Milestone_2_Submission_NewData

April 12, 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 Milestone 2: Assembling training data, due Wednesday, April 12, 2017

We are aware that you have little time this week, due to the midterm. So this milestone a bit easier to achieve than the others. The goal for this week is to prepare the data for the modeling phase of the project. You should end up with a typical data setup of training data X and data labels Y.

The exact form of X and Y depends on the ideas you had previously. In general though Y should involve the genre of a movie, and X the features you want to include to predict the genre. Remember from the lecture that more features does not necessarily equal better prediction performance. Use your application knowledge and the insight you gathered from your genre pair analysis and additional EDA to design Y. Do you want to include all genres? Are there genres that you assume to be easier to separate than others? Are there genres that could be grouped together? There is no one right answer here. We are looking for your insight, so be sure to describe your decision process in your notebook.

In preparation for the deep learning part we strongly encourage you to have two sets of training data X, one with the metadata and one with the movie posters. Make sure to have a common key, like the movie ID, to be able to link the two sets together. Also be mindful of the data rate when you obtain the posters. Time your requests and choose which poster resolution you need. In most cases w500 should be sufficient, and probably a lower resolution will be fine.

The notebook to submit this week should at least include:

- Discussion about the imbalanced nature of the data and how you want to address it
- Description of your data
- What does your choice of Y look like?
- Which features do you choose for X and why?
- How do you sample your data, how many samples, and why?

Important: You do not need to upload the data itself to Canvas.

1.0.2 Location of the uploaded data

The project repository: https://github.com/adubitskiy/cs109b
The Milestone 2 notebook: https://github.com/adubitskiy/cs109b/blob/master/Milestone_2_Submission_N
Link to the Google Drive folder: https://drive.google.com/open?id=0B9PSivXSSQOTQWY2X0kyUTBNOFU
We have 3 files there: imdb_info.pickle, tmdb_info.pickle and posters.zip

In [21]:
```
import time
import random
import cPickle
import urllib
import csv
import collections
from IPython.display import Image
from IPython.core.display import HTML
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MultiLabelBinarizer
```

In [2]:
```
def load_part(file_name):
    with open(file_name, 'rb') as handle:
        return cPickle.load(handle)
```

In [3]:
```
tmdb_dict = load_part('data/tmdb_info.pickle')
print 'Loaded ' + str(len(tmdb_dict)) + ' TMDB movies'
```

```
Loaded 58829 TMDB movies
```

In [4]:
```
labels = map(lambda x: [g['name'] for g in x], [movie.genres for movie in t
mlb = MultiLabelBinarizer()
label_df = pd.DataFrame(mlb.fit_transform(labels))
label_df.columns = mlb.classes_
```

In [5]:
```
imdb_dict = load_part('data/imdb_info.pickle')
print 'Loaded ' + str(len(imdb_dict)) + ' IMDB movies'
```

```
Loaded 15000 IMDB movies
```

1.0.3 Discussion about the imbalanced nature of the data and how you want to address it

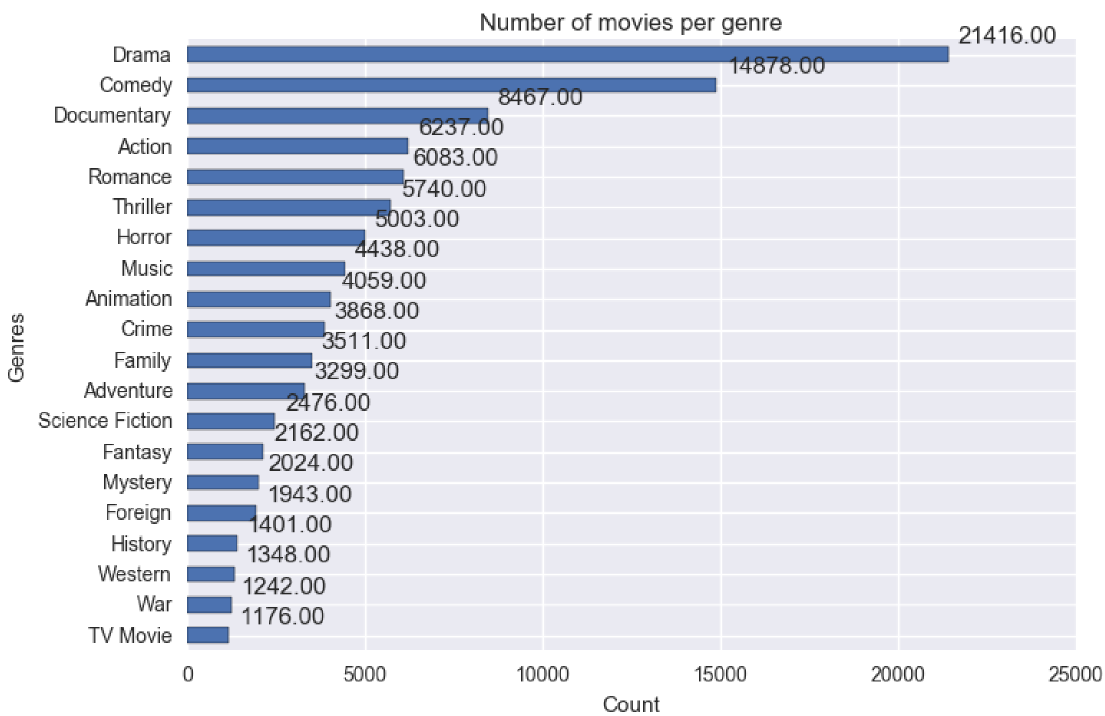
In [6]:
```
label_number_df = pd.DataFrame({'cnt' : label_df.sum(axis = 0)})
label_number_df
```

Out[6]:

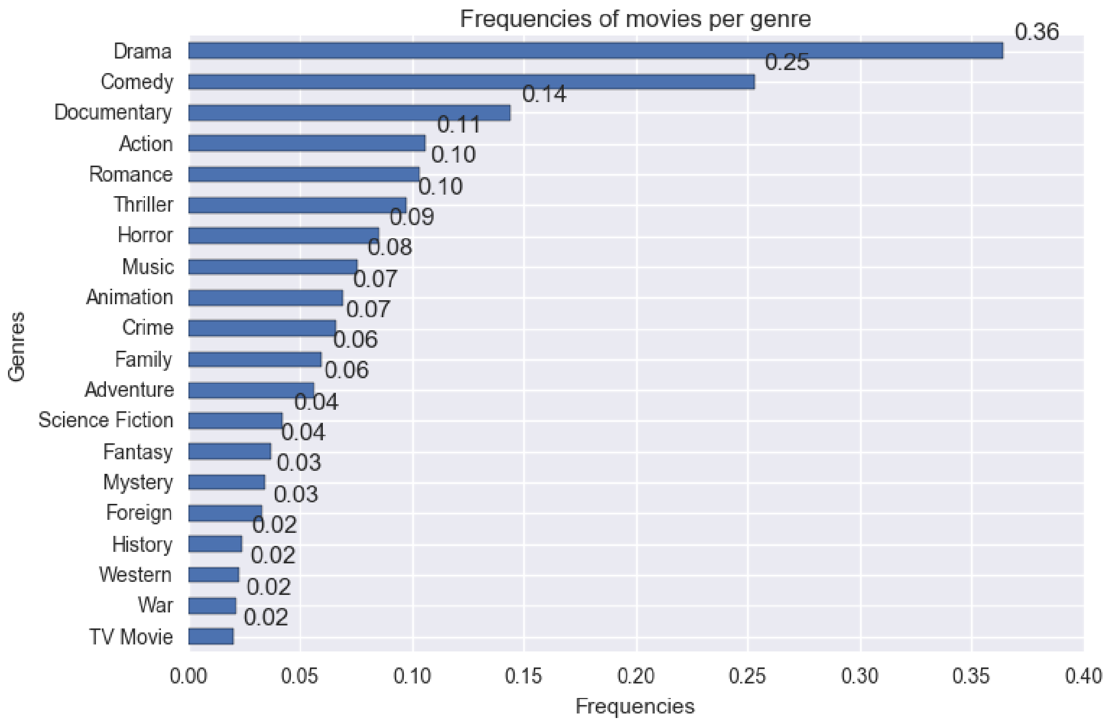
	cnt
Action	6237
Adventure	3299

```
Animation          4059
Comedy            14878
Crime              3868
Documentary        8467
Drama             21416
Family             3511
Fantasy            2162
Foreign            1943
History            1401
Horror             5003
Music              4438
Mystery            2024
Romance            6083
Science Fiction    2476
TV Movie           1176
Thriller           5740
War                1242
Western            1348
```

```
In [7]: ax = label_number_df.sort_values('cnt')['cnt'].plot(kind="barh", title = 'N
        for p in ax.patches:
            ax.annotate(
                "%.2f" % p.get_width(), (p.get_x() + p.get_width(), p.get_y()), xyt
        plt.xlabel('Count')
        plt.ylabel('Genres')
        plt.show()
```



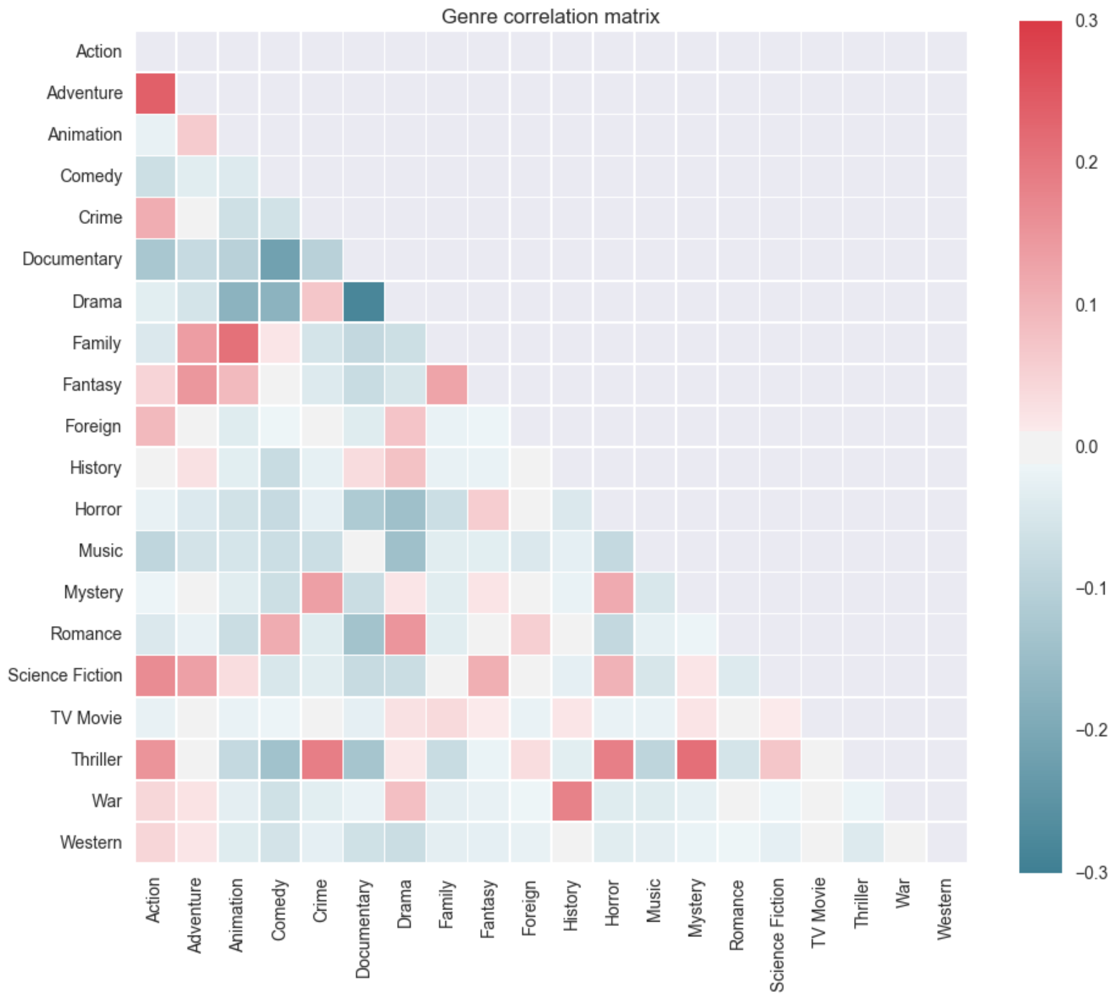
```
In [8]: label_number_df['ratio'] = label_number_df.divide(len(tmdb_dict))
        ax = label_number_df.sort_values('ratio')['ratio'].plot(kind="barh", title
        for p in ax.patches:
            ax.annotate(
                "%.2f" % p.get_width(), (p.get_x() + p.get_width(), p.get_y()), xyt
        plt.xlabel('Frequencies')
        plt.ylabel('Genres')
        plt.show()
```



As one can see, drama is used as a label on a whopping 36% of movie labels. We also got unexpectedly high number of documentaries.

```
In [9]: # Compute the correlation matrix
        corr = label_df.corr()
        # Generate a mask for the upper triangle
        mask = np.zeros_like(corr, dtype=np.bool)
        mask[np.triu_indices_from(mask)] = True
        # Set up the matplotlib figure
        f, ax = plt.subplots(figsize=(11, 9))
        ax.set_title('Genre correlation matrix')
        # Generate a custom diverging colormap
        cmap = sns.diverging_palette(220, 10, as_cmap=True)
```

```
# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, square=True, linewidths=.5
plt.show()
```



Some of the genres are likely to be assigned together: Action & Adeventure, Family & Comedy.

```
In [10]: genre_comb = map(
             lambda x: ",".join( sorted(set([ g['name'] for g in x]) )),
             [movie.genres for movie in tmdb_dict.values()]
         )
         genre_comb_df = pd.DataFrame({ 'genres' : genre_comb})
         print 'Number of different genre combinations in our dataset:'
         genre_comb_df['genres'].value_counts()
```

```
Number of different genre combinations in our dataset:
```

```
Out[10]: Drama
         Documentary
```

Comedy
Music
Animation
Horror
Comedy,Drama
Drama,Romance
Comedy,Romance
Action
Western
Thriller
Crime,Drama
Comedy,Drama,Romance
Drama,Thriller
Documentary,Music
Family
Horror,Thriller
Romance
Action,Drama
Crime
Animation,Family
Comedy,Horror
Drama,Foreign
Drama,History
Drama,War
Drama,Family
Action,Comedy
Science Fiction
Comedy,Family

Action,Adventure,Animation,Comedy,Family,Fantasy,Science Fiction,Thriller
Animation,Comedy,Romance,Science Fiction
Adventure,Music,Mystery,Thriller
Action,Animation,Fantasy,Horror,Science Fiction,Thriller
Comedy,Music,Mystery,Romance,TV Movie
Action,Fantasy,Horror,Thriller
Foreign,History
Action,Adventure,Animation,Comedy,Drama,Family,Fantasy
Action,Animation,Thriller
Comedy,Documentary,Drama,Family,Romance
Action,Comedy,Family,Science Fiction
Action,Adventure,Family,Romance,TV Movie
Action,Crime,Fantasy,Music,Science Fiction,Thriller
Comedy,Drama,Foreign,Thriller
Comedy,Family,Mystery,Romance
Action,Crime,Fantasy,Romance
Action,Comedy,Fantasy,Romance
Comedy,Foreign,TV Movie
Action,Drama,Fantasy,Horror

```
Adventure,Comedy,Drama,Science Fiction
Action,Animation,Crime,Science Fiction,Thriller
Documentary,History,Mystery
Action,Adventure,Family,Fantasy,Science Fiction
Action,Animation,Fantasy,Foreign,Science Fiction
Adventure,Crime,Horror,Mystery,Thriller
Crime,Horror,Mystery,Science Fiction,Thriller
Adventure,Drama,Fantasy,Horror
Adventure,Fantasy,Romance,Science Fiction
Adventure,Crime,Drama,Mystery
Action,Comedy,Crime,Mystery,Thriller
Name: genres, dtype: int64
```

From here, our discussion can focus on the ways to account for class imbalance when creating our model (via class weights), or we can go into our discussion of reformatting the multi class system.

1.0.4 How do you sample your data, how many samples, and why?

We collect data from two sources: TMDB and IMDB.
The primary source of our data is the TMDB. We collect their full movie object including information about cast and crew. To ensure the sample is random we collect the data in the following way:
1. Get the id of the latest movie from TMDB.
2. Generate a random number between 1 and the lartest movie id.
3. Try to get a movie from TMDB using the random number as movie id.
4. If failed continue to step 2. 5. Get the movie object back. 6. If the movie obejct does not have valid IMDB id or genres continue to step 2. 6. Update the mobvie object with cast / crew information.
7. Save the movie object.
So far we've collected more than 55K TMDB movies with posters. They saved as a dictionary keyed by the TMDB movie id in 'pickle' format.
The secondary source of data is IMDB. Some of the movies we collect from the TMDB have missing values (budget, cast etc.). We collect those values from the IMDB. So far we've collected around 10K IMDB movies amd the porcess is ongoing. They also saved as a dictionary keyed by the TMDB movie id in 'pickle' format.

1.0.5 Description of your data

```
In [11]: tmdb_movie = tmdb_dict[397822]
         print 'Features from a TMDB movie object: '
         tmdb_movie.__dict__
```

```
Features from a TMDB movie object:
```

```
Out[11]: {'adult': False,
          'backdrop_path': None,
          'base_uri': 'https://api.themoviedb.org/3',
          'belongs_to_collection': None,
```

```
'budget': 0,
'cast': [{u'cast_id': 1,
  u'character': u'Sybille Erler',
  u'credit_id': u'573a4f059251415578000cc8',
  u'id': 1287454,
  u'name': u'Gertrud K\xfcckelmann',
  u'order': 1,
  u'profile_path': None},
 {u'cast_id': 2,
  u'character': u'Jochen Faber',
  u'credit_id': u'573a4f12c3a36806a6000cdb',
  u'id': 18546,
  u'name': u'Hans S\xf6hnker',
  u'order': 2,
  u'profile_path': u'/qjE8uzAwDmDMd8xJquF9Ivu9dTA.jpg'},
 {u'cast_id': 3,
  u'character': u'Elisabeth Faber',
  u'credit_id': u'573a4f2192514173d7000931',
  u'id': 1090603,
  u'name': u'Antje Weisgerber',
  u'order': 3,
  u'profile_path': u'/b5aYpopOsMm8ZuMrGbniKaB5UrX.jpg'},
 {u'cast_id': 4,
  u'character': u'Draaden',
  u'credit_id': u'573a4f309251415578000ccf',
  u'id': 13377,
  u'name': u'Paul Henckels',
  u'order': 4,
  u'profile_path': u'/zhobicq3xGKwQed7jddckd0sjHl.jpg'},
 {u'cast_id': 5,
  u'character': u'Dr. Hanna Claassen',
  u'credit_id': u'573a4f489251415578000cd8',
  u'id': 19524,
  u'name': u'Tilly Lauenstein',
  u'order': 5,
  u'profile_path': None},
 {u'cast_id': 6,
  u'character': u'Mutter der F\xfcrstin',
  u'credit_id': u'573a4f56c3a3687b9e00030c',
  u'id': 29556,
  u'name': u'Tilla Durieux',
  u'order': 6,
  u'profile_path': None},
 {u'cast_id': 7,
  u'character': u'F\xfcrst von Hartefeld-Rosenau',
  u'credit_id': u'573a4f68925141556c000cd3',
  u'id': 32003,
  u'name': u'Heinz Klingenberg',
```

```
               u'order': 7,
               u'profile_path': None},
              {u'cast_id': 8,
               u'character': u'Hertha',
               u'credit_id': u'573a4f74925141556c000cd5',
               u'id': 19526,
               u'name': u'Maria Sebaldt',
               u'order': 8,
               u'profile_path': None},
              {u'cast_id': 9,
               u'character': u'Alfred',
               u'credit_id': u'573a4f82c3a36806af000e63',
               u'id': 23720,
               u'name': u'Harald Juhnke',
               u'order': 9,
               u'profile_path': u'/oNcQzhur0vwzUck14b4YD1FlXnW.jpg'}],
             'crew': [{u'credit_id': u'573a4eea9251415578000cc3',
               u'department': u'Directing',
               u'id': 48330,
               u'job': u'Director',
               u'name': u'Wolfgang Liebeneiner',
               u'profile_path': u'/b7HMw705JBmwJtj9zzBbhkDipNh.jpg'}],
             'genres': [{u'id': 18, u'name': u'Drama'}],
             'homepage': None,
             'id': 397822,
             'imdb_id': u'tt0046379',
             'original_language': u'de',
             'original_title': u'Die St\xe4rkere',
             'overview': None,
             'popularity': 0.000219,
             'poster_path': u'/iVIUDbRdZeWvqG65BR04gQohn9V.jpg',
             'production_companies': [],
             'production_countries': [{u'iso_3166_1': u'DE', u'name': u'Germany'}],
             'release_date': u'1953-08-10',
             'revenue': 0,
             'runtime': None,
             'spoken_languages': [],
             'status': u'Released',
             'tagline': None,
             'title': u'Die St\xe4rkere',
             'video': False,
             'vote_average': 0.0,
             'vote_count': 0}
```


The poster images saved locally as JPEG files, the file name is the movie id plus '.jpg' extension.

```
In [12]: Image(url= 'posters/' + str(tmdb_movie.id) + '.jpg')

Out[12]: <IPython.core.display.Image object>
```

```
In [13]: imdb_movie = imdb_dict[270368]
         print 'Features from a IMDB movie object: '
         imdb_movie.items()
```

```
Features from a IMDB movie object:
```

```
Out[13]: [('rating', 7.8),
          ('genres', [u'Documentary', u'Short']),
          ('votes', 5),
          ('color info', [u'Black and White']),
          ('producer',
           [<Person id:0514156[http] name:_Liss, Abe_>,
            <Person id:6970759[http] name:_Magdoff, Sam_>]),
          (u'distributors', [<Company id:0512350[http] name:_Contemporary Films_>])
          ('title', u'Flavio'),
          ('writer', [<Person id:0662953[http] name:_Parks, Gordon_>]),
          ('camera and electrical department',
           [<Person id:0662953[http] name:_Parks, Gordon_>]),
          ('runtimes', [u'12']),
          ('director', [<Person id:0662953[http] name:_Parks, Gordon_>]),
          ('cast',
           [<Person id:0196239[http] name:_da Silva, Flavio_>,
            <Person id:6970758[http] name:_Martinez, Peter_>]),
          ('editor', [<Person id:1478240[http] name:_Gittler, Allan_>]),
          ('year', 1964),
          ('original music', [<Person id:0839657[http] name:_Suriñac, Carlos_>]),
          ('countries', [u'USA']),
          (u'production companies',
           [<Company id:0512349[http] name:_Elektra Studios_>]),
          ('kind', u'movie'),
          ('country codes', [u'us']),
          ('canonical title', u'Flavio'),
          ('long imdb title', u'Flavio (1964)'),
          ('long imdb canonical title', u'Flavio (1964)'),
          ('smart canonical title', u'Flavio'),
          ('smart long imdb canonical title', u'Flavio (1964)')]
```

1.0.6 What does your choice of Y look like?

We intend to use multilabel classification and assign a set of genre labels to each movie.

```
In [14]: mlb = MultiLabelBinarizer()
         y = mlb.fit_transform(labels)
         print 'Label matrix shape:', y.shape
         print 'Label example: ',  y[0]
         print 'The classes: ', mlb.classes_
```

```
Label matrix shape: (58829L, 20L)
Label example:  [0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

```
The classes:  [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']
```

1.0.7 Which features do you choose for X and why?

Here I think it would be great to really hone in on what our feature set will look like. I am a big fan of the textual analysis, movie poster image recognition and then the feature set we have with the inherent database.

We could use textual analysis. For example, we already try to use the overview TMDb field that contains the description of the movie in our Milestone 1 analysis (https://github.com/adubitskiy/cs109b/blob/master/Milestone_1_Submission.ipynb) and certainly helped to predict genres.

We also think that cast (actors/actresses) and crew (director, producer) could certainly help in predicting genres. Certain actors could appear mostly in comedies. A director could well-known for his horror films and so on.

Other features that could help: - popularity (budget, revenue, length, ratings). - length. Documentary movies could be shorter in general. - scripts. It would be great to bring in a script or script database and do preliminary analysis of differences in scripts among genres (https://github.com/AnnaVM/Project_Plotline/blob/master/code/scraping_script.py) - poster images. We would reduce the feature set via PCA. We could implement a basic PCA on one of the images to show the reduction of the pixel feature set.

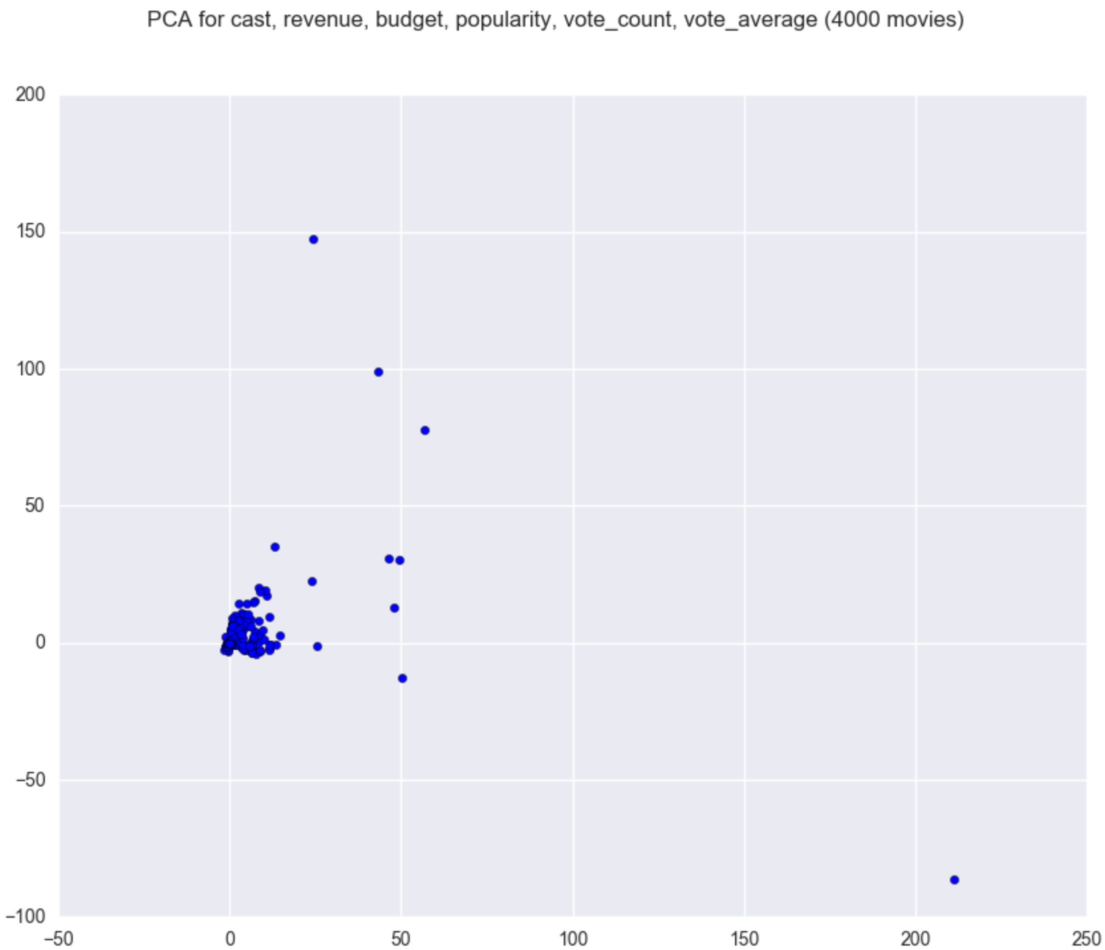
```
In [22]: import cPickle
         import random
         import time
         from collections import defaultdict, Counter

         import matplotlib.pyplot as plt
         import numpy as np
         import pandas as pd
         from sklearn import preprocessing
         from sklearn.decomposition import PCA
         from sklearn.manifold import MDS


         movie_dict = tmdb_dict

In [19]: def get_movie_attribute_name_list(movie_dict):
             attr_set = set()
             for movie in movie_dict.itervalues():
                 attr_set |= set(movie.__dict__.keys())
             return list(attr_set)


         def get_movie_df(movie_dict):
```

```
    all_movie_attribute_name_list = [
        # 'poster_path', 'backdrop_path', 'base_uri', 'id', 'imdb_id',
        # 'genres',

        'production_countries', 'overview', 'video',
        'title', 'tagline', 'crew', 'homepage',
        'belongs_to_collection', 'original_language', 'status', 'spoken_la
        'adult', 'production_companies',
        'original_title',

        'revenue', 'vote_count', 'release_date', 'popularity', 'budget', '
    ]
    movie_attribute_name_list = [
        'revenue',
        'vote_count',
        'popularity',
        'budget',
        'vote_average',
        'cast',
        'genres',
        # 'release_date',
        # 'runtime',
    ]

    movie_attribute_dict = defaultdict(list)
    for movie in movie_dict.itervalues():
        for movie_attribute_name in movie_attribute_name_list:
            attr_list = movie_attribute_dict[movie_attribute_name]
            attr_list.append(getattr(movie, movie_attribute_name))

    return pd.DataFrame(movie_attribute_dict)


def prepare_cast(movie_df):
    cast_list = [cast_member['name'] for movie_cast_list in movie_df['cast
    cast_counter = Counter(cast_list)

    appearances_limit = 2
    included_cast_list = [cast_name for cast_name, num_movies in cast_cour
                          if num_movies >= appearances_limit]
    included_cast_set = set(included_cast_list)

    num_movies = len(movie_df)
    #print num_movies
    movie_attribute_dict = defaultdict(lambda: np.zeros((num_movies,), dty

    for i, movie_cast_list in enumerate(movie_df['cast']):
        for cast_member in movie_cast_list:
```

```
                cast_name = cast_member['name']
                if cast_name in included_cast_set:
                    movie_attribute_dict['cast_' + cast_name][i] = 1

        new_movie_df = movie_df.drop("cast", axis=1)

        for key, column in movie_attribute_dict.iteritems():
            new_movie_df[key] = column

        print new_movie_df.shape
        return new_movie_df


    def prepare_columns(movie_df):
        return prepare_cast(movie_df)


    def get_subsample(movie_df, num_movies=1000):
        random.seed(109)
        return movie_df.loc[np.random.choice(movie_df.index, num_movies, repla


    def explore_pca(movie_df):
        movie_df = movie_df.drop(['genres'], axis=1)
        # print movie_df.describe()

        scaled_movies = preprocessing.scale(movie_df)

        pca = PCA(n_components=2)
        pca_X = pca.fit_transform(scaled_movies)

        print "explained variance ratio:"
        print pca.explained_variance_ratio_

        plt.figure(figsize=(10, 8))
        plt.scatter(pca_X[:, 0], pca_X[:, 1])
        plt.suptitle("PCA for cast, revenue, budget, popularity, vote_count, v
        plt.show()


    movie_df = get_movie_df(tmdb_dict)
    movie_df = get_subsample(movie_df, num_movies=4000)

    movie_df = prepare_columns(movie_df)

    explore_pca(movie_df)
```

```
(4000, 3590)
```

```
explained variance ratio:
[ 0.00453315  0.00374395]
```



One theory is that PCA could generate some principal components that represents popularity and genres as well. In this case we generated PCA for 4000 random movies and we looked at the actors and actresses with 2 more appearances in those movies (3590 actors in total). Actors were represented by binary columns and it dramatically increased dimensionality of the data. We took the first two principal components. Since their explained variance ratio is small (0.5% and 0.4%) the plot is not very interesting.

```
In [20]: def get_one_genre_name(movie_genres):
             one_genre = "Other"
             for genre in movie_genres:
                 genre_name = genre['name']
                 if genre_name == 'Comedy' or genre_name == 'Thriller':
                     one_genre = genre_name
                     break
             return one_genre
```

```
def convert_genres(genres):
    return [get_one_genre_name(movie_genres) for movie_genres in genres]


def explore_mds(movie_df):
    scaled_movies = preprocessing.scale(movie_df.drop('genres', axis=1))
    genres = movie_df['genres']
    genre_labels = np.array(convert_genres(genres))

    # print genre_labels

    mds = MDS(n_components=2, verbose=1, n_jobs=1, max_iter=120)
    mds_X = mds.fit_transform(scaled_movies)

    plt.figure(figsize=(10, 8))
    comedy_indices = genre_labels == 'Comedy'
    thriller_indices = genre_labels == 'Thriller'
    plt.scatter(mds_X[comedy_indices, 0], mds_X[comedy_indices, 1], c='r',
    plt.scatter(mds_X[thriller_indices, 0], mds_X[thriller_indices, 1], c=
    plt.legend()
    plt.suptitle("MDS for Comedy and Thriller movies")
    plt.show()


movie_df = get_movie_df(tmdb_dict)
movie_df = get_subsample(movie_df, num_movies=1000)

movie_df = prepare_columns(movie_df)
explore_mds(movie_df)
```

```
(1000, 380)
breaking at iteration 21 with stress 18038831.3493
breaking at iteration 9 with stress 18164057.0319
```

Here we used MDS for 1000 random movies, there were 380 actors with 2 or more appearances in these movies. We hoped to see some clustering based on genres. It wasn't the case. One can see some patterns but not easily separated two clusters.

```
In [ ]:
```