.4
Adventure	4.5	3.4	1.0	3.5	0.2	1.3	0.2
Crime	1.0	0.5	1.1	0.9	0.2	0.2	0.1
Horror	0.4	0.5	0.6	0.7	0.3	0.7	0.4
Science Fiction	0.1	0.4	0.6	0.4	0.1	1.2	0.6
Family	0.3	0.1	5.0	0.4	0.5	3.1	0.3
Fantasy	0.5	0.5	2.9	0.5	0.2	1.9	0.5
Animation	0.6	0.3	3.8	0.2	0.4	1.6	0.3
Mystery	0.7	0.9	0.7	0.6	0.1	2.0	0.1
History	0.0	27.9	1.8	1.1	2.5	2.3	0.9
War	34.6	0.0	0.8	2.3	2.0	0.0	0.0
Music	2.6	1.0	0.0	1.9	6.1	2.3	0.0
Western	2.3	3.6	2.7	0.0	0.0	0.5	0.5
Documentary	5.8	3.7	9.9	0.0	0.0	0.5	0.0
TV Movie	8.3	0.0	5.8	0.8	0.8	0.0	0.0
Foreign	7.3	0.0	0.0	1.8	0.0	0.0	0.0

Here we have conditional probabilities in rows. For example, if we look at Drama movies, 21.8% of them are also Comedies and 25.1% are Thrillers. In other words: P(Comedy | Drama) = 21.8%.

Let's say that we have a Romance movie. What is more likely for that movie to be a Drama or a Horror movie as well? Our common sense guess will be that Drama is more likely. But how much more likely? P(Drama | Romance) = 67.4%, P(Horror | Romance) = 1.6%.

```
In [28]: genres = [genre['name'] for genres in tmdb_movies_df['genres'] for genre i
         print(len(genres), len(tmdb_movies_df))
         np.unique(genres)
```

```
(27633, 11291)
```

```
Out[28]: array([u'Action', u'Adventure', u'Animation', u'Comedy', u'Crime',
                u'Documentary', u'Drama', u'Family', u'Fantasy', u'Foreign',
                u'History', u'Horror', u'Music', u'Mystery', u'Romance',
                u'Science Fiction', u'TV Movie', u'Thriller', u'War', u'Western'],
               dtype='<U15')
```

1.1.8 Additional visualization sketches and EDA with a focus on movie genres

PCA and genres

```
In [3]: import pandas as pd
        import matplotlib.pyplot as plt
        from sklearn.decomposition import PCA
```

```
import numpy as np
import ast
from mpl_toolkits.mplot3d import Axes3D
from collections import OrderedDict
from operator import itemgetter
```

```
In [4]: movie_df112 = pd.read_csv('Milestone_1/tmdb_movies_11291.csv', names = [
            'adult', 'backdrop_path', 'belongs_to_collection', 'budget', 'genres',
            'original_language', 'original_title', 'overview', 'popularity', 'poste
            'production_countries', 'release_date', 'revenue', 'runtime', 'spoken_l
            'video', 'vote_average', 'vote_count',
        ])
```

```
In [5]: def genre_clean(df):
            genres = []
            all_genre = []
            for row in df['genres'].tolist():
                cur_row = []
                for entry in ast.literal_eval(row):
                    cur_row.append(str(entry[u'name']))
                    all_genre.append(str(entry[u'name']))
                genres.append(' '.join(cur_row))
            return (genres, list(set(all_genre)))
        genre_meta = genre_clean(movie_df112)
        movie_df112['genre_vals'] = genre_meta[0]
```

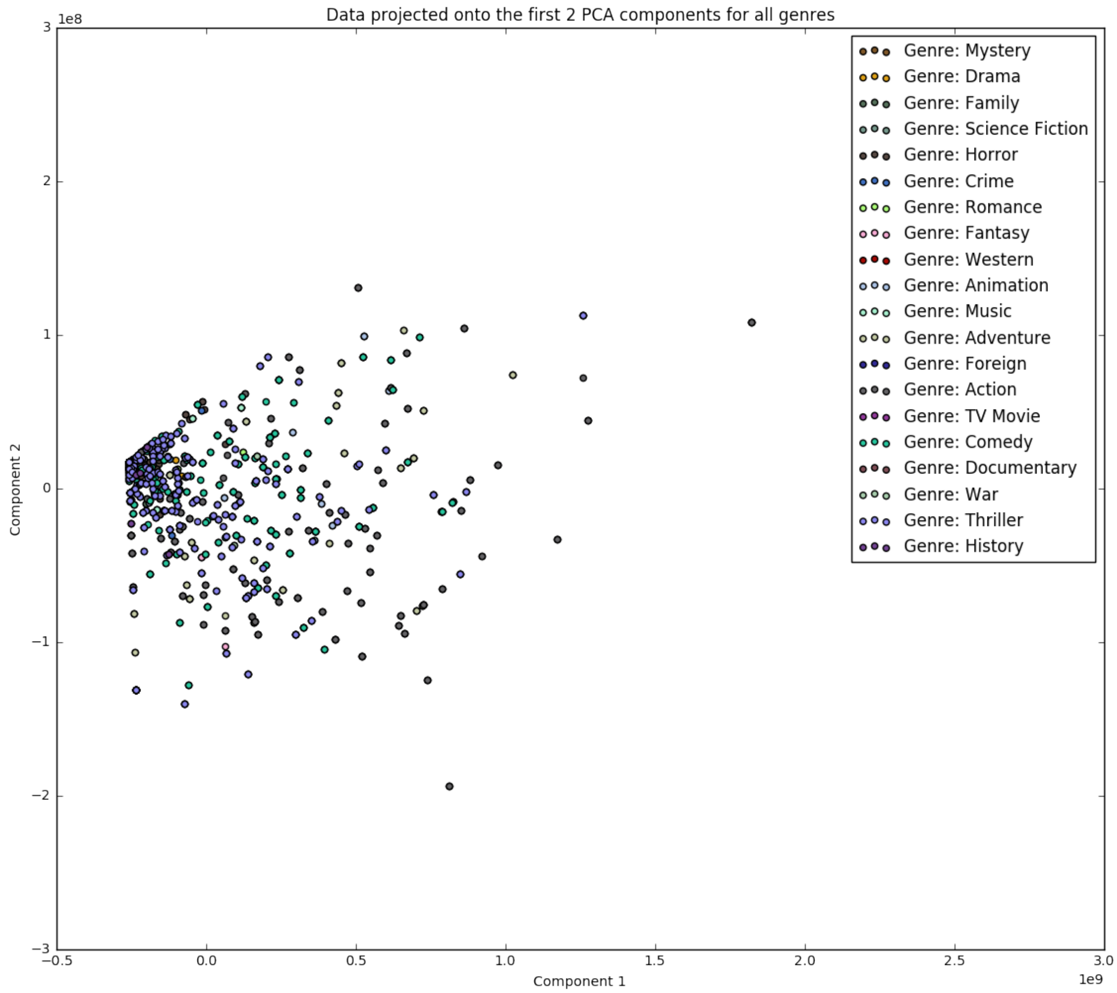
```
In [8]: def pca_and_naclean_temp(df):
            sub = df.dropna()
            X = pd.DataFrame(sub, columns=['budget', 'popularity', 'revenue', 'runt
            pca = PCA(n_components=3)
            X_r = pca.fit(X).transform(X)
            sub.loc[sub.budget > -5, 'pc1'] = X_r[:,0]
            sub.loc[sub.budget > -5, 'pc2'] = X_r[:,1]
            sub.loc[sub.budget > -5, 'pc3'] = X_r[:,2]
            return sub
```

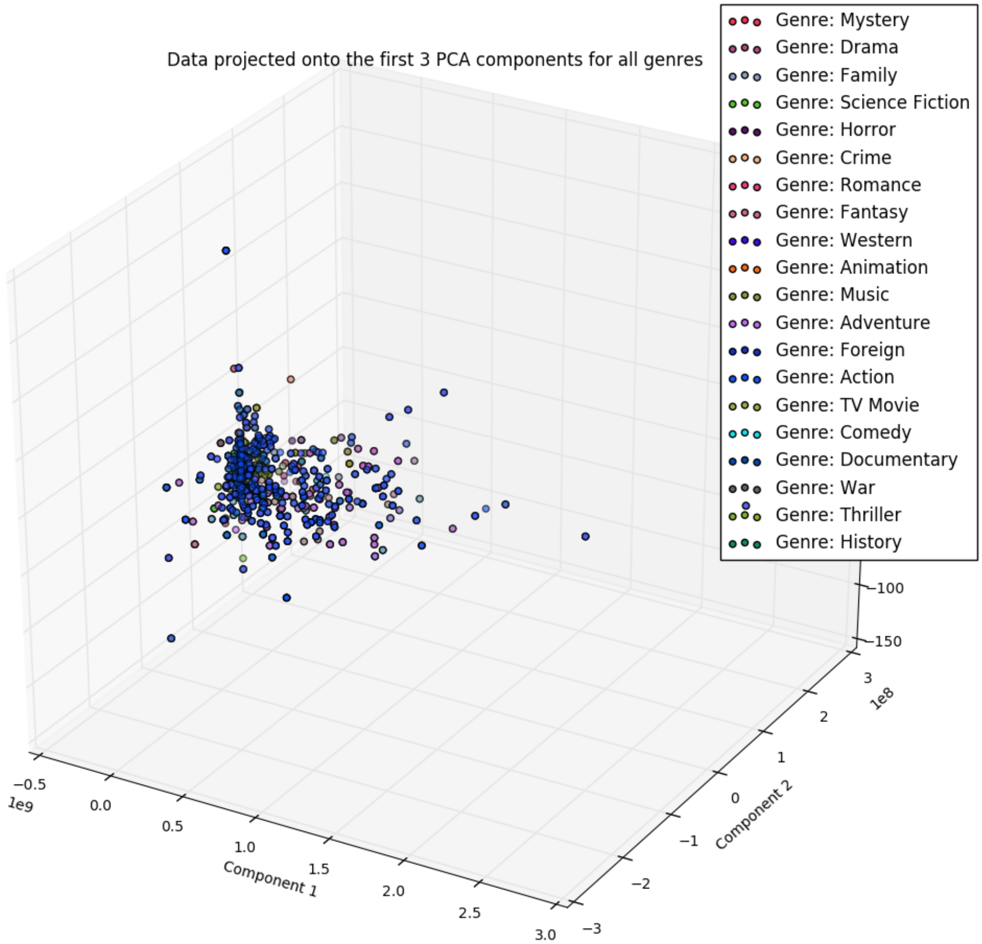
```
In [9]: clean_mov112 = pca_and_naclean_temp(movie_df112)
```

```
In [12]: def plot_pca_3(a_g, df, ds = ''):
             fig = plt.figure(figsize=(30, 12))
             ax1 = fig.add_subplot(1, 2, 1,  projection='3d')
             #for the sake of simplicty, I counted any movie with genre x as having
             for g in a_g:
                 ax1.scatter(df[df['genre_vals'].str.contains(g)]['pc1'], df[df['ge
             ax1.set_xlabel('Component 1')
             ax1.set_ylabel('Component 2')
             ax1.set_zlabel('Component 3')
             ax1.set_title('Data projected onto the first 3 PCA components'+ds)
             ax1.legend()
```

```
        plt.show()

    def plot_pca_2(a_g, df, ds = ''):
        fig = plt.figure(figsize=(30, 12))
        ax1 = fig.add_subplot(1, 2, 1)
        #for the sake of simplicty, I counted any movie with genre x as having
        for g in a_g:
            ax1.scatter(df[df['genre_vals'].str.contains(g)]['pc1'], df[df['ge
        ax1.set_xlabel('Component 1')
        ax1.set_ylabel('Component 2')
        ax1.set_title('Data projected onto the first 2 PCA components'+ds)
        ax1.legend(loc='best')
        plt.show()
```

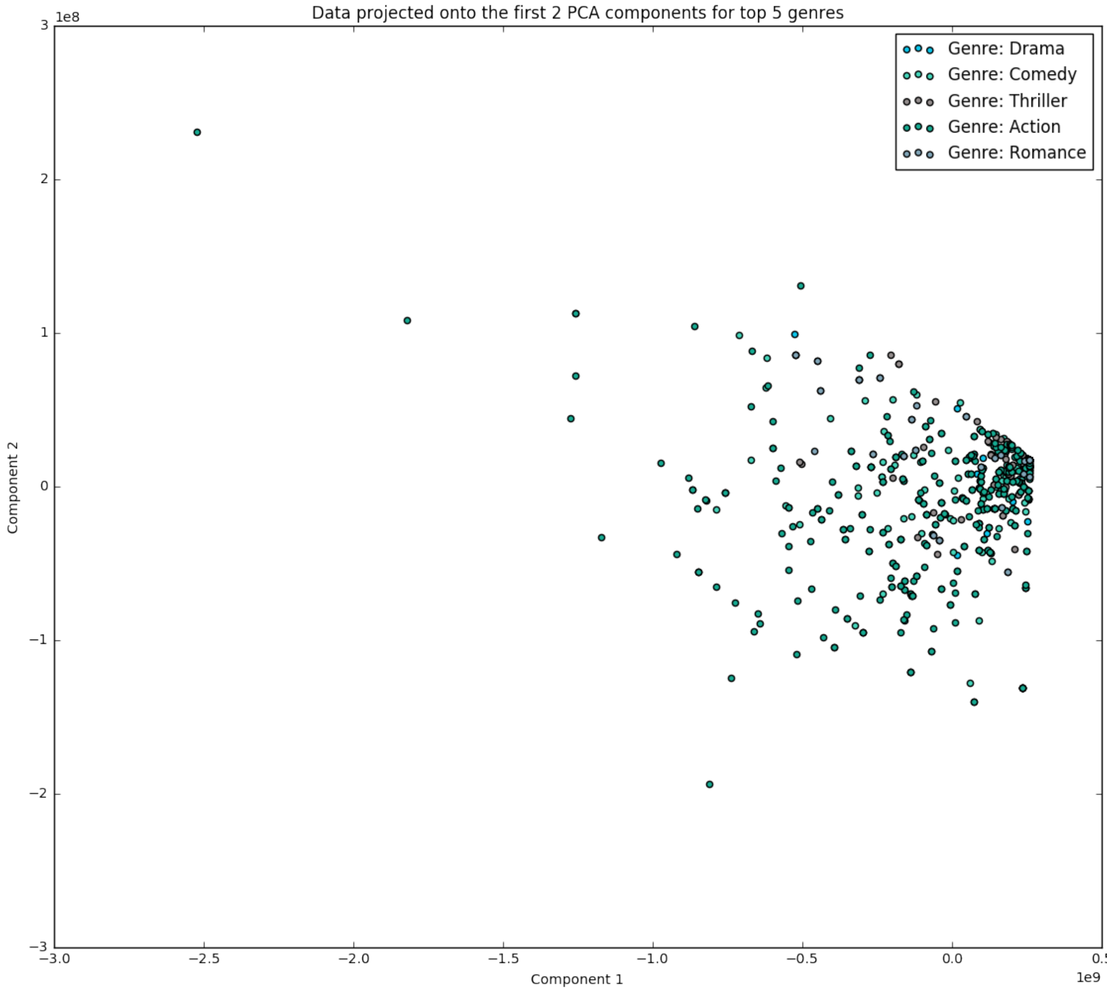
```
In [13]: plot_pca_2(genre_meta[1], clean_mov112, ds = ' for all genres')
         plot_pca_3(genre_meta[1], clean_mov112, ds = ' for all genres')
```

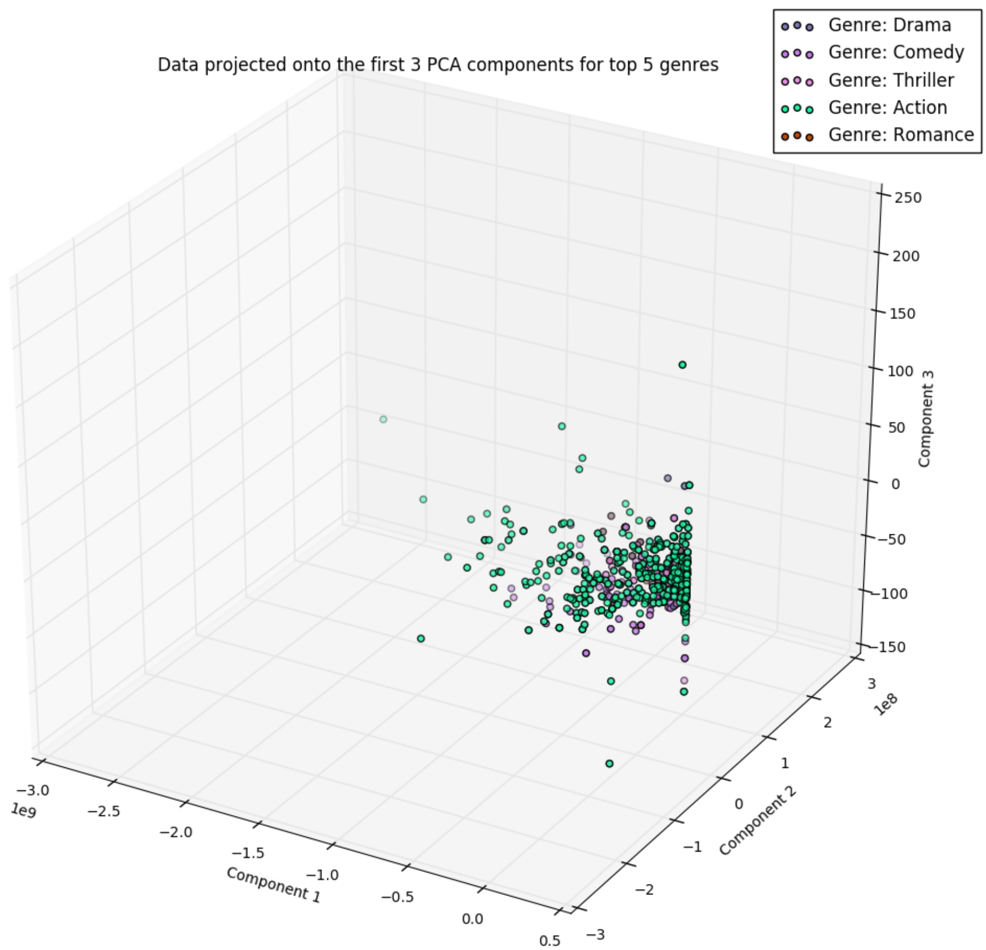




In our datasets current form, we have 5 strong, quantitative predictors: budget, popularity, revenue, runtime, and vote average. We anticipate supplementing these features with external features found from different data sources, but for now, we wanted to visualize how these features map out the movie landscape. By performing a principle component analysis on entire data set and plotting the first two and first 3 PCS (colored by genre), we can see a preliminary distribution as to which genres may be similar (based on clumps) and which movies are themselves similar. The PCA allows us to visualize in 2 and 3 dimensions, the sway of each predictor. However, individual movies share several genres and in this milestone, we have brainstormed ideas as to how to make a single class predictive model. In this plot, the same movie may show up multiple times under different genres.

```
In [36]: plot_pca_2([ 'Drama', 'Comedy', 'Thriller', 'Action', 'Romance'], clean_mo
         plot_pca_3([ 'Drama', 'Comedy', 'Thriller', 'Action', 'Romance'], clean_mo
```


Data projected onto the first 2 PCA components for top 5 genres
1e8
3
2
1
0
−1
−2
−3
Component 2
−3.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1e9
Component 1
Genre: Drama
Genre: Comedy
Genre: Thriller
Genre: Action
Genre: Romance

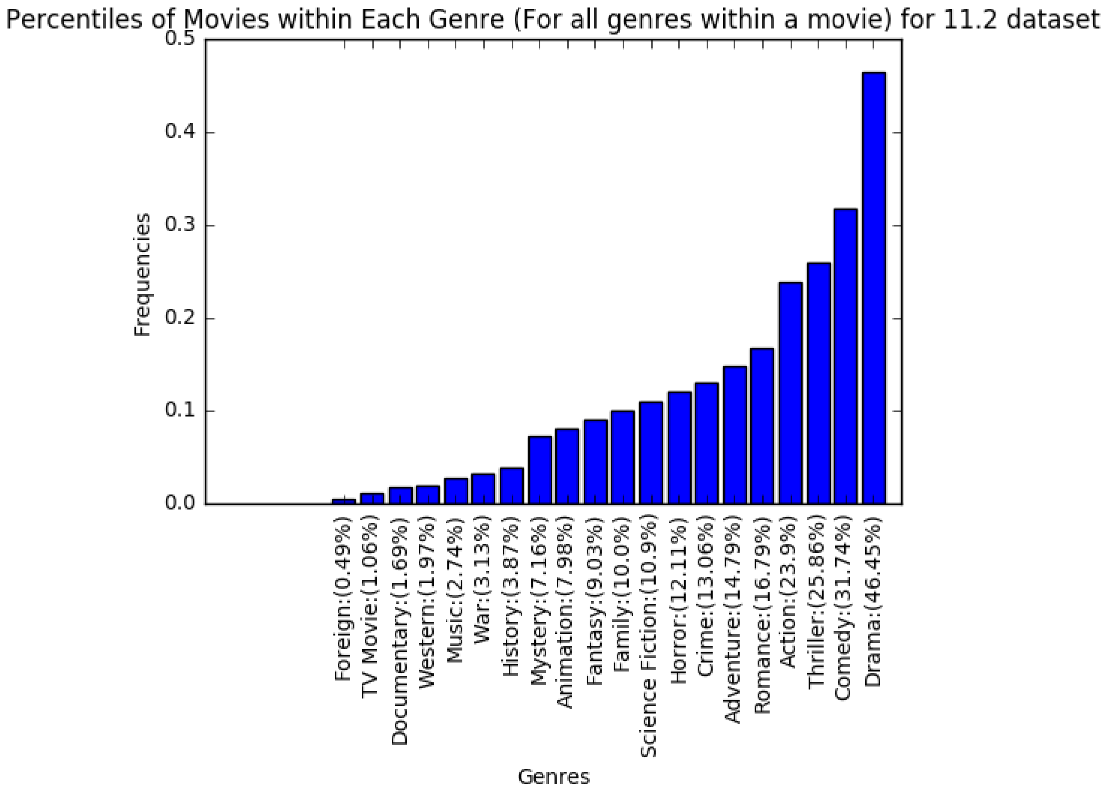


There are a lot of movies in the dataset and a lot of different genres. Each movie also falls under several genres, making visualization and the creation of a classification model trickier. For a better visual, we isolated the top 5 genres and plotted them.

```
In [37]: def create_dic_and_plot(df, every_genre, ds = ''):
             freq_dic = {}
             for genre in every_genre:
                 temp = float(len(df[df['genre_vals'].str.contains(genre)]))/float(
                 key = genre + ':(' + str(round(temp, 4)*100) + '%)'
                 freq_dic[key] = temp
             new = OrderedDict(sorted(freq_dic.items(), key=itemgetter(1)))

             plt.bar(range(len(new)), new.values(), align='center', color= 'blue')
             plt.xticks(range(len(new)), new.keys(), rotation=90)
             plt.ylabel('Frequencies')
             plt.xlabel('Genres')
             plt.title('Percentiles of Movies within Each Genre (For all genres wit
             plt.show()
```

Bar plots for genres

```
In [38]: create_dic_and_plot(movie_df112, genre_meta[1], ' for 11.2 dataset')
```



We also wanted to understand which genres are most used in classification. From our current dataset, we found Drama to be the most used label, followed by Comedy, Thriller, Action and Romance. Again, at this point in time, each movie has several distinct genre labels, hence the percentages adding to more than 100.

Influential positive and negative terms for different genres

```
In [39]: import ast
         import operator
         import numpy as np
         import pandas as pd
         import matplotlib
         import matplotlib.pyplot as plt
         from sklearn.cross_validation import KFold
         from sklearn.linear_model import LogisticRegression as LogReg
         from sklearn.feature_extraction.text import CountVectorizer
         import nltk.data
         from nltk.corpus import stopwords
         %matplotlib inline
         nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to C:\Windows\ServiceProfile
[nltk_data]     s\LocalService\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
Out[39]: True
```

Prints most influential positive and negative terms from the movies overview. Later we could extend this to analyse movies scripts.

```
In [40]: def getY(df, genre_name):
             def hasGenre(genres, genre_name):
                 for genre in genres:
                     if(genre['name'] == genre_name):
                         return 1.0
                 return 0.0
             return np.array([hasGenre(genre_list, genre_name) for genre_list in df

         def cross_validate(x, y, folds, reg_params):
             kf = KFold(x.shape[0], n_folds=folds)
             cv_score = np.zeros(reg_params.size)
             for i, c in enumerate(reg_params):
                 reg = LogReg(penalty='l1', C = c)
                 score_sum = 0.0
                 for train_index, test_index in kf:
                     reg.fit(x[train_index], y[train_index])
                     score_sum += reg.score(x[test_index], y[test_index])
                 cv_score[i] = score_sum/float(folds)
             return cv_score
```

```
In [41]: def getInfluentialTerms(x, genre_name):
             # get labels
             y = getY(tmdb_movies_with_overview, genre_name)
             # cross-validate for best regularization parameter
             all_c = np.power(10., range(-7, 8))
             cv_scores = cross_validate(x, y, 10, all_c)
             best_c = all_c[np.argmax(cv_scores)]
             # fit logistic regression
             logReg = LogReg(penalty='l1', C = best_c)
             logReg.fit(x, y)
             coef = logReg.coef_[0]
             # top and bottom percentiles
             top_1 = [coef >= np.percentile(coef, 99)]
             bottom_1 = [coef <= np.percentile(coef, 1)]
             return top_1, bottom_1
```

```
In [42]: tmdb_movies_df = load_tmdb_movies()
         has_overview = ~tmdb_movies_df['overview'].isnull()
         tmdb_movies_with_overview = tmdb_movies_df[has_overview]
```

```
In [43]: vectorizer = CountVectorizer(
             stop_words = stopwords.words("english"),
             token_pattern = '[a-zA-Z]+[0-9]*',
             max_df = 0.9,
             min_df = 5,
             dtype=np.float32 )
         x = vectorizer.fit_transform(tmdb_movies_with_overview['overview'].values)
         print 'predictor matrix shape:', x.shape
```

```
predictor matrix shape: (11145L, 8773L)
```

```
In [44]: feature_names = np.array(vectorizer.get_feature_names())
         feature_names
```

```
Out[44]: array([u'aaron', u'abandon', u'abandoned', ..., u'zone', u'zoo', u'zooey']
               dtype='<U17')
```

```
In [45]: def printInfluentialTerms(genre_name):
             top_1, bottom_1 = getInfluentialTerms(x, genre_name)
             print genre_name, 'most influential positive terms:', feature_names[to
             print genre_name, 'most influential negative terms:', feature_names[bo
             return feature_names[top_1], feature_names[bottom_1]
```

```
In [46]: drama_top, drama_bottom = printInfluentialTerms('Drama')
```

```
Drama most influential positive terms: [u'afterlife' u'aged' u'alcoholic' u'allied'
 u'astronaut' u'ballet' u'banned' u'betrothed' u'blizzard' u'boxer'
 u'brien' u'build' u'capsule' u'chronicle' u'colonies' u'connections'
 u'constant' u'crushed' u'cycle' u'depression' u'detention' u'difficult'
 u'disturbed' u'drama' u'dramatic' u'elaborate' u'emotionally' u'enigmatic'
 u'erin' u'finest' u'forty' u'grief' u'guns' u'halt' u'hardened' u'heat'
 u'holly' u'holocaust' u'idealistic' u'incriminating' u'industrial'
 u'interpretation' u'islamic' u'janitor' u'josh' u'laden' u'laundry'
 u'loan' u'loveless' u'luis' u'luna' u'maid' u'miracle' u'morgan' u'mute'
 u'orphans' u'painter' u'patrick' u'performs' u'physician' u'poet'
 u'primary' u'prosecutor' u'pursue' u'raped' u'rehab' u'repercussions'
 u'roads' u'ruth' u'sail' u'shoes' u'skill' u'smitten' u'spying'
 u'stockholm' u'tempted' u'tennessee' u'therapist' u'tragedy' u'trevor'
 u'trucker' u'unorthodox' u'vast' u'vision' u'wales' u'wells']
Drama most influential negative terms: [u'access' u'acting' u'active' u'aka' u'ange
 u'arsenal' u'awake' u'barbie' u'beast' u'buffalo' u'bumbling' u'cameron'
 u'carries' u'chasing' u'chicken' u'childbirth' u'concert' u'controls'
 u'cookie' u'crazed' u'curse' u'damaged' u'documentary' u'downed' u'edited'
 u'elude' u'emil' u'entity' u'fifty' u'foil' u'footage' u'furious' u'gates'
 u'goat' u'gotten' u'halloween' u'hapless' u'happenings' u'horde'
 u'importance' u'includes' u'inhabitants' u'kai' u'kidnappers' u'latter'
 u'lifeguard' u'losers' u'maniac' u'mickey' u'mischievous' u'model'
 u'murphy' u'operates' u'ops' u'pal' u'paranormal' u'paying' u'pirate'
```

```
 u'pirates' u'possessed' u'preserve' u'resorts' u'respective' u'revelation'
 u'senses' u'ships' u'smugglers' u'snake' u'sometime' u'sophisticated'
 u'species' u'spoof' u'stalked' u'stormy' u'talents' u'unaware' u'undergo'
 u'unlucky' u'uproarious' u'vacationing' u'villain' u'villains'
 u'volunteers' u'wildly' u'witch' u'zombie']
```

```
In [47]: comedy_top, comedy_bottom =  printInfluentialTerms('Comedy')
```

```
Comedy most influential positive terms: [u'abandonment' u'allan' u'annoying' u'appl
 u'attitudes' u'awkward' u'axe' u'brainy' u'bumbling' u'bunch' u'celebrity'
 u'cheap' u'cheerleader' u'chocolate' u'citizen' u'clinic' u'clouseau'
 u'clueless' u'comedic' u'comedy' u'comfortable' u'comic' u'confusion'
 u'cruchot' u'curmudgeonly' u'dating' u'examine' u'fake' u'fist'
 u'fraternity' u'furious' u'gary' u'grandpa' u'hapless' u'hilariously'
 u'horny' u'humor' u'hypochondriac' u'inadvertently' u'incompetent'
 u'intergalactic' u'inventor' u'irreverent' u'jokes' u'laced' u'leopold'
 u'lifeguard' u'lotus' u'mascot' u'mistakenly' u'mundane' u'nephew'
 u'patriarch' u'photos' u'positions' u'pretends' u'quirky' u'regina'
 u'relocated' u'reported' u'resolve' u'resourceful' u'retrieving' u'roy'
 u'sentenced' u'shallow' u'shrek' u'sixth' u'spend' u'spoof' u'stable'
 u'stripper' u'sure' u'surprisingly' u'swimming' u'tech' u'temporarily'
 u'thus' u'underway' u'unfortunate' u'uptight' u'velma' u'wedding'
 u'werewolves' u'whatever' u'zombie']
Comedy most influential negative terms: [u'affect' u'allied' u'anne' u'ash' u'augus
 u'blame' u'brutally' u'carried' u'cattle' u'celebrate' u'china' u'chloe'
 u'chronicle' u'clara' u'combination' u'deeper' u'defeating' u'designer'
 u'devastating' u'diagnosed' u'dinosaur' u'drama' u'enemies' u'enforcement'
 u'equally' u'escaping' u'facility' u'fairies' u'forbidden' u'gotham'
 u'gradually' u'guido' u'ha' u'hangs' u'hop' u'horrifying' u'humanity'
 u'hunting' u'injustice' u'inmate' u'justine' u'khan' u'language' u'lion'
 u'loyalty' u'malevolent' u'marine' u'mechanic' u'milo' u'motel' u'mouse'
 u'nicknamed' u'nights' u'promise' u'psychological' u'rabbit' u'racial'
 u'rape' u'reason' u'receives' u'rescued' u'rights' u'rising' u'robert'
 u'sadistic' u'secluded' u'shared' u'shocking' u'skin' u'sprawling'
 u'stability' u'stagecoach' u'staying' u'stewart' u'suspicion'
 u'suspicions' u'thriller' u'tomboy' u'torment' u'towards' u'trafficking'
 u'tragedy' u'tragic' u'unleashes' u'violence']
```

```
In [48]: thriller_top, thriller_bottom =  printInfluentialTerms('Thriller')
```

```
Thriller most influential positive terms: [u'accident' u'agent' u'alive' u'apartmen
 u'car' u'cia' u'computer' u'conspiracy' u'cop' u'crime' u'criminal'
 u'criminals' u'dangerous' u'dark' u'dead' u'deadly' u'death' u'deep'
 u'detective' u'discover' u'discovers' u'drug' u'escape' u'events' u'ex'
 u'fear' u'former' u'goes' u'group' u'horror' u'hospital' u'hostage'
 u'house' u'identity' u'job' u'john' u'kidnapped' u'kill' u'killer'
 u'killers' u'killing' u'mark' u'may' u'mind' u'mission' u'mob' u'murder'
```

```
 u'murdered' u'murders' u'mysterious' u'mystery' u'nuclear' u'officer'
 u'past' u'phone' u'police' u'prey' u'psychiatrist' u'rachel' u'remote'
 u'revenge' u'run' u'sam' u'secret' u'security' u'seemingly' u'seems'
 u'serial' u'sinister' u'soldier' u'something' u'soon' u'supernatural'
 u'survival' u'taken' u'target' u'terror' u'terrorist' u'thriller'
 u'trapped' u'u' u'uncover' u'underworld' u'unknown' u'violent']
Thriller most influential negative terms: [u'accidentally' u'adventure' u'adventure
 u'army' u'back' u'band' u'best' u'boy' u'captain' u'christmas' u'coach'
 u'college' u'comedy' u'competition' u'country' u'dance' u'day' u'de'
 u'decides' u'documentary' u'dreams' u'epic' u'even' u'evil' u'family'
 u'father' u'film' u'first' u'french' u'friend' u'friends' u'friendship'
 u'get' u'giant' u'good' u'great' u'hero' u'journey' u'king' u'land'
 u'life' u'little' u'live' u'living' u'long' u'love' u'magic' u'many'
 u'marriage' u'master' u'meet' u'meets' u'movie' u'music' u'named' u'never'
 u'new' u'old' u'parents' u'prince' u'princess' u'queen' u'quest'
 u'relationship' u'romantic' u'school' u'sex' u'show' u'star' u'stars'
 u'story' u'summer' u'super' u'th' u'three' u'tries' u'two' u'village'
 u'war' u'wedding' u'well' u'women' u'work' u'world' u'year']
```

```
In [49]: action_top, action_bottom = printInfluentialTerms('Action')
```

```
Action most influential positive terms: [u'advice' u'airborne' u'apes' u'archaeolog
 u'audition' u'avenger' u'batman' u'benevolent' u'bud' u'bumbling' u'cache'
 u'caine' u'captors' u'celebrities' u'civilization' u'clayton' u'clone'
 u'commando' u'corporation' u'criminals' u'detailing' u'directly' u'donor'
 u'emerge' u'enforcement' u'enterprise' u'exceptional' u'exiled' u'fail'
 u'fearsome' u'fighters' u'firefighter' u'flynn' u'hacker' u'hawk'
 u'hitman' u'internal' u'islands' u'items' u'jai' u'kgb' u'knife'
 u'loyalty' u'luc' u'manuscript' u'martial' u'millennium' u'missile'
 u'musketeers' u'nevada' u'newest' u'ninjas' u'operatives' u'parker'
 u'paths' u'patrol' u'province' u'robots' u'ruined' u'ruthless' u'safety'
 u'samurai' u'savage' u'scarce' u'sinbad' u'skilled' u'slaves' u'smuggling'
 u'sniper' u'speed' u'strategy' u'struck' u'superhero' u'superman'
 u'superpowers' u'swiftly' u'target' u'terrorists' u'transport' u'trucker'
 u'uss' u'viciously' u'vigilante' u'warriors' u'wrestler' u'wright']
Action most influential negative terms: [u'actual' u'apple' u'aristocratic' u'arms'
 u'banished' u'buy' u'campbell' u'canine' u'caring' u'catholic' u'changed'
 u'civilians' u'cold' u'comedic' u'connection' u'consequences' u'current'
 u'dance' u'danish' u'deserts' u'discoveries' u'documentary' u'dollar'
 u'doraemon' u'drunk' u'episode' u'erupt' u'exciting' u'feeds' u'festival'
 u'filmmakers' u'fled' u'fresh' u'friendly' u'fun' u'funny' u'ghosts'
 u'grandmother' u'hilarious' u'hitchcock' u'investigating' u'irene'
 u'janitor' u'jennifer' u'launched' u'lawrence' u'legions' u'lisa' u'lose'
 u'maiden' u'manipulative' u'marco' u'misfit' u'motel' u'n' u'neighbor'
 u'outrageous' u'pals' u'patient' u'polish' u'prepared' u'probe'
 u'promotion' u'psychic' u'radical' u'ralph' u'raped' u'rocky' u'rome'
 u'ruby' u'scooby' u'sight' u'sold' u'spreading' u'spree' u'surrounded'
```

```
u'television' u'tormented' u'tyler' u'unsuccessful' u'waiting' u'waitress'
u'week' u'yard' u'yellow' u'yi']
```

Genres 'Drama' and 'Thriller' are quite often assigned to same movie bit the most positive influential terms for them do not intersect!

```
In [50]: list(set(drama_top) & set (thriller_top))
```

```
Out[50]: []
```

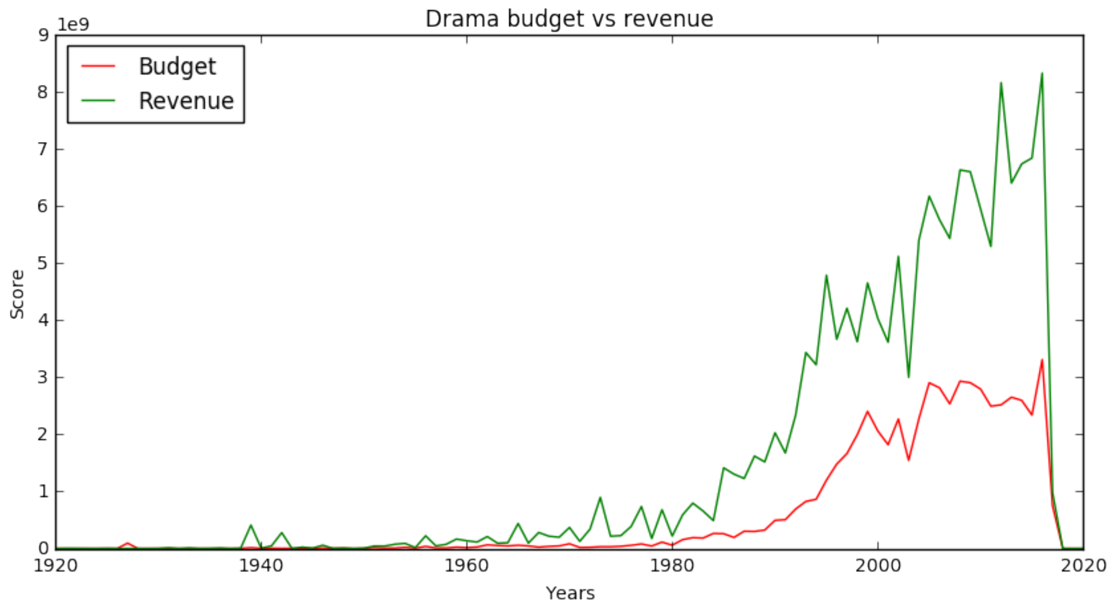
Budget and revenue for genres

```
In [51]: tmdb_movies_df['release_date']  = pd.to_datetime(tmdb_movies_df['release_c
         tmdb_movies_df['year'] = tmdb_movies_df['release_date'].map(lambda x: x.ye
```

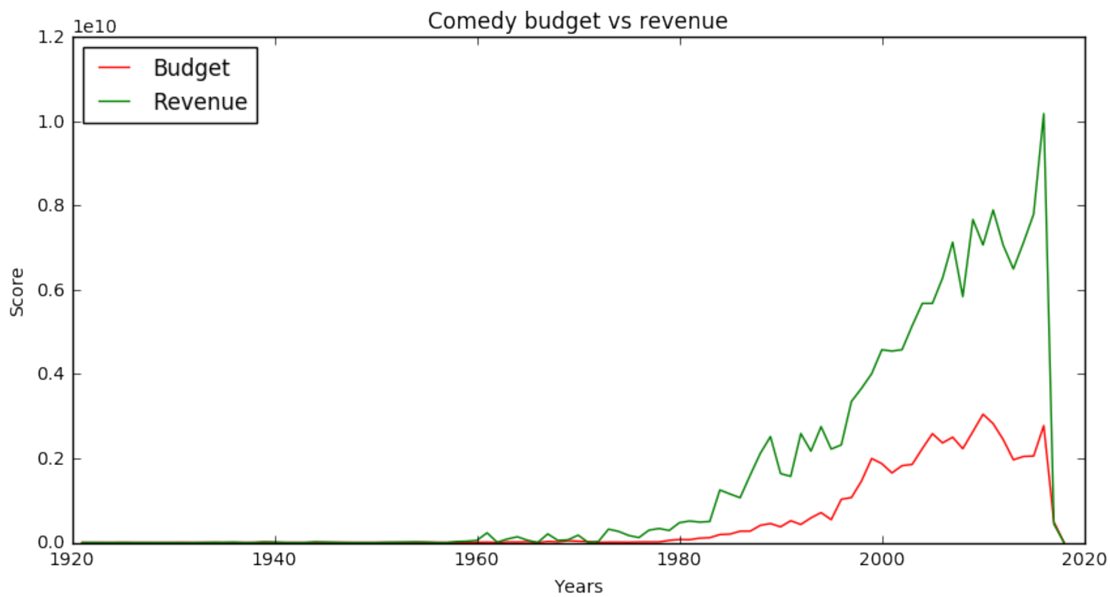
```
In [52]: def financeByGenre(df, genre_name):
             def hasGenre(genres, genre_name):
                 for genre in genres:
                     if(genre['name'] == genre_name):
                         return True
                 return False
             d = df[[hasGenre(genre_list, genre_name) for genre_list in df['genres'
             return d.groupby(['year'])[["budget", "revenue"]].sum()
```

```
In [53]: def plotFincnceByGenre(df, genre_name):
             d = financeByGenre(df, genre_name)
             fig = plt.figure(figsize=(10, 5))
             ax = fig.add_subplot(111)
             ax.plot(d.index, d['budget'], c='r', label = 'Budget')
             ax.plot(d.index, d['revenue'], c='g', label = 'Revenue')
             ax.set_xlabel('Years')
             ax.set_ylabel('Score')
             ax.set_title(genre_name  + ' budget vs revenue')
             ax.legend(loc = 'best')
             plt.ticklabel_format(useOffset=False)
```

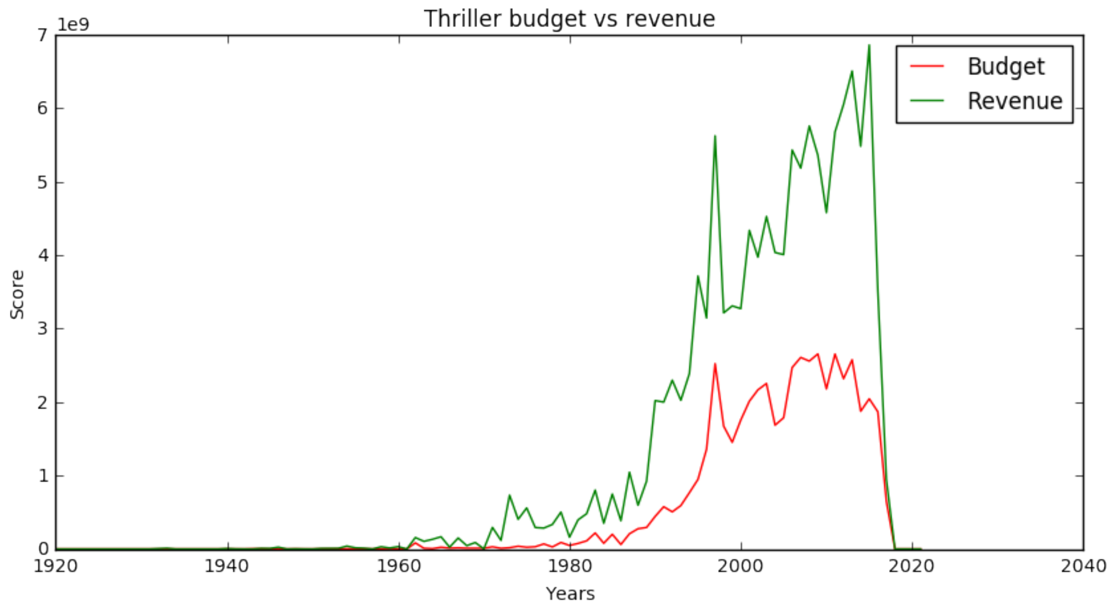
```
In [54]: plotFincnceByGenre(tmdb_movies_df, 'Drama')
```



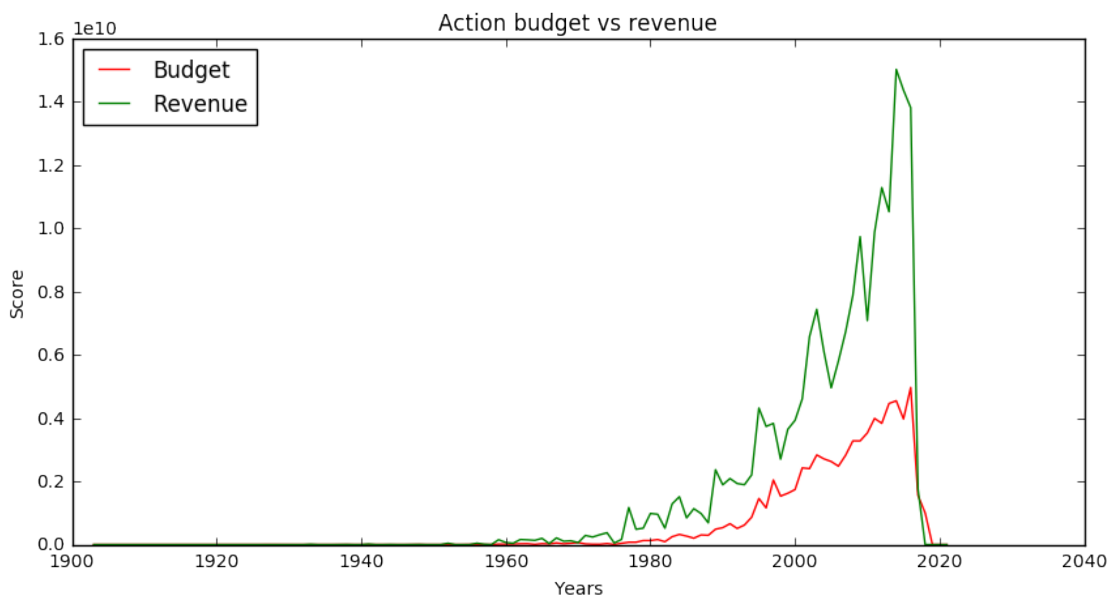

```
In [55]: plotFincnceByGenre(tmdb_movies_df, 'Comedy')
```



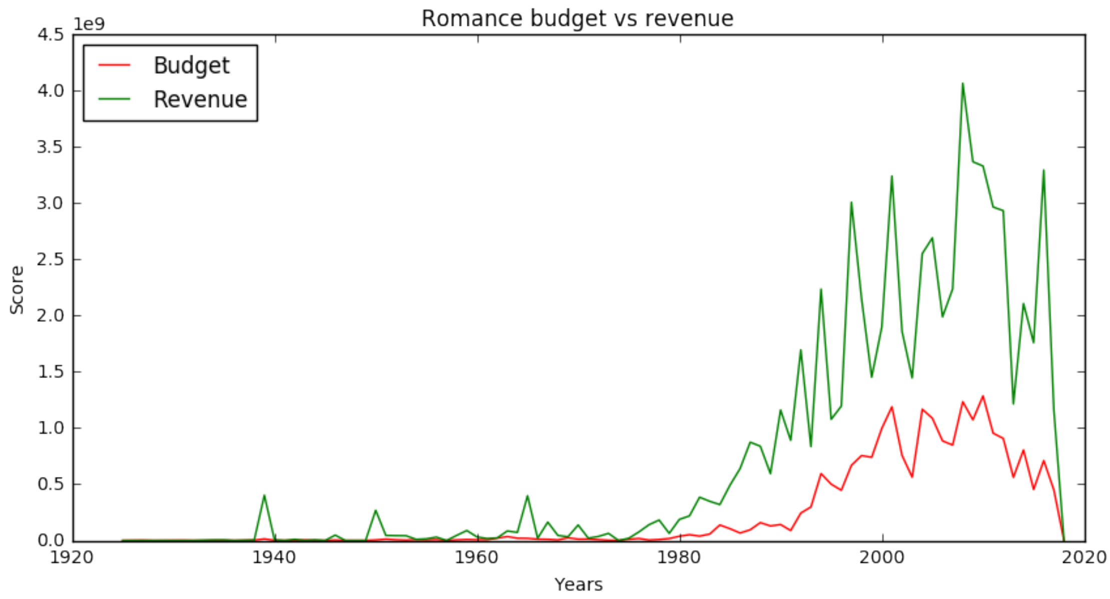

```
In [56]: plotFincnceByGenre(tmdb_movies_df, 'Thriller')
```



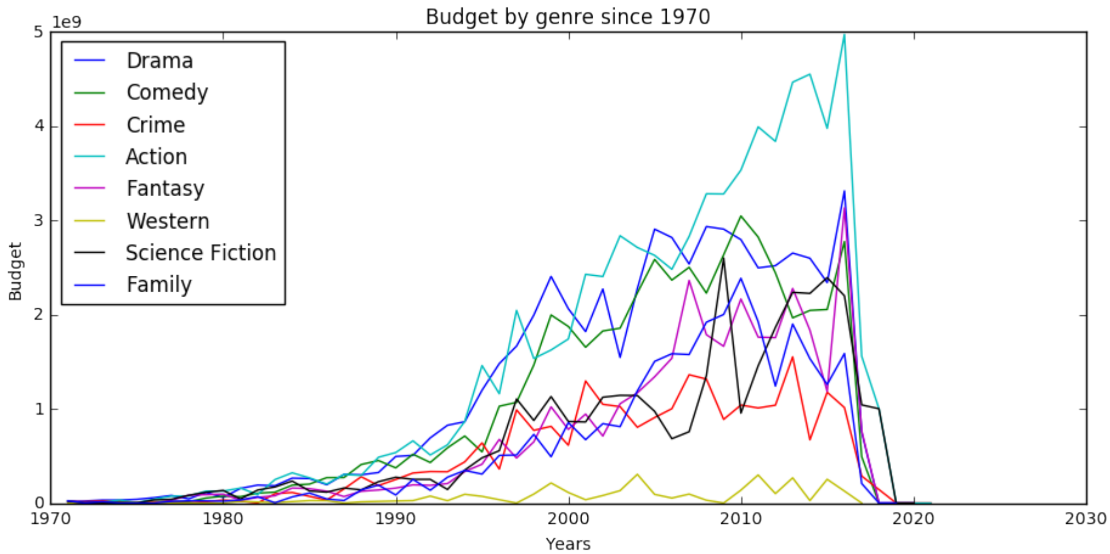

```
In [57]: plotFincnceByGenre(tmdb_movies_df, 'Action')
```




```
In [58]: plotFincnceByGenre(tmdb_movies_df, 'Romance')
```




```
In [59]: genre_names = ['Drama', 'Comedy', 'Crime', 'Action', 'Fantasy', 'Western',

In [60]: fig = plt.figure(figsize=(11, 5))
         ax = fig.add_subplot(111)

         for genre_name in genre_names:
             d = financeByGenre(tmdb_movies_df[tmdb_movies_df['year'] > 1970], genr
             ax.plot(d.index, d['budget'], label = genre_name)
             ax.legend(loc = 'best')

         ax.set_xlabel('Years')
         ax.set_ylabel('Budget')
         ax.set_title('Budget by genre since 1970')

Out[60]: <matplotlib.text.Text at 0xcfc7588>
```



1.1.9 A list of questions you could answer with this and related data. Get creative here!

Based on our EDA, we could answer highly predictive questions such as, how much money do you think movie X will make in its box office given input such as genre, cost of production, reviews, review texts, length of film and the rest of our current predictor set. We could generate a confidence interval that estimates upper and lower bounds for box office revenue.

After performing some initial textual analysis on the small description denoting an overview of the movie, we were able to see which genres used which positive and negative words more often. These overviews are relatively small, so using a text as a predictor set may yield little overlap between movie descriptions. However, as this is just an initial analysis, we are able to scale this kind of analysis to entire reviews and possibly even scripts. Because each movie will have a large set of reviews or a long script, we anticipate a classification analysis based on words would be feasible and powerful. Using this line of thinking, we would be able to take in movie reviews and scripts to predict genre and possibly financial data as described previously.

Although we haven't setup an initial model for image classification yet, we see this dataset as a powerful means to understanding more about a movie using just its poster. We showed earlier that we can get movie posters from this dataset. Then, after performing a dimensionality reduction like PCA, we believe a model could be trained to correctly classify movie genres based on their poster alone. We are fortunate that we have an extremely large dataset and poster movies are almost always the same dimensions (no loss in quality for standardizing). We anticipate that these two properties of the dataset will help with the fact that movie posters are incredibly diverse, implying we would need many principle components to capture a significant proportion of the variance among movie posters. From there, a model could take in a movie poster and predict genre.

The most powerful model isn't based on any singular predictor set. Instead, we would consider a methodology that takes in the dataset in full. From there, it is our hope that our models could complement each other where one model is more certain and another is less certain. Given such an extensive dataset and diverse predictors, we anticipate a strong predictive model.