CS109b - Project - Team 14
Milestone 2: Assembling Training Data
April 12, 2017¶

Introduction

Following our EDA, and planning phase of Milestone 1, the team assembled data and choose features for Milestone 2. The process involved extensive data acquisition; iterative consultation among team members responsible for different data sources and subsets; and exploratory coding to see how the identified data might fit into certain modeling approaches.

This document addresses key questions about the nature of our data, as well as next steps in approaching Milestone 3.

The imbalanced nature of the data and how to address it

As noted by a teaching fellow during recent office hours, addressing imbalanced classes in the data is a classic problem, one to be resolved through engineering rather than through mathematics. Accordingly, our team has three approaches in mind for this challenge.

1. First, we intend to choose appropriate parameter values for actual calls to classic machine-learning algorithms. Such algorithms might include SVM and random forests.
2. Second, we plan to let the deep learning process iterate sufficiently long to allow minority classes to be seen repeatedly by the learning mechanism.
3. Finally, our team is researching third-party tools to generate data for re-balancing classes. The SMOTE package in R, for example, has this functionality.

Description of the data

The data used for learning in this project will consist of several parts, interlinked by a shared key: IMDb movie ID. The triple lookup table from MovieLens (referenced in our report for Milestone 1) continues to add value in unifying project data from multiple sources.

The three main types of data that we have acquired are as follows:

- **Structured data:** Our team has acquired structured data from three sources - MovieLens, IMDb, and TMDb. Where features are redundant from different sources, we give preference to the cleanest and best integrated data. For example, when considering the people involved with movies (e.g., actors, directors), we are preferring IMDb data on account of a streamlined format that includes IMDb movie ID, IMDb person ID, and role. Basic metadata that we will incorporate includes title, original title, and year. (Genre will be discussed below.) This structured data should be useful for a range of classic machine-learning algorithms.

- **Free-text data:** Two data fields, one from each major data source, should support text mining for genre classification. The first of these fields is the plot description from IMDb; the second field is the overview from TMDb. From these fields, we plan to create a document-term matrix, possibly apply PCA for data reduction, and possibly create TF-IDF scores for refinement. (It's also possible that free-text data could support machine learning, if sufficient project time permits.). If possible, we will incorporate movie reviews for mining.

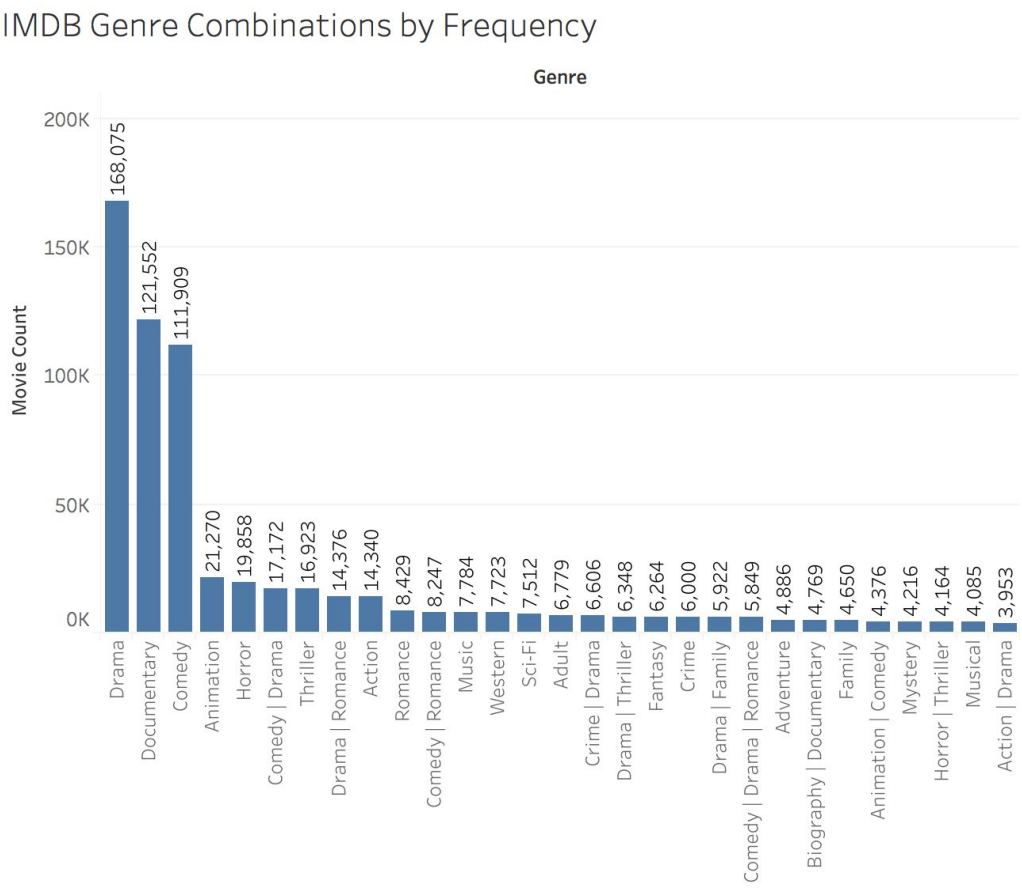
- **Image data:** Using the TMDb API, we downloaded approximately 40,000 movie posters to match our canonical movie set (as discussed below). The posters were downloaded in the w500 size, and we used a third-party tool called FastStone Photo Resizer to generate half-sized and quarter-sized images for exploratory coding. Image quality and ID validity have been verified manually on a sampling basis. The total data size of the w500 images is approximately 3 GB.

Choice of response variable (Y) for machine learning

During EDA for Milestone 1, our team found that the typical movie from IMDb or TMDb was labeled with multiple genres. Strategically, our team faces a choice between a multi-class or a multi-label approach to machine learning (including deep learning). Because the multi-class approach is more accessible and has been solved widely in industry tools, our team will pursue this approach.

Having chosen a multi-class approach, it is then necessary to apply reduction to the number of genres. Initially, for example, TMDB movies in our download set include approximately 3000 unique genres - construed as unique combinations of 1 or more individual genres. By ranking the most frequently-occurring genres, we intend to use a cumulative sum to account for 95%-99% of the movies. Our hope is that 100 or so genres will suffice. The remaining movies will be relabeled as having genre “Other” for classification purposes. Uncommon, longer genre compounds will be folded or remapped into simpler, shorter genre compounds.

A master list of singleton genres has been identified spanning MovieLens, IMDb, and TMDb (allowing for term unification and simplification). This master list is as follows: Action, Adventure, Biography, Comedy, Drama, Family, Fantasy, Horror, Mystery, Romance, Sci-Fi, Thriller, and Western. Combinations of these base genres forms a canonical list compound genres.



Movie Counts for each Genre Combination from 1 million IMdb movie records. The marks are labeled by Movie Count. Genre Combinations with less than 3,500 observations are not shown.

Choice of predictor variables (X) with rationale

Structured data Predictors

Parameter	Data type	Description	Relevance	Data source
title	string	the distribution title of the movie in North America	Title to identify movies by name	IMDb/TMDb/MovieLens
year/release_date	string	the year of release, or null		IMDb/TMDb/MovieLens
person (name/id)	integer	A person involved in the movie	Used with role	IMDb
role	String	a person's role in the movie: (director, writer, cast, production, manager, original music, casting director, visual effects)	Actor, director, writer, production manager	IMDb
production companies	list	list of production companies		TMDb
voting average rating	double	user rating on IMDb from 1 to 10	Relationship of popularity to genre type	IMDb/TMDb
vote_count	integer	number of votes	Relationship of number of votes to genre type	IMDb/TMDb
runtime	integer	Movie length in minutes	Relationship of movie length to genre types	TMDb
country_code	string list	production country (e.g., USA, AU)		IMDb
language_code	string list	list of languages		IMDb
original_languages	string list	Original language of the movie.		TMDb
budget	integer	Movie budget (million USD)		TMDb

Free-text Predictors

Parameter	Data type	Description	Relevance	Data source
plot_summary	string	Plot summary		IMDb/TMDb
plot_detailed	string list	expanded plot description		

Deep Learning Image Predictors

Parameter	Data type	Description	Relevance	Data source
poster	image file	Movie poster		IMDb/TMDb

Data sampling, size, and rationale

In order to unify as wide a range of data sources as possible, our team chose to rely on research data from the Department of Computer Science and Engineering at the University of Minnesota. The department hosts the MovieLens website, which is dedicated to movie ratings, prediction algorithms, and movie metadata. To support research in the area, the site makes available data sets of various sizes. The largest of these sets includes metadata and multi-source IDs for approximately 40,000 movies. This set has become the basis of our project for its authoritativeness, convenience, and closeness to our project goals. The data include movies spanning a range of production years, original languages, and (most importantly) genres. We are relying on the data sampling strategy of the provider, in this case, to some extent.

In terms of data size, 40,00 is the upper limit of the amount that the team expects to require. Recommendations from course staff suggest that some thousands of films will constitute an appropriate set for machine learning. Starting with an initial size of 40,000 should provide sufficient variability in the data, while allowing for the possibility of scaling down if computational issues arise.

Data reformatting and reduction of one-to-many relationships

The cast and credits for a movie are currently captured in a normalized relation format. For example, movies/actors constitutes a one-to-many relationship. The team is using the unique IMDb person ID, as noted above. We will treat movie people essentially like a traditional bag of words. First, we will reduce the list by removing people below a certain threshold of movie contribution (say, fewer 4 movies). Next, the data can be denormalized into a matrix for further reduction using Principle Components Analysis (PCA). This approach can be repeated for other roles such as director or writer.

Conclusion and next steps

Now that the data has been acquired for modeling, our team is eager to proceed with machine learning. The computational issues appear to be tractable in in our data size range, using a blend of personal computing and AWS cloud computing. Some issues of data linkage remain to be tackled, but the unique and shared IMDb should provide the master ‘glue’ for all of our acquired data.

For Milestone 3, the team team members will begin to specialise in roles and responsibilities. One subset of the team will explore several classic approaches to machine learning with structured, free-text, and image data. Another subset will dedicate some time to learning and exploring deep learning, in order to advance the team’s schedule through prototyping and knowledge sharing. (These subsets will partly overlap.)

As the parameters of data acquisition and linkage have become clearer, possibilities for machine learning have firmed up as well. It will be interesting to see where this exploration leads during Milestone 3.

imdbpy_data_extract

April 12, 2017

0.1 Data Extraction and Manipulation from IMDB text files

Links
http://imdbpy.sourceforge.net

```
In [2]: import pandas as pd
        from imdb import IMDb
        import datetime
        import time
        import re
        import numpy as np
```

Read in movie (imbd_ids) to be downloaded The list of ids comes from the movie lense data set

```
In [ ]: movie_list = pd.read_csv('./datasets/ml_data.csv', usecols = ["imdbId"])

        #for re-nrun on failures from first pass
        #movie_list = pd.read_csv('./datasets/pass2/imdb_fail.csv', usecols=["imbd_

        #size of data frame
        print 'number of movies:', movie_list.shape[0]

        movie_list.head(10)
```

Role master table (people labels in imdb XML)

```
In [21]: roles    = ["director", "writer","cast", "production-manager"
                    ,"original music","casting director","visual effects"]
         role_ids = [1,2,3,4,5,6,7]
         df_roles = pd.DataFrame({'role_id' : role_ids, 'role' : roles })

         #Roles available but excluded
         #-music-department
         #-art direction
         #-sound-crew
         #-art-department
```

imdb helper functions
gets all people for a given role & movie

```
In [22]: #returns a dataframe (imbd_id,role_id,person_id,name)
         def people_in_role(movie, imdb_id, role = "cast", role_id = 1):
             names=[]
             person_ids=[]
             n = 0

             try:
                 people_count = len(movie[role])

                 for person in movie[role]:
                     name = str(person)

                     #Extract the Person ID
                     person_xml = person.asXML()
                     try:
                         # returns <person id="#">
                         found = re.search('(<person id="([^"]|"")*")'
                                           , person_xml).group(1)
                         # returns "#"
                         found = re.search('("([^"]|"")*")', found).group(1)

                         # returns #
                         person_id = int(found[1:-1])
                     except AttributeError:
                         found = '0'

                     names.append(name)
                     person_ids.append(person_id)
                     n = n + 1
             except:
                     #no people in this movie have this role
                     n = 0

             return pd.DataFrame({   'imbd_id'   : [imdb_id] * n,
                                     'role_id'   : [role_id] * n,
                                     'person_id' : person_ids,
                                     'name'      : names
                                 })
```

gets all people in all roles for a given movie

```
In [23]: #returns a dataframe (imbd_id,role_id,person_id,name)
         def people_in_movie(movie, imdb_id):

             ls_movie_people = []
```

```
            df_movie_people = pd.DataFrame({    'imbd_id'    : [0]  * 0,
                                                'role_id'    : [0]  * 0,
                                                'person_id' : [0]  * 0,
                                                'name'       : [''] * 0
                                           })

            #gather people for each role
            for i in xrange((len(df_roles.index))):
                role_id = df_roles.role_id[i] #df_roles.loc[i,0].value
                role    = df_roles.role[i]

                p = people_in_role(movie = movie, imdb_id = imdb_id, role_id = rol
                ls_movie_people.append(p)

            #gather roles into single dataframe
            df_movie_people = pd.concat(ls_movie_people)

            return df_movie_people
```

Gets an XML tag for a movie with a default value if the tag is missing

```
In [24]: #wrapper for imdbpy funciton movie
         #adds error handeling incase xml tag is missing
         def movie_isnull(movie, tag_name, default_value=""):
           try:
             return_val = movie[tag_name]
           except:
             return_val = default_value
           return return_val
```

IMDb extract main funciton
Extract matching imdb movies that are in the movie lense list

```
In [ ]: #connect to imdb web service
        ia = IMDb()

        ls_movie = []
        ls_people = []
        ls_fail = []

        for i in xrange(len(movie_list.index)):

            #Slow the script to avoid overloading the imdb server
            time.sleep(.2)

            #retrieve movie object form imdb
            imdb_id = movie_list.iloc[i,0]
```

```
            try:
                #get movie object (contains movie XML data and helper functions)
                movie = ia.get_movie(str(imdb_id).zfill(7))

                #get 1:1 flat file fields and simple 1:[1D array] fields for movie
                d_movie = { 'imdb_id'        : imdb_id,
                            'title'          : movie_isnull(movie, "title", ""),
                            'rating'         : movie_isnull(movie, "rating", np.nan
                            'votes'          : movie_isnull(movie, "votes", np.nan)
                            'runtime'        : movie_isnull(movie, "runtimes", [np.
                            'year'           : movie_isnull(movie, "year", np.nan),
                            'cover_url'      : movie_isnull(movie, "cover url", "")
                            'cover_url_full' : movie_isnull(movie, "cover_url_full"
                            'plot_outline'   : movie_isnull(movie, "plot outline",
                            'kind'           : movie_isnull(movie, "kind", "None"),
                            'genres'         : '|'.join(movie_isnull(movie, "genres
                            'language_codes' : '|'.join(movie_isnull(movie, "langua
                            'country_codes'  : '|'.join(movie_isnull(movie, "countr
                            'plot'           : '\n'.join(movie_isnull(movie, "plot"
                          }

                #add movie record to list of successful downloads
                ls_movie.append(pd.DataFrame.from_records([d_movie]))

                #get people (actors, directors, writers ect)
                ls_people.append(people_in_movie(movie, imdb_id=imdb_id))

            except:
                #record failures for diagnostics and re-run
                ls_fail.append(imdb_id)

            #every thousand records: print status update
            if i % 100 == 0:
                Print status update
                print i, len(ls_fail), datetime.datetime.now().time()

        df_movies = pd.concat(ls_movie)
        df_people = pd.concat(ls_people)
```

In [30]:
```
#write results to file
df_movies.to_csv('imdb_movies.csv', encoding='utf-8')
df_people.to_csv('imdb_people.csv', encoding='utf-8')
df_roles.to_csv('imdb_roles.csv', encoding='utf-8')
```

In [31]:
```
#write failures to csv
df_fail = pd.DataFrame({'imbd_id'  : ls_fail})
df_fail.to_csv('imdb_fail.csv', encoding='utf-8')
print 'number of failures:', df_fail.shape[0]
```

```
number of failures: 0
```

```
In [ ]: #Combine Pass 1 and pass 2
        df_movies1 = pd.read_csv('./datasets/pass1/imdb_movies.csv')
        df_movies2 = pd.read_csv('./datasets/pass2/imdb_movies.csv')
        df_movies = pd.concat([df_movies1,df_movies2])
        df_movies.to_csv('imdb_movies.csv', encoding='utf-8')
        print len(df_movies.index)

        df_people1 = pd.read_csv('./datasets/pass1/imdb_people.csv',encoding='utf-8
        df_people2 = pd.read_csv('./datasets/pass2/imdb_people.csv',encoding='utf-8
        df_people = pd.concat([df_people1,df_people2])
        df_people.to_csv('imdb_people.csv', encoding='utf-8')
```

Project Milestone 2 - TMDB download

Team 14, Harvard CS109B, Spring 2017

April 2017

```
# set up paths
# project_dir <- "/Users/davidmodjeska/_Project/"
# data_filename <- "tmdb_data.csv"
# genres_filename <- "tmdb_genre_list.csv"
# top_movies_filename <- "top_10_movies_2016.csv"
```

```
#-------- General Setup ---------
#
# options(stringsAsFactors = FALSE)
#
# url <- "https://api.themoviedb.org"
# api_key <- "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" # for user David Modjeska
#
# function to make a GET request and convert the raw content to JSON
# start_time <- Sys.time()
# make_get_request <- function(url, path) {
#     raw_result <- GET(url = url, path = path)
#     recs_remaining <- raw_result$headers[["x-ratelimit-remaining"]]
#     if (recs_remaining == '39') {
#         start_time <<- Sys.time()
#     } else if (recs_remaining == '0') {
#         seconds <- as.double(difftime(Sys.time(), start_time, u = 'secs'))
#         if (seconds < 10) {
#             Sys.sleep((10 - seconds) + 1)
#         }
#     }
#
#     if (raw_result$status_code != 200)
#         stop(paste0("Creating request token failed with error code ",
#                     raw_result$status_code))
#
#     this_raw_content <- rawToChar(raw_result$content)
#     this_content <- fromJSON(this_raw_content)
#
#     return(this_content)
# }
#
#
# #----------- API Session -------------
#
# # create request token
# path <- paste0("3/authentication/token/new?api_key=", api_key)
# this_content <- make_get_request(url, path)
# request_token <- this_content$request_token
#
# # validate request token interactively
# validate_url <- paste0("https://www.themoviedb.org/authenticate/",
```

```
#                         request_token)
# browseURL(validate_url)
#
# # create session
# path <- paste0("3/authentication/session/new?api_key=", api_key,
#                "&request_token=", request_token)
# this_content <- make_get_request(url, path)
# session_id <- this_content$session_id
#
# # get genre list
# path <- paste0("3/genre/movie/list?api_key=", api_key, "&language=en-US")
# this_content <- make_get_request(url, path)
# genres <- this_content$genres
#
# genres_filename <- "tmdb_genre_list.csv"
# out_file <- str_c(project_dir, genres_filename)
# write_csv(genres, out_file)
#
# # get configuration
# path <- paste0("3/configuration?api_key=", api_key)
# this_content <- make_get_request(url, path)
# base_url <- this_content$images$base_url
# poster_size <- 'w500'
#
# #---------- A Favorite Movie: The Godfather ------------
#
# # find by external ID - "The Godfather"
# path <- paste0("3/find/tt0068646?api_key=", api_key,
#                "&language=en-US&external_source=imdb_id")
# this_content <- make_get_request(url, path)
# original_title <- this_content$movie_results$original_title
# genre_ids <- this_content$movie_results$genre_ids
# poster_path <- this_content$movie_results$poster_path
#
# # list this movie's genres on tmdb
# # note: this movie's genres on IMDB are: Crime, Drama
# this_movie_genres <- genres %>%
#     filter(id %in% genre_ids[[1]]) %>%
#     select(name)
# print(original_title)
# print(this_movie_genres)
#
# # plot poster
# poster_url <-paste0(base_url, poster_size, poster_path)
# poster_filename <- "poster.jpg"
# download.file(poster_url, poster_filename)
# poster <- load.image(poster_filename)
# plot(poster)
#
# #---------- Top movies in 2016 ------------
#
# # get data
# num_movies <- 10
```

```
# path <- paste0("3/discover/movie?api_key=", api_key,
#                "&language=en-US",
#                "&sort_by=popularity.desc",
#                "&include_adult=false",
#                "&include_video=false",
#                "&page=1",
#                "&year=2016")
# this_content <- make_get_request(url, path)
# original_title_10 <- this_content$results$original_title[1:num_movies]
# genre_ids_10 <- this_content$results$genre_ids[1:num_movies]
#
# # print data
# print_data <- data.frame(MOVIE <- character(), GENRE <- character())
# for (i in 1:num_movies) {
#     this_title <- original_title_10[i]
#     these_ids <- genre_ids_10[i]
#     these_genres <- genres %>%
#         filter(id %in% these_ids[[1]]) %>%
#         select(name) %>%
#         summarize(GENRE = paste(name, collapse = ", "))
#     this_data <- data.frame(MOVIE = this_title, GENRE = these_genres)
#     print_data <- rbind(print_data, this_data)
# }
#
# out_file <- str_c(project_dir, "top_10_movies_2016.csv")
# write_csv(print_data, out_file)
#
# #------- DOWNLOAD -----------------
#
# # set directories
# project_dir <- "/Users/davidmodjeska/CS109b/Project/"
# ml_data_dir <- "MovieLens/ml-latest/"
# tmdb_data_dir <- "TMDb/"
#
# # read ML data to get IMDb IDs
# in_file <- str_c(project_dir, "ml_data.csv")
# ml_data <- read_csv(in_file, col_names = TRUE, col_types = 'iccincii') %>%
#     mutate(num_genres = as.integer(num_genres))
#
# # extract movie results from one downloaded data chunk
# make_movie_rec <- function(this_content) {
#     this_movie_record <- this_content$movie_results
#     this_movie_record$genre_ids = sub("^c[(](.*)[)]$", "\\1",
#                                       this_movie_record$genre_ids)
#
#     return(this_movie_record)
# }
#
# # download a sample record
# imdb_id <- ml_data$imdbId[1]
# path <- paste0("3/find/tt0", imdb_id, "?api_key=", api_key,
#                "&language=en-US&external_source=imdb_id")
# this_content <- make_get_request(url, path)
```

```
# first_movie_record <- make_movie_rec(this_content)
#
# # get multiple records (note that make_get_request() will sleep as needed)
# all_movie_records <- first_movie_record
# for (i in 2:nrow(ml_data)) {
#     imdb_id <- str_pad(ml_data$imdbId[i], 7, pad = '0')
#     path <- paste0("3/find/tt", imdb_id, "?api_key=", api_key,
#                    "&language=en-US&external_source=imdb_id")
#     this_content <- make_get_request(url, path)
#     if (is.data.frame(this_content$movie_results)) {
#         this_movie_record <- make_movie_rec(this_content)
#         all_movie_records <- rbind(all_movie_records, this_movie_record)
#     }
# }
#
# # count number of genres per movie
# all_movie_records <- mutate(all_movie_records,
#                             num_genres = str_count(genre_ids, '[,]') + 1) %>%
#    select(-backdrop_path, -poster_path)
#
#
# # save file
# out_file <- str_c(project_dir, data_filename)
# write_csv(all_movie_records, out_file)
#
# #----------- DOWNLOAD POSTERS
#
# for (i in 1:nrow(ml_data)) {
#     imdb_id <- str_pad(ml_data$imdbId[i], 7, pad = '0')
#     path <- paste0("3/find/tt", imdb_id, "?api_key=", api_key,
#                    "&language=en-US&external_source=imdb_id")
#     this_content <- make_get_request(url, path)
#     if (is.data.frame(this_content$movie_results)) {
#         poster_path <- this_content$movie_results$poster_path
#         poster_url <-paste0(base_url, poster_size, poster_path)
#         poster_filename <- str_c(poster_dir, imdb_id, ".jpg")
#         download.file(poster_url, poster_filename)
#     }
# }
```