CIS4930 Group Project

Dax Gerts, Christine Moore, Carl Amko, Denzel Mathew, Aaron Silcott

December 16, 2015

Introduction

The following report details the research and procedures designed to carry out data mining tasks on a reasonable sample of the Internet Movie Database (IMDb) and its collection of movies. These procedures were broken up into the following major categories:

- Dataset Creation
- Genre prediction (classification)
- *The usual* casts (association rule mining)
- Finding similar movies (clustering)

Literature Review

Source No. 1 'Automatic Video Classification: A Survey of the Literature' by Darin Brezeale and Diane J. Cook, Senior Member, IEEE IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, VOL. UNKNOWN, NO. UNKNOWN, UNKNOWN 2007

This source is an analysis of the different methods and areas of interest related to classifying videos by a genre. This paper uses more video based processing in order to classify footage from movies or their trailers by genre or type. While this paper uses attributes and features along the lines of optical flow, pixel clustering, and image histogram differences, it demonstrates that other features besides facts about the movie can be used to classify genres. While our project did not involve any video/image processing or computer vision algorithms, I think it nciely demonstrates that there are many "out of the box" attributes and features that can be used as classification attributes. Our takeaway from this paper is to always be thinking of all the different kinds features or traits that can be used for classifcation problems.

Source No. 2 'Predicting Movie Success Based on IMDB Data' by Nithin VR, Pranav M, Sarath Babu PB, and Lijiya A International Journal of Data Mining Techniques and Applications Volume: 03, June 2014, Pages: 365-368

This source investigated Rif there was a way to predict monetary success and high ratings using the data from IMDB. Both our project and the work completed in the source used the same source data files pulled from IMDB. However the data utilized differs between us. The people in the paper selected only movies from 2000 to 2012 and ones that were in English and from the US, whereas our data was from a longer timespan and not limited to the other constraints. For the sake of more accurate results, we could do what they did and strategically pick subsets of attributes for processing. They also only selected data with available box office gross data, whereas some of our data if extremely incomplete. The paper also utilizes data from Rotten Tomatoes and Wikipedia. Another area of difference is that the paper was looking for correlations where we were tackling a classification problem. An interesting approach these authors took that could be implemented with our datasets is a greedy backward procedure. The authors removes the worst attribute at each iteration of checking correlation values as well as any redundant attributes in order to produce the ideal subset. This paper calculates its dependents using linear regression and support vector models which our project doesn't touch as much. Their best results yielded 51% accuracy, so not a number to be used as a definite predictor. While their results weren't perfect, some of their methodologies could be implemented for our task of genre classification.

Overview of Data

The data overview is divided into the following steps, as significant decisions or observations worth explaining were made at each point.

- Building the dataset
- Preprocessing
- Datset Statistics

Building the Dataset

Numerous efforts were carried out to build a work sample set for this project, and while an exhaustive amount of detail could be written for each attempted method, only a brief overview of each attempt will be given before providing a more lengthy explanation of the methods that were finally chosen.

Round One: Load flat files from FTP server directly into R

- Efforts: Several helper functions were written to load IMDb files into R, parse first as Raw Text, and clean. Attempts were then made to join the contents of each file into complete working set before sampling.
- Cons: The files were too large to effectively load and clean in R. While a sample could be taken from the *movies* file fairly easily, each attribute joined would require traversal and cleaning of approximately 0.5 GB of raw text.

Round Two: Reconstruct data in mySQL database using IMDb's *imdbpy2sql.py* program

- Efforts: Multiple attempts were made to fully reconstruct the IMDb database from flat files. Once the script would complete running, the finished database was either queried for existing tables or backed up to csv files.
- Cons: Each attempt to either provide a sample set from mySQL query or from performing a sqldump to csv files showed that the movie IDs were missing, rendering the data useless.

Round Three: Python Script and Web APIs

- Efforts: Employed python scripts and API calls to the Internet Movie Database and the Open Movie Database.
- Cons: Very slow (internet connection-bound)

Our working dataset was created using a python script making calls to the OMDb API. While this method was slow, completing approximately 3-5 http requests per second on an average wifi connection, it saved an enormous amount of time in preprocessing and fine-tuning as it loaded all the available, useful information as JSON objects which could be read into R efficiently and cheaply.

The *imdbscraper.py* script was set to a sample size of 30 thousand values and made a unique http request for each randomly generated seven-digit id up to the approximate maximum of 5262000. Each request return a json object which was stored in an array of sample size.

Once the specified number of requests were made, the json array was dumped and written to a file.

Note that although the dataset is created from OMBb's 3rd-party API, each request contains an IMDb ID corresponding to the same values in the IMDb database.

It was discovered further on in the process on building the sample set that the Open Movie Database was missing a number of values from the IMDb database. To rectify this, another python script, *castscrape.py* was

written using a list of the subset of IMDb movie IDs selected in preprocessing. The script used the IMDbPy API to return additional cast items: *Producer*, *Cinematographer*, *Composer*, and *CostumeDesigners*, with varying degress of success.

While *Producre* and *Cinematographer* were succesfully retrieved, albeit requiring a substantial amount of parsing and cleaning via regular expressions, the inavailability of documentation led to *Cinematographer* and *Composer* being scrapped as viable candidate variables.

At the end of initial dataset building phase, the working set contained the following variables.

```
##  [1] "imdbID"          "Plot"            "Rated"
##  [4] "Title"           "Writer"          "Type"
##  [7] "imdbVotes"       "seriesID"        "Season"
## [10] "Director"        "Released"        "Awards"
## [13] "Genre"           "imdbRating"      "Poster"
## [16] "Episode"         "Language"        "Country"
## [19] "Runtime"         "Metascore"       "Response"
## [22] "Year"            "Error"           "Actor1"
## [25] "Actor2"          "Actor3"          "Actor4"
## [28] "Producer"        "Cinematographer"
```

Preprocessing

Load Dataset Into R

Having created a working dataset, the function, *jsonToCsv*, was written to speed up the process of formatting the data for use in R. *jsonToCsv* first reads the json object from the given file (defaulting to *imdb_30K_sample.json*) using the R package *jsonlite*. While the function returns a data frame, it also backs up the data to a local csv file.

Not all of the original attributes described in the previous section were for use in any data mining task, however they were retained for archival purposes. In particular, *seriesID*, *Episode*, and *Season* were largely useless because non-movie elements were dropped from the sample set. Also, the values for *Poster*, *imdbID*, *Response*, and *Error* were artifacts retained from converting the JSON objects to an R useable format.

Each attribute were individually examined and set to the right type. An example of some of the variable preprocessing can be seen below with the *timeSet* function which was used on *Runtime*. The original *Runtime* data had three cases: **"N/A"**, **"# hours # min"**, or **"# min"**. These were reduced down to as single numeric for the number of minutes, with "N/A" values being 0 minutes.

```
# Runtime reformat, string -> minutes (integer)
timeSet <- function(x) {
  if(length(x == 2)) {
    x = as.numeric(x)
  } else if (length(x == 4))  {
    x = (as.numeric(substr(x,1,1))*60)+as.numeric(substr(x,2,3))
  } else {
    x = as.numeric("0")
  }
  x
}
```

The values for *Writers*, *Actors*, *Genres*, and several other variables were originally retrieved from JSON as strings of names, often with rough formatting. What was typically done to handle this format was to specify a maximum number of accepted values, in the case of actors 4, and expand the list into that maximum number of unique columns.

```
## [1] "Actor1" "Actor2" "Actor3" "Actor4"
```

More details about the preprocessing steps can be found in the appendix with the provided code base.

The preprocessing stage ends with two significant steps:

1. Subset data to retain only movies (30K rows ==> ~11K rows)
2. Save dataset to R data object and to csv (for coherent use across all project members)

At the end of the preprocessing stage, the full count of movies by data set attribute was as shown below. Note that variables pertaining to TV shows and other non-movie entries show 0 results for movies. This is a results of building the final sample set.

```
##          imdbID            Plot           Rated           Title
##           11142           11142            1164           11142
##          Writer            Type       imdbVotes        seriesID
##            7055           11142            4228               0
##          Season        Director        Released          Awards
##               0            9696            6712           11142
##           Genre      imdbRating          Poster         Episode
##           11142            4567           11142               0
##        Language         Country         Runtime       Metascore
##           11142           11142            6916             133
##        Response            Year           Error          Actor1
##           11142           11142               0            9080
##          Actor2          Actor3          Actor4        Producer
##            8150            7363            6698            6620
## Cinematographer
##            6318
```

Dataset Statistics

The following are some of the observations made in determining statistical validity for our sample set, as well grounds for determing how to subset the original 30K-row rough samples set.

Before Preprocessing

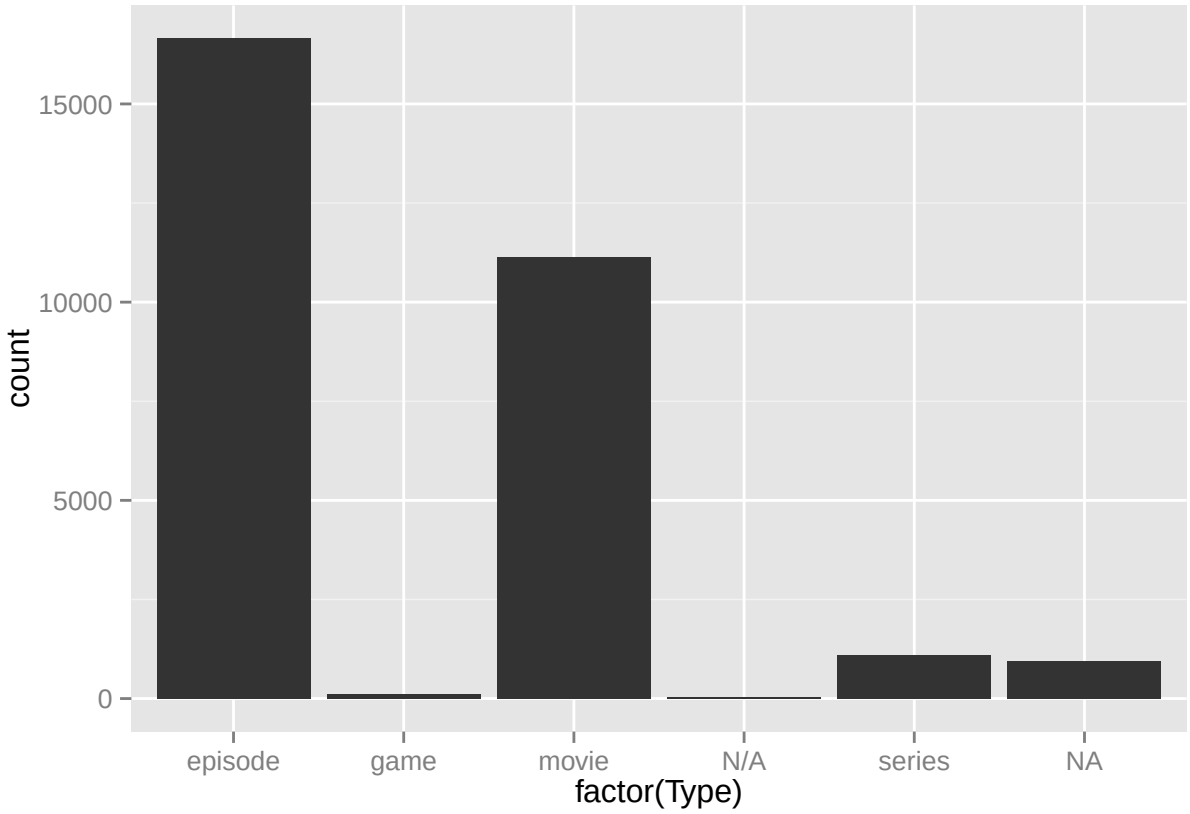
An initial query was made to examine how many of the original 30K elements were valid movies as well as how many showed up as null values.

```
# Prepare example set
stats <- jsonToCsv()

# Null rows in original query set (dropped in preprocessing)
sum(is.na(as.factor(stats$Type)))
```

```
## [1] 950
```

```
# Entry-type information by level
ggplot(stats, aes(x = factor(Type))) + geom_bar(stat = "bin")
```



As shown here, a significant majority of the tuples were actually episodes, which made sense in the context of there being many episodes per TV show, each with there own entry. Although outside the scope of the problems tackled here, there was a significant right skew to the number of seasons per show, indicating that many, potentially even a majority of TV shows only air for a single season.

After Preprocessing

To confirm successful deletion of non-movie and null entities from the sample data, the following code was run.

```
# Load preprocessed data
stats <- readRDS("clean10Kdataset.rds")

# Null rows in original query set (dropped in preprocessing)
sum(is.na(as.factor(stats$Type)))
```

```
## [1] 0
```

```
# Entry-type information by level
summary(as.factor(stats$Type))
```

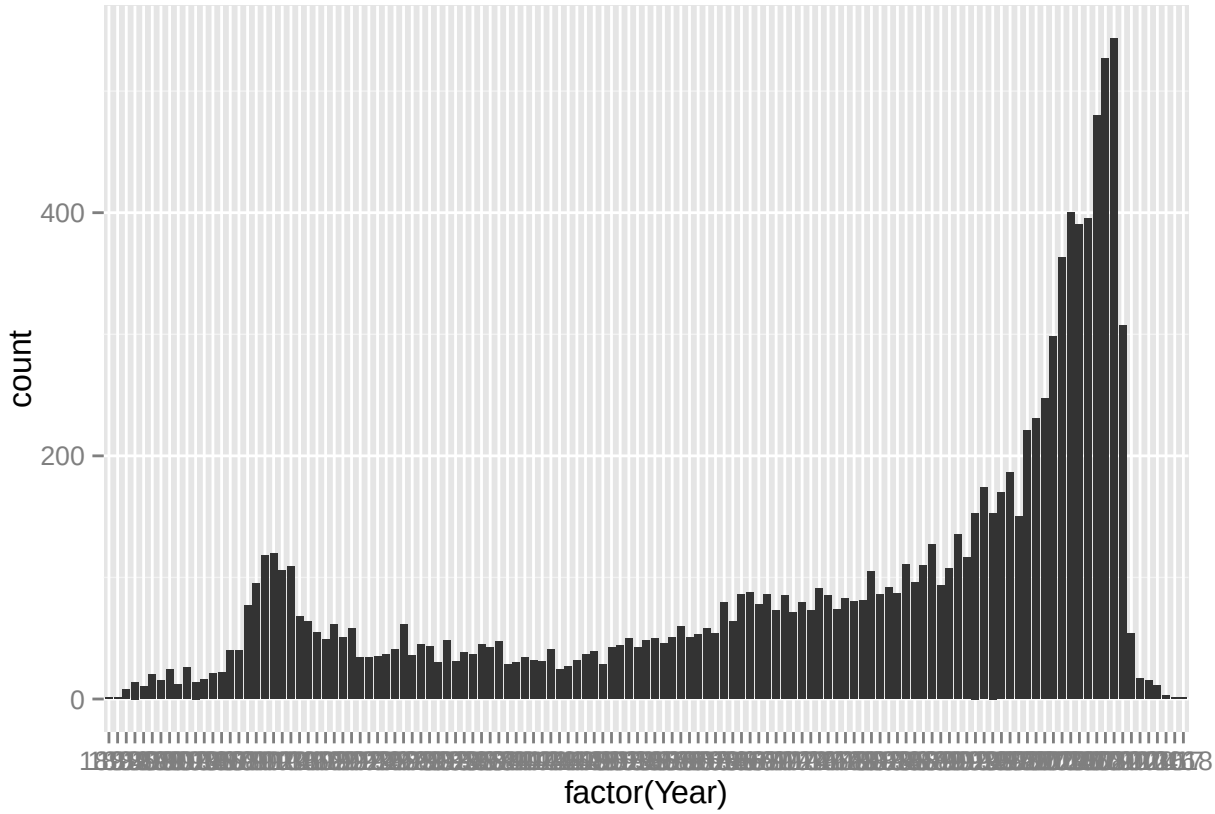
```
## episode    game   movie     N/A  series
##       0       0   11142       0       0
```

The results show that roughly a two-thirds of the original thirty thousand sample values were dropped in preprocessing. The remaining values, approximately eleven thousand elements, provide a small but adequate sample set for initial data mining tasks.

The following are some fairly self-explanatory visualizations of the sample data.

Movies By Year There was a clear and reasonable increase in the number of movies sampled for each year approaching the present day, which can be inferred to correspond to the growth of the movie industry over time.

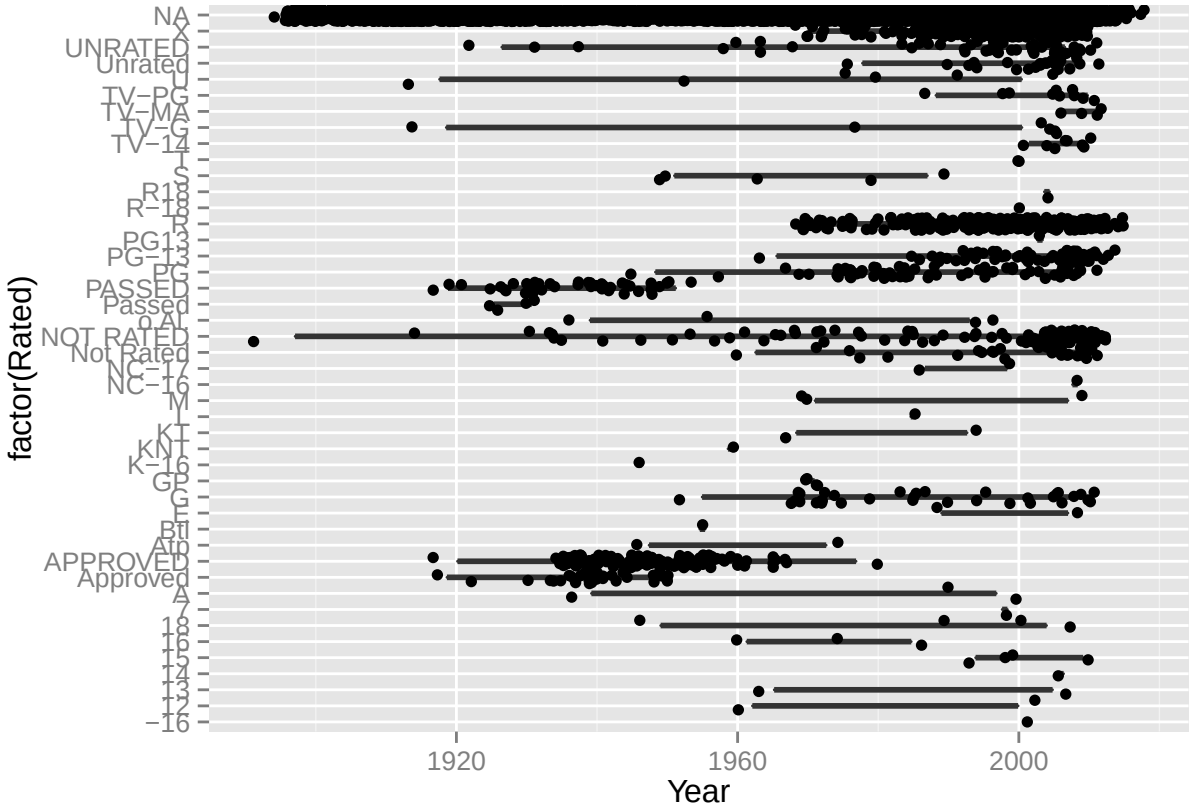
```
# Display distribution of movie years
ggplot(stats, aes(x = factor(Year))) + geom_bar(stat = "bin")
```



Ratings Given By Year

While someone crowded, the changes in movie rating methods over periods of time can be seen, as well as the clear stratification of films marketed towards different age groups.

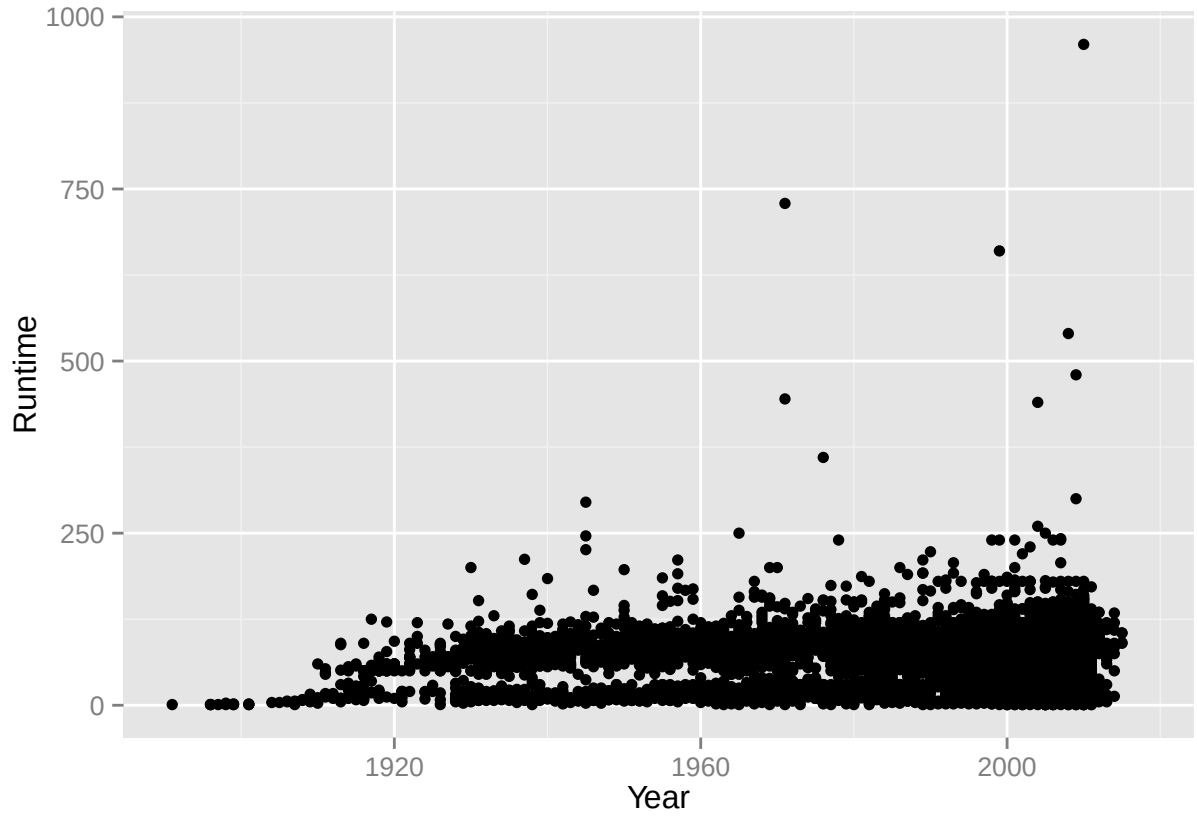
```
# Display ratings by year
qplot(Year,factor(Rated),data=stats,geom=c("boxplot","jitter"))
```



Movie Length By Year

One last visualization provides the distribution of movie runtimes over the years. While several inferences could be made, without further analysis the only thing that can be safely assumed is that there is a clear distinction between short movies (under an hour) and longer films and that the distinction appears to fade as we move towards the present day.

```
# Display distribution of movies runtimes by years
qplot(Year,Runtime,data=stats)
```



Method and Materials

Genre Classification

Unique Genres

The unique genre classes were found by performing a pruning of the initial genre data.

```
#Prune the data to remove excess collection space
dataPruned <- lapply(dataGenre, function(x) {
  # Make the collection 'unique', removing all the blank collection spots
  x <- unique(x)

  # chop off the last 'NA' value
  x <- x[-length(x)]

})
#list unique genre entries
unique(unlist(dataPruned))
```

This resulted in 29 unique genre classes and therefore 29 clusters to use in cluster analysis.

The task of genre prediction reflected the efforts made in Individual Project I. Using classification methods RIPPER, C4.5, oblique trees, naive bayes, and the knn classifier, we hoped effectively predict the genre of a film based off its other characteristics.

One of the first steps in achieving this goal was first formatting the data in a way that was helpful and relevant to the task at hand. For example, the imdbID attribute was removed from the dataset when creating the training and test sets since it was independent of predicting the genre of the film.

Another type of format preprocessing that we addressed was for the Producer and Cinematographer columns the data was formatted as such ['LastName, FirstName']. In addition the date for the 'Writer' column had parenthetical notes that also affected how the classification algorithm performed. Using Regex's and gsub, these attribute values were modified to fit an acceptable format. Additionally, we also trimmed the column data by removing the attributes that are independent of genre such as Rating. Some of the attributes were stored as lists so using the 'unnest' function was necessary in order to process the data. Any row with NA as the Genre value was also removed. All the nominal values stored in the data also needed to be cast as factors before being run through the classification algorithms.

Here are the classification algorithms we utilized, note we don't run the oblique method because it fails due to memory allocation:

```
#RIPPER CLASSIFIER
ripperModelMovie <- JRip(Genre~., data = training)
ripperPredictionsMovie <- predict(ripperModelMovie, movieDataWOutGenre)
# summarize results
ripCMMovie <- confusionMatrix(ripperPredictionsMovie, test$Genre)


#C4.5 CLASSIFICATION
c45ModelMovie <- J48(Genre~., data = training)
c45ModelPredictionsMovie <- predict(c45ModelMovie, test[,-12])
c45CMMovie <- confusionMatrix(c45ModelPredictionsMovie, test$Genre)


#OBLIQUE CLASSIFICATION
obliqueModelMovie <- oblique.tree(formula = Genre~., data = training,
                                  oblique.splits = "only")
obliqueModelPredictionsMovie <- predict(obliqueModelMovie, test)
obCMMovie <- confusionMatrix(colnames(obliqueModelPredictionsMovie)
                             [max.col(obliqueModelPredictionsMovie)], test$Genre)


#NAIVE BAYES CLASSIFIER
# train a naive bayes model
naiveBayesModel <- naiveBayes(Genre~., data=training)
# make predictions
#look at this
predictions <- predict(naiveBayesModel, movieDataWOutGenre)
# summarize results
nbCMMovie <- confusionMatrix(predictions, test$Genre)


#KNN CLASSIFIER
#lets try getting rid of NA values
subset <- test$Genre[1:245]
kkModel <- kknn(Genre~., test = test, train = training, distance = 1,
                kernel = "triangular")
p <- predict(kkModel, movieDataWOutGenre)
knnCMMovie <- confusionMatrix(p, subset)
```

Association Rules

For Association Rule Mining, the first subset of the dataset was reduced down to just cast members compromising of Director, Producer, Writer, and Actors. Actors initially was brought in a list of Actors and further preprocessing was done on this end. This proved lackluster and was eventually moved to the dataset creation end of the project. Actors were then fractured into four Actor components and handled appropriately.

```
# Load dataset/subset dataset
assocData <- readRDS("clean10Kdataset.rds")

# Remove columns not worthy of analysis
assocDataSubset <- assocData[,c("Director","Producer","Writer","Cinematographer",
                                "Actor1","Actor2","Actor3","Actor4")]
```

The dataset was then passed into the apriori function to generate rules

```
#Build ruleset
movieRules <- apriori(assocDataSubset,parameter = list(support = 0.0001,confidence = 0.9));
```

Redundant rules were then removed.

```
#Removes redundant rules
uniqueMovieRules <- redundantRules(movieRules)
```

Sorted ruleset by lift.

```
#Sorts trimmed rules by lift
sortedUniqueMovieRules <- sort(uniqueMovieRules, by = "lift")
```

For the Genre/Cast portion of the Association Rules, I attempted to bring in the Genre as a factor of 29 attributes and tried to merge it back into the main subset. This did not prove useful when trying run apriori as it gave errors of it not being in logical or factor format.

Clustering

For cluster analysis, the dataset was reduced to included just the 'Genre' attribute. This was chosen because we believed it had the strongest correlation to grouping similar movies, as well as convenience to match the number of clusters (instructed to be equal to the number of genres). In order to prepare the dataset, the working set was converted into a weighted transactional database, where the weight was equal to one over the number of constituent genres.

```
# Load dataset
data <- readRDS("clean10Kdataset.rds")
```

As mentioned above, we found 29 distinct genres to use as clusters. This included "N/A" as a genre. We then created a matrix which mapped out the individual existence of the genre type to that movie.

Using kMeans, we performed a distance based cluster analysis and appended its results to the matrix.

```r
# kmean cluster with k = 29 clusters (genres)
kmeanCluster <- kmeans(distanceMatrix, 29)

# append cluster number result to matrix
distanceMatrix$cluster <- as.factor(kmeanCluster$cluster)
```

On our second attempt, we instead weighted the values of the distance matrix for each movie as a percentage. For example: If a movie was classified as 'Short' and 'Comedy', its weight would then be 0.5 in 'Short' and 0.5 in 'Comedy'.

Results

Genre Classification

The following are the results produced by each classification technique applied to the problem of genre classification.

OBLIQUE & KKN Results

Briefly, both the Oblique tree and the K Nearest Neighbors methods were found to require far more memory than could be supplied by R when applied to the sample set.

RIPPER Results

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##     0.18708389     0.02137097     0.17328171     0.20149824     0.26864181
## AccuracyPValue  McnemarPValue
##     1.00000000            NaN
```

C4.5 Results

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.1714381      0.0000000      0.1581189      0.1854004      0.2686418
## AccuracyPValue  McnemarPValue
##      1.0000000            NaN
```

NAIVE BAYES Results

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   3.462051e-01   2.494280e-01   3.291821e-01   3.635281e-01   2.686418e-01
## AccuracyPValue  McnemarPValue
##   5.655861e-21            NaN
```

Association Rules

When running apriori on the general subset produced 20 rules based on support values = 0.0001 and confidence = 0.9;

```
lhs                                                rhs
{Actor2=John Murray}                            => {Actor1=Dewey 'Pigmeat' Markham}
{Cinematographer=['Campos, Ant\xc3\xb4nio']} => {Director=Antônio Campos}
{Director=Lionel Soukaz}                        => {Cinematographer=['Soukaz, Lionel']}
{Director=John Randolph Bray}                   => {Writer=John Randolph Bray}
{Writer=Miodrag Jovanovic}                      => {Director=Miodrag Jovanovic}
{Director=Karel Zeman}                          => {Writer=Karel Zeman (screenplay)}
{Director=Branko Segovic}                       => {Writer=Stanoje Makivic, Branko Segovic}
{Director=Branko Segovic}                       => {Cinematographer=['Pavlovic, Mihailo']}
{Writer=Stanoje Makivic, Branko Segovic}        => {Cinematographer=['Pavlovic, Mihailo']}
{Director=Henry 'Hy' Mayer}                     => {Writer=Henry 'Hy' Mayer}
{Cinematographer=['Oberti, Ralf']}              => {Writer=Ralf Oberti, Rodolfo Paredes}
{Cinematographer=['Oberti, Ralf']}              => {Director=Jorge López, Ralf Oberti}
{Writer=Ralf Oberti, Rodolfo Paredes}           => {Director=Jorge López, Ralf Oberti}
{Cinematographer=['Paul, Robert W.']}           => {Director=Robert W. Paul}
{Director=Scott Eldridge}                    => {Producer=['Eldridge, Scott', 'Fried, Joe']}
{Cinematographer=['Yamazaki, Rima']}            => {Director=Rima Yamazaki}
{Director=Daniel Szczechura}                    => {Writer=Daniel Szczechura}
{Director=Daniel Szczechura}                    => {Cinematographer=['Fedak, Waclaw']}
{Writer=Milutin Kosovac}                        => {Director=Milutin Kosovac}
{Writer=Daniel Szczechura}                      => {Cinematographer=['Fedak, Waclaw']}
```

While

Bugs There are no results when attempting to rectify the "lhs" portion of the the apriori. I kept getting "Unknown Item Labels" whenever trying to filter the "lhs" portion. After further researching into this, I found that I had to specify SPECIFICALLY what each Director, Producer, Actors,etc were on the "lhs" which will prove to be infeasibly as there are many possibilities for each section.

There are no results when attempting to rectify the "Genre/Cast" portion of the apriori. I attempted to fracture "Genre" into their respective genre types and run the apriori on that. There were many errors that led up to this that eventually led to this portion of the analysis to not work entirely. My attempts at the "Genre/Cast" are in the assocRules.r file but commented out to make the general apriori function runnable.

Clustering

```
##  [1] "Short"       "Fantasy"     "Comedy"      "Drama"       "Biography"
##  [6] "Romance"     "N/A"         "Crime"       "Western"     "Thriller"
## [11] "Adventure"   "Action"      "Animation"   "Mystery"     "History"
## [16] "Documentary" "Horror"      "War"         "Musical"     "Family"
## [21] "Music"       "Sport"       "Sci-Fi"      "Film-Noir"   "Adult"
## [26] "Reality-TV"  "News"        "Game-Show"   "Talk-Show"
```

```
## [1] 723
```

Discussion

Genre Classification

What were our options for classification?

Our options for classification included: 1. imdbID 8. imdbVotes 15. imdbRating 22. Response 2. Plot 9. seriesID 16. Poster 23. Year 3. Rated 10. Season 17. Episode 24. Error 4. Title 11. Director 18. Language 25. Producer 5. Writer 12. Released 19. Country 26. Cinematographer 6. Actors 13. Awards 20. Metascore 27. Composer 7. Type 14. Genre 21. Runtime 28. CostumeDesigner

Of the 28 options listed, at the end of the day only 12 attributes ended up being used for genre classification. These attributes are:

1. Rated 7. Cinematographer
2. Title 8. Genre
3. Director 9. Writer
4. Released 10. Actors
5. Runtime 11. Country/Language
6. Year 12. Producer

The attributes that were removed were chosen because they were deemed irrelevant to determining the genre of a movie. Some of the attributes were subbed back into the overall data table to check and see if they affected the overall accuracy.

'Language' and 'Country' specifically were an interesting pair of attributes because they relayed very similar information so either or were chosen to be used during classification. 'Released' relayed almost identical information as the attribute 'Year' but was formatted as a date which threw off some of the classification algorithms, so it was removed as a result.

How did you evaluate different classification techniques?

When it came to evaluating each of our classification techniques, we computed confusion matrices that provided useful metrics related to the success of our classifiers. Through the confusion matrices we were able to see metrics like accuracy and confidence intervals.

Through comparing their accuracies we were able to see which classification techniques worked "better" than others where better is defined as higher percentage of accuracy.

We were unable to evaluate the oblique tree classification technique becuase the size of the vector created in the oblique tree algorithm exceed 1.5 gigabytes and R Studio would not store that large of a variable. So that classification method is commented out in the code as shown in the displayed script.

What measures have you taken to improve the results?

The measures we took to improve the results as much as possible include trial and error of which attribtues from the database yielded the greatest accuracy. Extreme care was also put into formatting and preprocessing the data so that the classification methods would correctly run, examples can be see in the methods and materials section. To improve the results, multiple trials could have improved results by looking at subsets of data that isolate certain areas of interest like only movies from 2010-2015.

Association Rules

The number of different roles you have in your dataset. The number of different roles involved for the usual casts apriori function is 7: Director, Producer, Writer, Actor1, Actor2, Actor3, and Actor4

What you put in the left-hand-side of the rules. As mentioned in the Results section, I was not able to get results for this section.

How you would automatically check all the possibilities. I ran apriori (the only way that R was allowing me to run) with no variables set for "lhs" and "rhs". I thought by doing the apriori this way would allow for all possibilities by not restricting those sides.

Clustering

Would clustering the movies into k' clusters where k' > k help in better categorization? And how.

In order to answer this question, we counted the quantity of unique genre combinations that were present in the dataset.

This resulted in 723 distinct genre entries used in our dataset.

Given this result, using `k' > k` as our new cluster quantity could definitely improve the movie categorization. For example, movies that frequently appeared as 'Short' and 'Comedy' could be considered for its own cluster now (a cluster for a hybrid genre, if you will).

Which clustering method works best in this case? And why.

Because we were instructed to use `k = # genres` as our cluster amount, the best clustering method was definitely to use a distance based algorithm.

A density-based cluster analysis could require more clusters to be accurate, as several of the movie data points were classified as 'hybrid' genres (such as the example mentioned in the previous question). Because of their frequency, they would be very dense and take away a cluster from one of the 'pure' genres that it used (i.e., 'Short' or 'Comedy').

Another important consideration is the speed of the algorithm. kMeans performed very quickly with respect to the dataset size.

Conclusion

If there was any lesson that was learned from this project, it was that data mining requires not only good data, but a clear understanding of that data. While we were able to build a reasonable sample dataset from the IMDb resources, the major stumbling point for each task was the ability to make valid statistical insights from the data and results.

In some cases, particularly with naive classification and ruleset-building, this didn't end up being too much of an issue, as the methods could produce relatively valid results even in the absence of clear analysis. However, when the "similar movies" task was handled, it was clear that this misunderstanding of the data made it impossible to obtain valid and usable results.

Regardless of the results, after the work done to tackle these data mining problems there are several clear avenues of approach for future endeavors. Ideally, a bigger dataset would be built, allowing for a greater variety of sampling methods to be applied, namely k-folds and stratified sampling. It seems apparent that the results for each task could have been improved by taking a stratified sample by movie years or even genres (for developing *usual casts*). Building a larger dataset could be done rather easily, it's primarily a question of time and resources. In the process of building the small "10K" dataset used here, the limiting factor was internet speed and availability, the risk of having a connection drop out mid web-sraping process prevented trying to run longer/larger python API-calls.

References & Resources

The Internet Movie Database IMDb

IMDbPY - Python API and Materials for IMDB searches IMDbPY & imdbpy2sql.py

The Open Movie Database OMDb API

Readings

- 'Automatic Video Classification: A Survey of the Literature' by Darin Brezeale and Diane J. Cook, Senior Member, IEEE IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, VOL. UNKNOWN, NO. UNKNOWN, UNKNOWN 2007
- 'Predicting Movie Success Based on IMDB Data' by Nithin VR, Pranav M, Sarath Babu PB, and Lijiya A International Journal of Data Mining Techniques and Applications Volume: 03, June 2014, Pages: 365-368

Appendix

Code Repository

The full code base can be found at IMDB-Analysis

Individual member contributions can be evaluated at IMDB-Analysis/Contributors

Student name to github usernames:

- Dax Gerts: SlightlyUnorthodox
- Christine Moore: Pocathoughts
- Carl Amko: Okma
- Denzel Mathew: dmathew93
- Aaron Silcott: aaroniey

Python & R Scripts for Dataset Creation

imdbscraper.py

```
## Title: IMDB sample database creator
## Filename: imdbscraper.py
## Author: Dax Gerts
## Description: used the OMDb API to rapidly build a clean JSON data sample of the IMDB database

import requests
import numpy as np
import json
import urllib2
import csv
import codecs

# Set sample size
sample_size = 30000

# Generate random ids
random_ids = np.random.choice(5262000,size=sample_size,replace=False)

# Generate empty list-space for results
data = [None] * sample_size

# For each id, query OMDb API and retrieve JSON object
for i in range(sample_size):
  temp_id = "%07d" % random_ids[i]
    data[i] = json.load(urllib2.urlopen('http://www.omdbapi.com/?i=tt' +
                                        temp_id + '&plot=short&r=json'))
    print("QUERY#"+str(i))

# Dump all JSON objects in list to single object, write to file
with codecs.open('imdb_re_30k_sample.txt','w','utf8') as f:
    json.dump(data, f)
```

castscrape.py

```
## Title: IMDB cast rebuild
## Filename: castscrape.py
## Author: Dax Gerts
## Description: reconstructs additional cast features by movieID in sample set

import imdb
import csv
import re
import csv

ia = imdb.IMDb()

with open("clean10Kimdbids.csv") as f:
  reader = csv.reader(f)
    ids = map(tuple, reader)

with open('imdb_10K_cast_plus.csv','wb') as csvfile:
    idwriter = csv.writer(csvfile, delimiter=',')
    #for i in 1:len(ids):
    for i in range(1,len(ids)):
        #Assign temp string
        temp = str(ids[i])

        #Use regex to cut off ends of temp string
        temp = re.sub('\'\,\)$','',temp)
        temp = re.sub('\(\'tt','',temp)
        print(temp)

        #Identify movie by id
        movie = ia.get_movie(temp)

        #Attemp to retrieve producer list
        try:
            #Read producer as string
            producer = str(movie['producer'])

            #Repeatedly extract inline names
            producer = re.findall('_(.+?)_>',producer)

        except KeyError:
            #Fail to read producer, write "NA"
            producer = "NA"

        try:
            #Read cinema__ as string
            cinema = str(movie['cinematographer'])

            #Repeatedly extract inline names
            cinema = re.findall('_(.+?)_>',cinema)

        except KeyError:
```

```
            #Fail to read, write "NA"
            cinema = "NA"

        try:
            #Read composers as string
            composer = str(movie['composer'])

            #Repeatedly read inline names
            composer = re.findall('_(.+?_>',composer)
        except KeyError:
            #Fail to read, write "NA"
            composer = "NA"

        try:
            #Read costume-designer as tring
            costume = str(movie['costume-designer'])

            #Repeatedly read inline names
            costume = re.findall('_(.+?_>',costume)
        except KeyError:
            #Fail to read, write "NA"
            costume = "NA"

        #Write producer string to file
        idwriter.writerow([temp,producer,cinema,composer,costume])
```

datasetCreation.r

```
# Project: CIS4930 Group Project
# Authors: Dax Gerts, ...
# Date: 23 November 2015
# Description: Methods for creation of IMDB dataset

#Dynamically load/install required packages
ready <- FALSE
loadPackages1 <- function() {
  if( require(R.utils) == FALSE) { install.packages("R.utils") }
  if( require(stringr) == FALSE) { install.packages("stringr") }
  if( require(data.table) == FALSE) { install.packages("data.table") }
  if( require(jsonlite) == FALSE) { install.packages("jsonlite") }
  if( require(plyr) == FALSE) { install.packages("dplyr")}
  if( require(ggplot2) == FALSE) { install.packages("ggplot2")}
  ready <- TRUE
}
while(ready == FALSE) { ready <- loadPackages1() }

#Set seed
set.seed(7131)

# Runtime reformat, string -> minutes (integer)
timeSet <- function(x) {
  if(length(x == 2)) {
```

```
    x = as.numeric(x)
  } else if (length(x == 4))  {
    x = (as.numeric(substr(x,1,1))*60)+as.numeric(substr(x,2,3))
  } else {
    x = as.numeric("0")
  }
  x
}

# Reformat OMDB JSON queries as csv
jsonToCsv <- function(filename = "imdb_30K_sample.json",write=TRUE,
                      csvfile = "imdb_30K_sample.csv") {
  data <- fromJSON(txt=as.character(filename))
  if(write==TRUE) {
    write.csv(data,file=csvfile,row.names = FALSE)
  }
  data
}

# Runtime reformat, string -> minutes (integer)
timeSet <- function(x) {
  if(length(x == 2)) {
    x = as.numeric(x)
  } else if (length(x == 4))  {
    x = (as.numeric(substr(x,1,1))*60)+as.numeric(substr(x,2,3))
  } else {
    x = as.numeric("0")
  }
  x
}


# Clean data set, set proper types, drop invalid rows,
preprocessing <- function(data) {

  # Step 1: Variable type checking

  # 1.1 Plot - char (fine as is)

  # 1.2 Rated - factor (49 levels) (sparse)
  data$Rated <- as.factor(data$Rated)
  data$Rated[data$Rated=="N/A"] <- NA
  summary(data$Rated)

  # 1.3 Title - char (fine as is)

  # 1.4 Writer - factor (>10000 levels)
  data$Writer <- as.factor(data$Writer)
  data$Writer[data$Writer=="N/A"] <- NA
  summary(data$Writer)

  # 1.5 Actors - list ==> factors
  data$Actors <- strsplit(data$Actors,", ")
```

```
data$Actors <- sapply(data$Actors,'[',seq(max(sapply(data$Actors,length))),
                      simplify=FALSE)
data$Actors[1:5]

### BEGIN ACTORS FIX

# Description: Actors was broken into 4 columns and cast as factors

#Split actors to temp unique columns
temp<- ldply(data$Actors)
colnames(temp) <- c('Actor1','Actor2','Actor3','Actor4')

#Reassign actors to working data frame under new names
#("Actor1","Actor2","Actor3","Actor4")
data$Actor1 <- temp[1]
data$Actor2 <- temp[2]
data$Actor3 <- temp[3]
data$Actor4 <- temp[4]

#Delete temporary data frame
rm(temp)

#Drop defunct actors column
data <- data[,-c(5)]

#Catch alternate NAs
data$Actor1[data$Actor1=="N/A"] <- NA
data$Actor2[data$Actor2=="N/A"] <- NA
data$Actor3[data$Actor3=="N/A"] <- NA
data$Actor4[data$Actor4=="N/A"] <- NA

#Unlist and set actors as factors
data$Actor1 <- as.factor(unlist(data$Actor1))
data$Actor2 <- as.factor(unlist(data$Actor2))
data$Actor3 <- as.factor(unlist(data$Actor3))
data$Actor4 <- as.factor(unlist(data$Actor4))

### END ACTORS FIX

### START CLEANING UP PRODUCER AND CINEMATOGRAPHER
### producer first

# single quote removal
data$Producer <- lapply(data$Producer, function(x){
  gsub("\'", "", x)#
})

# double quote removal
data$Producer <- lapply(data$Producer, function(x){
  gsub("\"", "", x)#
})

# remove brackets
```

```
data$Producer <- lapply(data$Producer, function(x){
  gsub("\\[", "", x)#
})

data$Producer <- lapply(data$Producer, function(x){
  gsub("\\]", "", x)#
})

data$Producer <- lapply(data$Producer, function(x){
  strsplit(x,",")
})

firstName <- lapply(data$Producer, function(x){
  x[[1]][2]
})

lastName <- lapply(data$Producer, function(x){
  x[[1]][1]
})

data$Producer <- paste(firstName, lastName)

### end fixing of producer

### fix cinematographer

# single quote removal
data$Cinematographer <- lapply(data$Cinematographer, function(x){
  gsub("\'", "", x)#
})

# double quote removal
data$Cinematographer <- lapply(data$Cinematographer, function(x){
  gsub("\"", "", x)#
})

# remove brackets
data$Cinematographer <- lapply(data$Cinematographer, function(x){
  gsub("\\[", "", x)#
})

data$Cinematographer <- lapply(data$Cinematographer, function(x){
  gsub("\\]", "", x)#
})

data$Cinematographer <- lapply(data$Cinematographer, function(x){
  strsplit(x,",")
})

firstName <- lapply(data$Cinematographer, function(x){
  x[[1]][2]
})
```

```r
lastName <- lapply(data$Cinematographer, function(x){
  x[[1]][1]
})

data$Cinematographer <- paste(firstName, lastName)

### end fixing cinematographer

### END OF CLEANING UP PRODUCER AND CINEMATOGRAPHER

# 1.6 Type - factor
data$Type <- as.factor(data$Type)
data$Type[data$Type=="N/A"] <- NA
summary(data$Type)

# 1.7 imdbVotes - numeric
data$imdbVotes <- as.numeric(data$imdbVotes)
summary(data$imdbVotes)

# 1.8 seriesID - char (fine as is)

# 1.9 Season
data$Season <- as.numeric(data$Season)
summary(data$Season)

# 1.10 Director - factor (> 10000 levels)
data$Director <- as.factor(data$Director)
data$Director[data$Director=="N/A"] <- NA
summary(data$Director)

# 1.11 Released - date
data$Released <- as.Date(data$Released,"%d %b %Y")

# 1.12 Awards - to finicky to do anything with now

# 1.13 Genre - fact list
data$Genre <- strsplit(data$Genre,", ")
data$Genre <- sapply(data$Genre,'[',seq(max(sapply(data$Genre,length))),simplify=FALSE)
data$Genre[1:5]

# 1.14 imdbRating - numeric
data$imdbRating <- as.numeric(data$imdbRating)

# 1.15 Poster - char (fine as is)

# 1.16 Episode - numeric
data$Episode <- as.numeric(data$Episode)

# 1.17 Language - factor list
data$Language <- strsplit(data$Language,", ")
data$Language <- sapply(data$Language,'[',seq(max(sapply(data$Language,length))),
                        simplify=FALSE)
data$Language[1:5]
```

```r
# 1.18 Country
data$Country <- strsplit(data$Country,", ")
data$Country <- sapply(data$Country,'[',seq(max(sapply(data$Country,length))),
                       simplify=FALSE)
data$Country[1:5]

# 1.19 Runtime - numeric (calls setTime function to convert values)
data$Runtime <- gsub("[^0-9]"," ",data$Runtime)
for(i in 1:length(data$Runtime)) {
  data$Runtime[i] = timeSet(data$Runtime[i])
}
data$Runtime <- as.numeric(data$Runtime)
summary(data$Runtime)

# 1.20 imdbID (fine as is)

# 1.21 Metascore
data$Metascore <- as.factor(data$Metascore)
data$Metascore[data$Metascore=="N/A"] <- NA
levels(data$Metascore)

# 1.22 Response (irrelevant)

# 1.23 Year - as numeric (only takes first year in range)
data$Year <- as.numeric(gsub("\\-.*","",data$Year))
summary(data$Year)

# 1.24 Error (fine as is)

# 2 Load and prep extra cast values

# 2.1 Name columns
temp <- read.csv("imdb_10K_cast_plus.csv",header=FALSE)
colnames(temp) <- c("imdbID","Producer","Cinematographer","Composer","CostumeDesigners")

# 2.2 Format IDs to match
temp$imdbID <- sapply(temp$imdbID,function(x) sprintf("%07d",x))
temp$imdbID <- paste("tt",as.character(temp$imdbID),sep="")

# 2.3 Merge new values to table
data <- merge(data,temp,by="imdbID",all=TRUE)

# 2.4 Format new variables
data$Producer <- as.factor(data$Producer)
data$Cinematographer <- as.factor(data$Cinematographer)
data$Composer <- as.factor(data$Composer)
data$CostumeDesigners <- as.factor(data$CostumeDesigners)
# 3 Prepare valid data

# 3.1 Drop non-movie entries
data <- data[data$Type == "movie",]

# 3.2 Drop bad/"N/A" entries
```

```
  data <- data[!(is.na(as.factor(data$Title))),]

  # 3.3 Save table as R object
  saveRDS(data,"clean10Kdataset.rds")

  # 3.4 Write clean csv
  csvData <- data.frame(lapply(data,as.character),stringsAsFactors=FALSE)
  write.csv(csvData,file="clean10Kdataset.csv",row.names = FALSE)

  # 3.5 Return output table
  data
}
```

Data Mining Tasks

Genre Classification (GenreClassification.r)

```
#############################################################################################
#                                    Dynamically Load/Install Packages
#############################################################################################
#Dynamically load/install required packages
ready <- FALSE
loadPackages3 <- function() {
  if( require(R.utils) == FALSE) { install.packages("R.utils") }
  if( require(tidyr) == FALSE) { install.packages("tidyr") }
  if( require(dplyr) == FALSE) { install.packages("dplyr") }
  if( require(e1071) == FALSE) { install.packages("e1071") }
  if( require(kknn) == FALSE) { install.packages("klaR")}
  if( require(RWeka) == FALSE) { install.packages("RWeka") }
  if( require(oblique.tree) == FALSE) { install.packages("oblique.tree") }
  if (require(caret) == FALSE) { install.packages("caret") }
  ready <- TRUE
}
while(ready == FALSE) { ready <- loadPackages3() }

set.seed(1911)

#############################################################################################
#                                    IMDB DATA SET
#############################################################################################

#access the imdb data set we created
movieData <- readRDS("clean10Kdataset.rds")

# clean up the movie data set
# description, rating, imdbVotes, seriesID, season, type, awards, imdbRating, Poster,
# episode, released, imdbID, Metascore, response, language, and year
# were removed from table for analysis because they did not provide information helpful
# to classification

# check variables
```

```r
names(movieData)

# remove unneeded variables
movieData <- movieData[,-c(1,2,6,7,8, 9, 11, 12, 14, 15, 16, 17, 20, 21, 23, 30 ,31)]

# confirm variable removal
names(movieData)

# now are datasets contain only the important variables pertinent to classification
# episodes taken about because all values are N/A for this dataset

#country is going to be cut down to 1 country
movieData$Country <- lapply(movieData$Country,"[",1)

#movieData$Language <- lapply(movieData$Language,"[",1)

# Cut genre down to 2 items
movieData$Genre <- lapply(movieData$Genre,"[",1:2)

# Cleans writer entries
movieData$Writer <- lapply(movieData$Writer, function(x) {
  gsub( " *\\(.*?\\) *", "", x)
  })

tgemp <- movieData

# Reordering first and last name and removing exraneous separators
# First step is to remove single quote and double quotes
# yes, soempeople's name may have a single quote in it however numbers are so small
# it wont affect the data

######################################## fix producer ########################################

# single quote removal
movieData$Producer <- lapply(movieData$Producer, function(x){
  gsub("\'", "", x)#
})

# double quote removal
movieData$Producer <- lapply(movieData$Producer, function(x){
  gsub("\"", "", x)#
})

# remove brackets
movieData$Producer <- lapply(movieData$Producer, function(x){
  gsub("\\[", "", x)#
})

movieData$Producer <- lapply(movieData$Producer, function(x){
  gsub("\\]", "", x)#
})

movieData$Producer <- lapply(movieData$Producer, function(x){
```

```
strsplit(x,",")
})

firstName <- lapply(movieData$Producer, function(x){
 x[[1]][2]
})

lastName <- lapply(movieData$Producer, function(x){
  x[[1]][1]
})

movieData$Producer <- paste(firstName, lastName)

########################### end fixing of producer ########################

########################### fix cinematographer ############################

# single quote removal
movieData$Cinematographer <- lapply(movieData$Cinematographer, function(x){
  gsub("\'", "", x)#
})

# double quote removal
movieData$Cinematographer <- lapply(movieData$Cinematographer, function(x){
  gsub("\"", "", x)#
})

# remove brackets
movieData$Cinematographer <- lapply(movieData$Cinematographer, function(x){
  gsub("\\[", "", x)#
})

movieData$Cinematographer <- lapply(movieData$Cinematographer, function(x){
  gsub("\\]", "", x)#
})

movieData$Cinematographer <- lapply(movieData$Cinematographer, function(x){
  strsplit(x,",")
})

firstName <- lapply(movieData$Cinematographer, function(x){
  x[[1]][2]
})

lastName <- lapply(movieData$Cinematographer, function(x){
  x[[1]][1]
})

movieData$Cinematographer <- paste(firstName, lastName)

########################### end fixing of Cinematographer ######################

# Create unique rows for each genre
```

```
movieData <- unnest(movieData,Genre)
movieData <- unnest(movieData, Writer)
#movieData <- unnest(movieData, Actors)
movieData <- unnest(movieData, Country)
#movieData <- unnest(movieData, Language)

# Drop rows with NA genre because we can't classify these
movieData <- movieData[!(is.na(as.factor(movieData$Genre))),]
movieData <- movieData[!movieData$Genre == "N/A",]
movieData$Genre <- as.factor(movieData$Genre)

# Cast as factor
movieData$Rated <- as.factor(movieData$Rated)
movieData$Title <- as.factor(movieData$Title)
movieData$Director <- as.factor(movieData$Director)
movieData$Actor1 <- as.factor(movieData$Actor1)
movieData$Actor2 <- as.factor(movieData$Actor2)
movieData$Actor3 <- as.factor(movieData$Actor3)
movieData$Actor4 <- as.factor(movieData$Actor4)
movieData$Producer <- as.factor(movieData$Producer)
movieData$Cinematographer <- as.factor(movieData$Cinematographer)
movieData$Writer <- as.factor(movieData$Writer)
movieData$Country <- as.factor(movieData$Country)
#movieData$Language <- as.factor(movieData$Language)


#insert pruning methods here
#movieData <- movieData[movieData$Language == 'English',]
#movieData <- movieData[movieData$Country == 'USA',]

# Create test and training sets
part <- createDataPartition(movieData$Genre,p=0.8,list=FALSE)
training <- movieData[part,]
test <- movieData[-part,]

# Save training/test data for later
saveRDS(training,file="training.rds")
saveRDS(test,file="test.rds")

### TEST AND TRAINING SETS ARE A GO

# For future use, make new data table without the genre label in it
movieDataWOutGenre <- test[,-12]

###################### TO DO
# try making them factors
#figure out if unnest is best way to go
#try differnet attribute comobinations to seewhats best

#RIPPER CLASSIFIER
ripperModelMovie <- JRip(Genre~., data = training)
ripperPredictionsMovie <- predict(ripperModelMovie, movieDataWOutGenre)
# summarize results
```

```
ripCMMovie <- confusionMatrix(ripperPredictionsMovie, test$Genre)

names(movieData)

#C4.5 CLASSIFICATION
c45ModelMovie <- J48(Genre~., data = training)
c45ModelPredictionsMovie <- predict(c45ModelMovie, movieDataWOutGenre)
c45CMMovie <- confusionMatrix(c45ModelPredictionsMovie, test$Genre)


#OBLIQUE CLASSIFICATION
#obliqueModelMovie <- oblique.tree(formula = Genre~., data = training,
#         oblique.splits = "only")
#obliqueModelPredictionsMovie <- predict(obliqueModelMovie, test)
#obCMMovie <- confusionMatrix(colnames(obliqueModelPredictionsMovie)
#         [max.col(obliqueModelPredictionsMovie)], test$Genre)


#NAIVE BAYES CLASSIFIER
# train a naive bayes model
naiveBayesModel <- naiveBayes(Genre~., data=training)
# make predictions
predictions <- predict(naiveBayesModel, movieDataWOutGenre)
# summarize results
nbCMMovie <- confusionMatrix(predictions, test$Genre)


#KNN CLASSIFIER
kkModel <- kknn(Genre~., test = test, train = training, distance = 1,
                kernel = "triangular")
knnPredictions <- predict(kkModel, movieDataWOutGenre)
knnCMMovie <- confusionMatrix(knnPredictions, test$Genre)


##############################################################################################
#                                FINAL DISPLAYED RESULTS
##############################################################################################
#.............................CONFUSION MATRICES FOR IRIS.....................................
# RIPPER
print(ripCMMovie)

#C4.5
print(c45CMMovie)

#OBLIQUE
#print(obCMMovie)

#NAIVE BAYES
print(nbCMMovie)

#KNN
print(knnCMMovie)
```

```
################## IGNORE BELOW FOR NOW

#get iris accuracies
accRipMovie <- ripCMMovie$overall[1]
accC45Movie <- c45CMMovie$overall[1]
accOBMovie <- obCMMovie$overall[1]
accNBMovie <- nbCMMovie$overall[1]
accKNNMovie <- knnCMMovie$overall[1]


accuracyMatrix <- matrix(c(accRipMovie, accC45Movie, accOBMovie, accNBMovie, accKNNMovie),
                         ncol = 2, byrow = TRUE)
colnames(accuracyMatrix) <- c("Movie Data Accuracies", "Life Expectancy Data Accuracies")
rownames(accuracyMatrix) <- c("Ripper", "C4.5", "Oblique Tree", "Naive Bayes", "KNN")
accuracyMatrix <- as.table(accuracyMatrix)
print(accuracyMatrix)
```

"Usual cast" rule-finding (assocRules.r)

```
# Project: CIS4930 Group Project
# Authors: Dax Gerts, Denzel
# Date: 23 November 2015
# Description: Procedure for detecting "usual casts" (association rule mining)

#Source assocData prep files
# source('assocDatasetCreation.r')

#Dynamically load/install required packages

ready <- FALSE
loadPackages3 <- function() {
  if( require(arules) == FALSE) { install.packages("arules") }
  if( require(reshape2) == FALSE) { install.packages("reshape2") }
  if( require(stringr) == FALSE) { install.packages("stringr") }
  if( require(tidyr) == FALSE) { install.packages("tidyr") }
  if( require(base) == FALSE) { install.packages("base") }
  ready <- TRUE
}
while(ready == FALSE) { ready <- loadPackages3() }

set.seed(7131)

#define redundant rules function
redundantRules <- function(rules) {
  sub <- is.subset(rules,rules)
  sub[lower.tri(sub,diag=T)] <- NA
  red <- colSums(sub,na.rm=T) >= 1
  rrules <- rules[!red]
  rrules
}
```

```
makeEvenSet <- function(x){
  x$Action <- 0
  x$Adventure <- 0
  x$Adult <- 0
  x$Animation <- 0
  x$Biography <- 0
  x$Comedy <- 0
  x$Crime <- 0
  x$Documentary <- 0
  x$Drama <- 0
  x$Family <- 0
  x$Fantasy <- 0
  x$FilmNoir <- 0
  x$GameShow <- 0
  x$History <- 0
  x$Horror <- 0
  x$Music <- 0
  x$Musical <- 0
  x$Mystery <- 0
  x$None <- 0
  x$News <- 0
  x$RealityTV <- 0
  x$Romance <- 0
  x$SciFi <- 0
  x$Short <- 0
  x$Sport <- 0
  x$TalkShow <- 0
  x$Thriller <- 0
  x$War <- 0
  x$Western <- 0

  for(i in 1:nrow(x)){
    coll <- x[i,"Genre"]
    value <- 1
    for(j in 1:length(coll[[1]])){
      if (is.na(coll[[1]][j])){
        next
      }
      if (coll[[1]][j] == "Action"){
        x[i,"Action"] = value
      }
      if (coll[[1]][j] == "Adventure"){
        x[i,"Adventure"] = value
      }
      if (coll[[1]][j] == "Adult"){
        x[i,"Adult"] = value
      }
      if (coll[[1]][j] == "Animation"){
        x[i,"Animation"] = value
      }
      if (coll[[1]][j] == "Biography"){
        x[i,"Biography"] = value
      }
```

```
      if (coll[[1]][j] == "Comedy"){
        x[i,"Comedy"] = value
      }
      if (coll[[1]][j] == "Crime"){
        x[i,"Crime"] = value
      }
      if (coll[[1]][j] == "Documentary"){
        x[i,"Documentary"] = value
      }
      if (coll[[1]][j] == "Drama"){
        x[i,"Drama"] = value
      }
      if (coll[[1]][j] == "Family"){
        x[i,"Family "] = value
      }
      if (coll[[1]][j] == "Fantasy"){
        x[i,"Fantasy"] = value
      }
      if (coll[[1]][j] == "Film-Noir"){
        x[i,"FilmNoir"] = value
      }
      if (coll[[1]][j] == "Game-Show"){
        x[i,"GameShow"] = value
      }
      if (coll[[1]][j] == "History"){
        x[i,"History"] = value
      }
      if (coll[[1]][j] == "Horror"){
        x[i,"Horror"] = value
      }
      if (coll[[1]][j] == "Music"){
        x[i,"Music"] = value
      }
      if (coll[[1]][j] == "Musical"){
        x[i,"Musical"] = value
      }
      if (coll[[1]][j] == "Mystery"){
        x[i,"Mystery "] = value
      }
      if (coll[[1]][j] == "N/A"){
        x[i,"None"] = value
      }
      if (coll[[1]][j] == "News"){
        x[i,"News "] = value
      }
      if (coll[[1]][j] == "Reality-TV"){
        x[i,"RealityTV"] = value
      }
      if (coll[[1]][j] == "Romance"){
        x[i,"Romance"] = value
      }
      if (coll[[1]][j] == "Sci-Fi"){
        x[i,"SciFi"] = value
```

```
      }
      if (coll[[1]][j] == "Sport"){
        x[i,"Sport"] = value
      }
      if (coll[[1]][j] == "Short"){
        x[i,"Short"] = value
      }
      if (coll[[1]][j] == "Talk-Show"){
        x[i,"TalkShow"] = value
      }
      if (coll[[1]][j] == "Thriller"){
        x[i,"Thriller"] = value
      }
      if (coll[[1]][j] == "War"){
        x[i,"War"] = value
      }
      if (coll[[1]][j] == "Western"){
        x[i,"Western"]  = value
      }
    }
  }
  y <- x [,c("Action", "Adventure", "Adult", "Animation", "Biography", "Comedy",
             "Crime", "Documentary", "Drama", "Family", "Fantasy", "FilmNoir",
             "GameShow" ,"History", "Horror", "Music", "Musical", "Mystery",
             "None","News","RealityTV","Romance", "SciFi", "Sport", "Short",
             "TalkShow", "Thriller", "War", "Western")]
  return (y)
}

#Read in the RDS
assocData <- readRDS("clean10Kdataset.rds")


#Get the Regular Association Rules Subset
assocDataSubset <- assocData[,c("Director","Producer","Writer","Cinematographer","Actor1"

#Build ruleset
movieRules <- apriori(assocDataSubset,parameter = list(support = 0.0001,confidence = 0.9));
# genreRules <- apriori(assocData,parameter = list(support = 0.01,confidence = 0.9));

#Removes redundant rules
uniqueMovieRules <- redundantRules(movieRules)
# uniqueGenreRules <- redundantRules(genreRules)

#Sorts trimmed rules by lift
sortedUniqueMovieRules <- sort(uniqueMovieRules, by = "lift")
# sortedUniqueGenreRules <- sort(uniqueGenreRules, by = "lift")
inspect(head(sortedUniqueMovieRules,20))
```

Similar movie clustering (clustering.r)

```
# Project: CIS4930 Group Project
# Authors: Dax Gerts, ...
# Date: 23 November 2015
# Description: Procedure for finding similar movies (clustering)

#Dynamically load/install required packages
ready <- FALSE
loadPackages2 <- function() {
  if( require(R.utils) == FALSE) { install.packages("R.utils") }
  if( require(ggplot2) == FALSE) { install.packages("ggplot2") }
  ready <- TRUE
}
while(ready == FALSE) { ready <- loadPackages2() }

# Load dataset
data <- readRDS("clean10Kdataset.rds")

#install.packages("cluster")
library(cluster)

#subset dataset to Genre list
dataGenre <- data[, 13]

#Prune the data to remove excess collection space
dataPruned <- lapply(dataGenre, function(x) {
  # Make the collection 'unique', removing all the blank collection spots
  x <- unique(x)

  # chop off the last 'NA' value
  x <- x[-length(x)]

})

#list unique genre entries
unique(unlist(dataPruned))

# count unique genres
length(unique(dataPruned))

#preprocessing functions
makeEvenSet <- function(x){
  x$Action <- 0
  x$Adventure <- 0
  x$Adult <- 0
  x$Animation <- 0
  x$Biography <- 0
  x$Comedy <- 0
  x$Crime <- 0
  x$Documentary <- 0
  x$Drama <- 0
  x$Family <- 0
```

```
x$Fantasy <- 0
x$FilmNoir <- 0
x$GameShow <- 0
x$History <- 0
x$Horror <- 0
x$Music <- 0
x$Musical <- 0
x$Mystery <- 0
x$None <- 0
x$News <- 0
x$RealityTV <- 0
x$Romance <- 0
x$SciFi <- 0
x$Short <- 0
x$Sport <- 0
x$TalkShow <- 0
x$Thriller <- 0
x$War <- 0
x$Western <- 0
for(i in 1:nrow(x)){
  coll <- x[i,"Genre"]
  value <- 1
  for(j in 1:length(coll[[1]])){
    if (is.na(coll[[1]][j])){
      next
    }
    if (coll[[1]][j] == "Action"){
      x[i,"Action"] = value
    }
    if (coll[[1]][j] == "Adventure"){
      x[i,"Adventure"] = value
    }
    if (coll[[1]][j] == "Adult"){
      x[i,"Adult"] = value
    }
    if (coll[[1]][j] == "Animation"){
      x[i,"Animation"] = value
    }
    if (coll[[1]][j] == "Biography"){
      x[i,"Biography"] = value
    }
    if (coll[[1]][j] == "Comedy"){
      x[i,"Comedy"] = value
    }
    if (coll[[1]][j] == "Crime"){
      x[i,"Crime"] = value
    }
    if (coll[[1]][j] == "Documentary"){
      x[i,"Documentary"] = value
    }
    if (coll[[1]][j] == "Drama"){
      x[i,"Drama"] = value
    }
```

```
      if (coll[[1]][j] == "Family"){
        x[i,"Family "] = value
      }
      if (coll[[1]][j] == "Fantasy"){
        x[i,"Fantasy"] = value
      }
      if (coll[[1]][j] == "Film-Noir"){
        x[i,"FilmNoir"] = value
      }
      if (coll[[1]][j] == "Game-Show"){
        x[i,"GameShow"] = value
      }
      if (coll[[1]][j] == "History"){
        x[i,"History"] = value
      }
      if (coll[[1]][j] == "Horror"){
        x[i,"Horror"] = value
      }
      if (coll[[1]][j] == "Music"){
        x[i,"Music"] = value
      }
      if (coll[[1]][j] == "Musical"){
        x[i,"Musical"] = value
      }
      if (coll[[1]][j] == "Mystery"){
        x[i,"Mystery "] = value
      }
      if (coll[[1]][j] == "N/A"){
        x[i,"None"] = value
      }
      if (coll[[1]][j] == "News"){
        x[i,"News "] = value
      }
      if (coll[[1]][j] == "Reality-TV"){
        x[i,"RealityTV"] = value
      }
      if (coll[[1]][j] == "Romance"){
        x[i,"Romance"] = value
      }
      if (coll[[1]][j] == "Sci-Fi"){
        x[i,"SciFi"] = value
      }
      if (coll[[1]][j] == "Sport"){
        x[i,"Sport"] = value
      }
      if (coll[[1]][j] == "Short"){
        x[i,"Short"] = value
      }
      if (coll[[1]][j] == "Talk-Show"){
        x[i,"TalkShow"] = value
      }
      if (coll[[1]][j] == "Thriller"){
        x[i,"Thriller"] = value
```

```
      }
      if (coll[[1]][j] == "War"){
        x[i,"War"] = value
      }
      if (coll[[1]][j] == "Western"){
        x[i,"Western"]  = value
      }
    }
  }
  y <- x [,c("Action", "Adventure", "Adult", "Animation", "Biography", "Comedy",
             "Crime", "Documentary", "Drama", "Family", "Fantasy", "FilmNoir",
             "GameShow" ,"History", "Horror", "Music", "Musical", "Mystery",
             "None","News","RealityTV","Romance", "SciFi", "Sport", "Short",
             "TalkShow", "Thriller", "War", "Western")]
  return (y)
}

makePerSet <- function(x){
  x$Action <- 0
  x$Adventure <- 0
  x$Adult <- 0
  x$Animation <- 0
  x$Biography <- 0
  x$Comedy <- 0
  x$Crime <- 0
  x$Documentary <- 0
  x$Drama <- 0
  x$Family <- 0
  x$Fantasy <- 0
  x$FilmNoir <- 0
  x$GameShow <- 0
  x$History <- 0
  x$Horror <- 0
  x$Music <- 0
  x$Musical <- 0
  x$Mystery <- 0
  x$None <- 0
  x$News <- 0
  x$RealityTV <- 0
  x$Romance <- 0
  x$SciFi <- 0
  x$Short <- 0
  x$Sport <- 0
  x$TalkShow <- 0
  x$Thriller <- 0
  x$War <- 0
  x$Western <- 0
  for(i in 1:nrow(x)){
    coll <- x[i,"Genre"]
    den = 1
    for(q in 1:length(coll[[1]])){
      if(is.na(coll[[1]][q])){
      } else {
```

```
      den <- q
    }
  }
  value <- 1/den
  for(j in 1:length(coll[[1]])){
    if (is.na(coll[[1]][j])){
      next
    }
    if (coll[[1]][j] == "Action"){
      x[i,"Action"] = value
    }
    if (coll[[1]][j] == "Adventure"){
      x[i,"Adventure"] = value
    }
    if (coll[[1]][j] == "Adult"){
      x[i,"Adult"] = value
    }
    if (coll[[1]][j] == "Animation"){
      x[i,"Animation"] = value
    }
    if (coll[[1]][j] == "Biography"){
      x[i,"Biography"] = value
    }
    if (coll[[1]][j] == "Comedy"){
      x[i,"Comedy"] = value
    }
    if (coll[[1]][j] == "Crime"){
      x[i,"Crime"] = value
    }
    if (coll[[1]][j] == "Documentary"){
      x[i,"Documentary"] = value
    }
    if (coll[[1]][j] == "Drama"){
      x[i,"Drama"] = value
    }
    if (coll[[1]][j] == "Family"){
      x[i,"Family "] = value
    }
    if (coll[[1]][j] == "Fantasy"){
      x[i,"Fantasy"] = value
    }
    if (coll[[1]][j] == "Film-Noir"){
      x[i,"FilmNoir"] = value
    }
    if (coll[[1]][j] == "Game-Show"){
      x[i,"GameShow"] = value
    }
    if (coll[[1]][j] == "History"){
      x[i,"History"] = value
    }
    if (coll[[1]][j] == "Horror"){
      x[i,"Horror"] = value
    }
```

```
      if (coll[[1]][j] == "Music"){
        x[i,"Music"] = value
      }
      if (coll[[1]][j] == "Musical"){
        x[i,"Musical"] = value
      }
      if (coll[[1]][j] == "Mystery"){
        x[i,"Mystery "] = value
      }
      if (coll[[1]][j] == "N/A"){
        x[i,"None"] = value
      }
      if (coll[[1]][j] == "News"){
        x[i,"News "] = value
      }
      if (coll[[1]][j] == "Reality-TV"){
        x[i,"RealityTV"] = value
      }
      if (coll[[1]][j] == "Romance"){
        x[i,"Romance"] = value
      }
      if (coll[[1]][j] == "Sci-Fi"){
        x[i,"SciFi"] = value
      }
      if (coll[[1]][j] == "Sport"){
        x[i,"Sport"] = value
      }
      if (coll[[1]][j] == "Short"){
        x[i,"Short"] = value
      }
      if (coll[[1]][j] == "Talk-Show"){
        x[i,"TalkShow"] = value
      }
      if (coll[[1]][j] == "Thriller"){
        x[i,"Thriller"] = value
      }
      if (coll[[1]][j] == "War"){
        x[i,"War"] = value
      }
      if (coll[[1]][j] == "Western"){
        x[i,"Western"]  = value
      }
    }
  }
  y <- x [,c("Action", "Adventure", "Adult", "Animation", "Biography", "Comedy",
             "Crime", "Documentary", "Drama", "Family", "Fantasy", "FilmNoir",
             "GameShow" ,"History", "Horror", "Music", "Musical", "Mystery",
             "None","News","RealityTV","Romance", "SciFi", "Sport", "Short",
             "TalkShow", "Thriller", "War", "Western")]
  return (y)
}
```

```
#creating the data sets for comparison
evenDistanceMatrix <- makeEvenSet(data)
percentageDistanceMatrix <- makePerSet(data)

# kmean cluster with k = 29 clusters (genres)
evenKmeanCluster <- kmeans(evenDistanceMatrix, 29)
percentKmeanCluster <- kmeans(percentageDistanceMatrix, 29)


# append cluster number result to matrix
evenDistanceMatrix$cluster <- as.factor(evenKmeanCluster$cluster)
percentageDistanceMatrix$cluster <- as.factor(percentKmeanCluster$cluster)

##output the data to csv
write.csv(evenDistanceMatrix, "evenDataAppended.csv")
write.csv(percentageDistanceMatrix, "percentageDataAppended.csv")
csvDataEv <- data.frame(lapply(evenKmeanCluster$withinss,as.character),
                        stringsAsFactors=FALSE)
write.csv(csvDataEv,file="clusterev.csv",row.names = FALSE)
csvDataPe <- data.frame(lapply(percentageDistanceMatrix,as.character),
                        stringsAsFactors=FALSE)
write.csv(csvDataPe,file="clusterpc.csv",row.names = FALSE)

#distMatrix <- dist(distanceMatrix, method = "manhattan")


# Questions

# Which clustering method works best in this case? And why.

# Would clustering the movies into k' clusters where k' > k
#help in better categorization? And how.
```