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

The packages and required workspace prepartion steps are shown below for reproducibility purposes.

```
#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(ggplot2) == FALSE) { install.packages("ggplot2") }
  if( require(plyr) == FALSE) { install.packages("dplyr")}
  ready <- TRUE
}
while(ready == FALSE) { ready <- loadPackages1() }

#Set seed
set.seed(7131)
```

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: Employ 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.

As seen below, the *imdbscraper.py* script was set to a sample size of 30 thousand values (for reasons explained later) 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.

```
## Title: IMDB sample database creator
## Filename: imdbscraper.py
## Author: Dax Gerts

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)
```

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 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*.

```
## Title: IMDB sample database creator
## Filename: castscrape.py
## Author: Dax Gerts

import imdb
import csv
import re
import csv

# Create IMDB API object
ia = imdb.IMDb()

# Read pre-genereated list of sample IMDbIDs
with open("clean10Kimdbids.csv") as f:
  reader = csv.reader(f)
    ids = map(tuple, reader)

# Open output file and begin writing results
with open('imdb_10K_cast_plus.csv','wb') as csvfile:
    idwriter = csv.writer(csvfile, delimiter=',')

  # Traverse list of 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])
```

Preprocessing

Load Dataset Into R

Having created a working dataset, the following 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.

```
# 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
}
data <- jsonToCsv()
```

Variables

The initial OMDb query returned the following attributes for each movie.

```
names(data)
```

```
##  [1] "Plot"       "Rated"      "Title"      "Writer"     "Actors"
##  [6] "Type"       "imdbVotes"  "seriesID"   "Season"     "Director"
## [11] "Released"   "Awards"     "Genre"      "imdbRating" "Poster"
## [16] "Episode"    "Language"   "Country"    "Runtime"    "imdbID"
## [21] "Metascore"  "Response"   "Year"       "Error"
```

Not all of these attributes were for use in any data mining task, however they were retained for archiveal 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 more for archival purposes than any data mining task.

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 full preprocessing script is shown below. Critical steps are described in the code comments. Note that the script ends with two significant steps:

1. Subset data to retain only movies (30K rows ==> ~12K rows)
2. Save dataset to R data object and to csv (for coherent use across all project members)

```
# 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)
```

```r
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 (setting max number)
# and re-parsed 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

# 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)
```

```r
# 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 - fine as is

# 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]

# 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, meaningless)

  # 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
}
```

At the end of the preprocessing stage, the full list of dataset variables was:

```
data <- preprocessing(data)
names(data)
```

```
##  [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"  "Composer"
## [31] "CostumeDesigners"
```

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

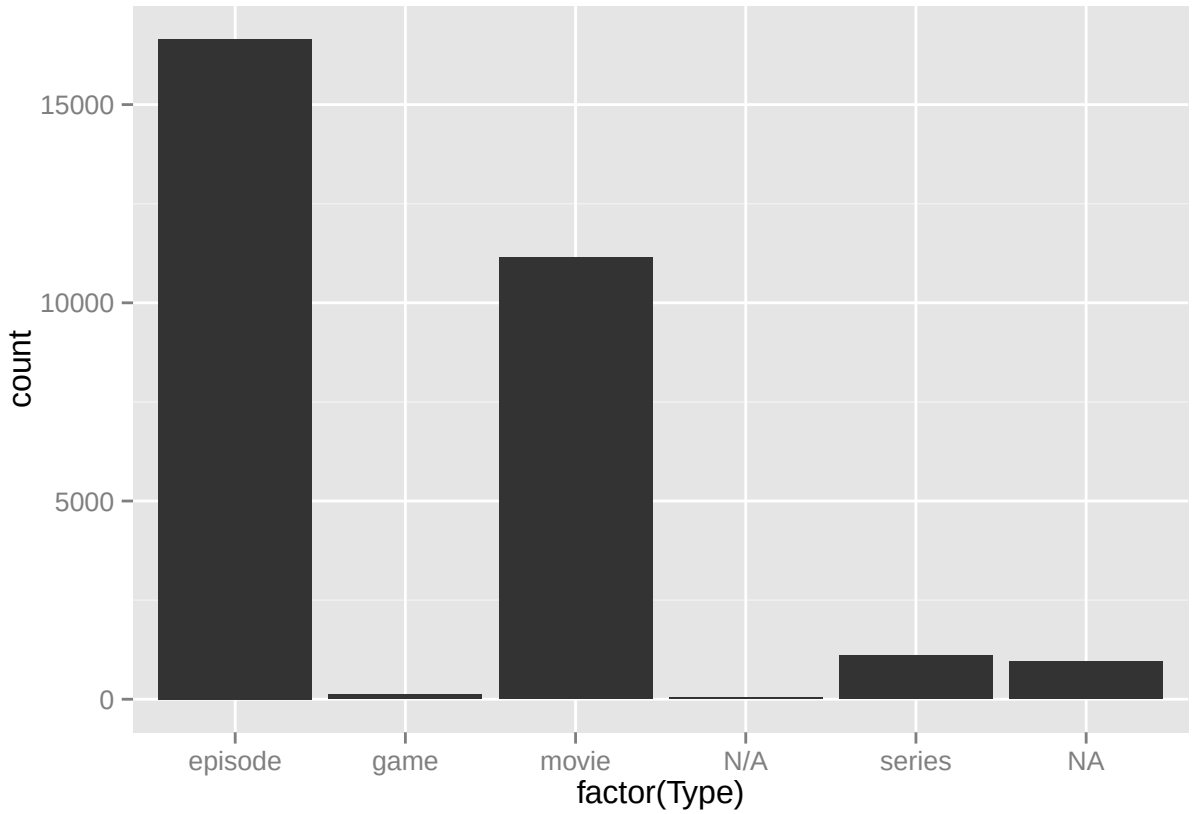
An initial query was made to examine how many of the original 30K elements were valid movies as well as how many showed up as null values.

```
# 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.

```
# Run preprocessing procedures
stats <- preprocessing(stats)

# 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))
```

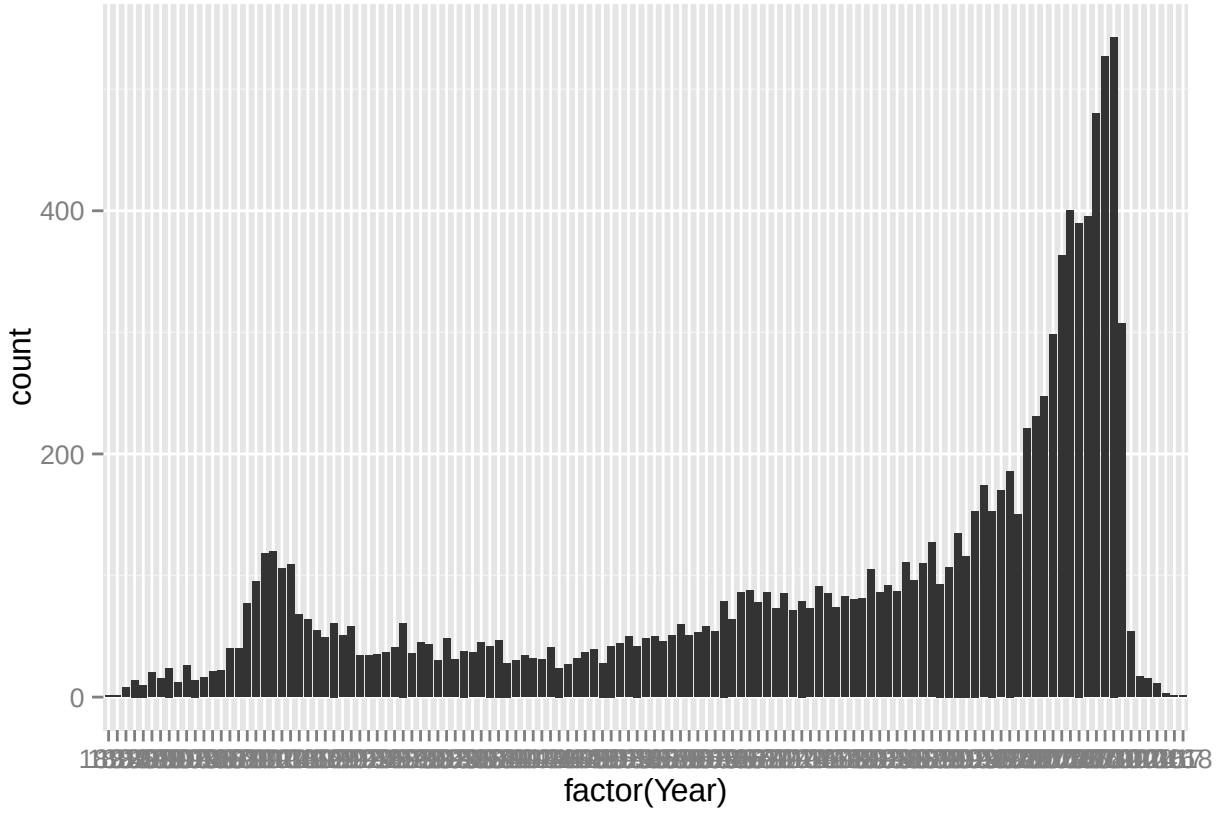
```
## episode    game   movie     N/A  series
##       0       0   11142       0       0
```

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.

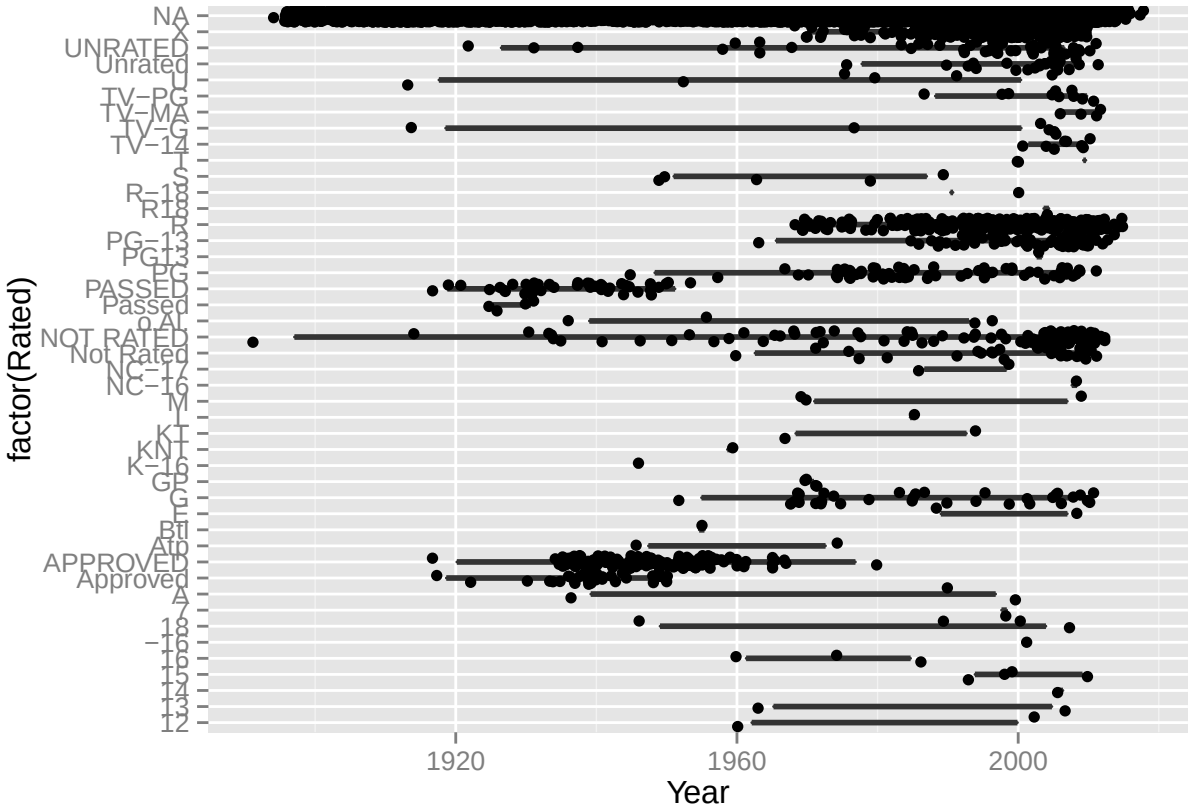
```
# Display distribution of movie years
ggplot(stats, aes(x = factor(Year))) + geom_bar(stat = "bin")
```



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.

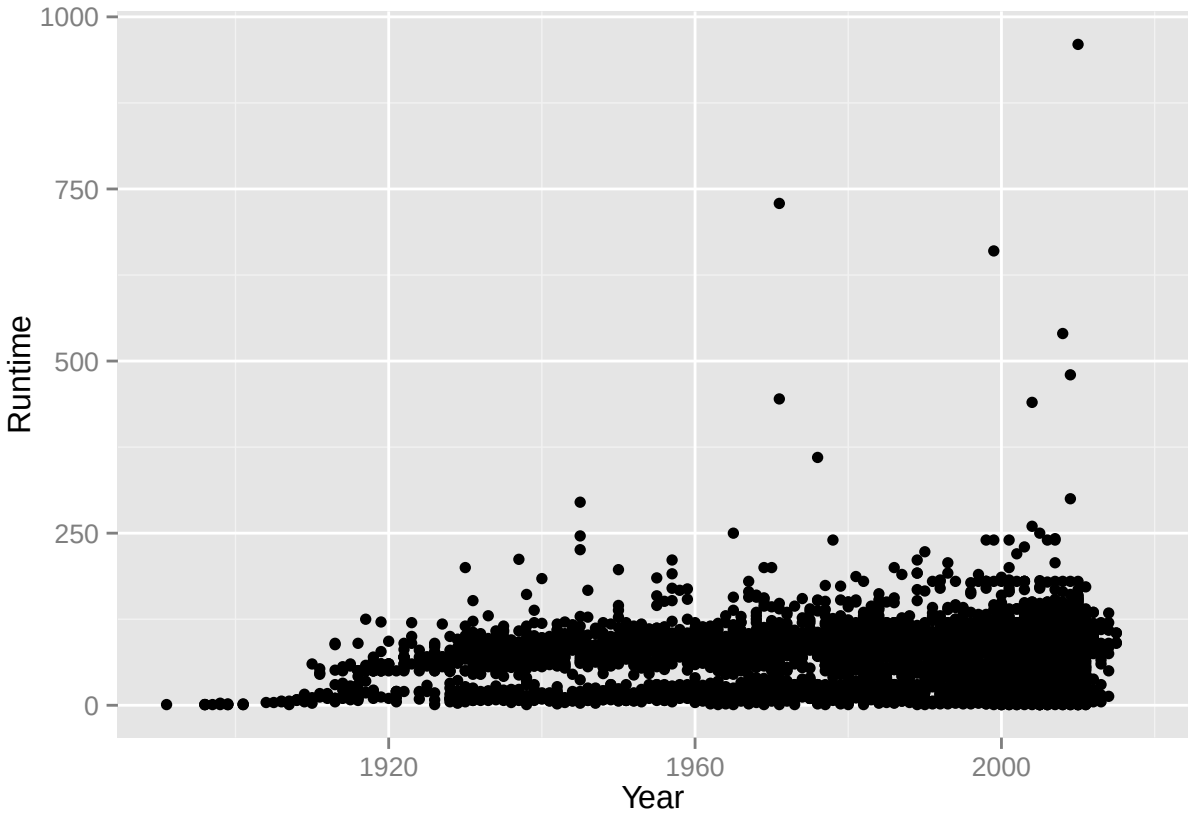
```
# Display ratings by year
qplot(Year,factor(Rated),data=stats,geom=c("boxplot","jitter"))
```



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.

One type of preprocessing that we addressed was for the Producer and Cinematographer columns the data was formatted as such ['LastName, FirstName']. In order to use this data in a format that could be used by the classification methods we performed the following operations. Note: Same process was run for Cinematographer

```
#################################### 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)
```

One other type of processing we had to address was for the column 'Writer' because it read in with parenthetical notes that affected how the classification algorithms performed. The processing is shown below:

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

The following code below are snippets of the key functions of the GenreClassification.R script:

Here we remove the columns that don't aid classification:

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

Here we use unnest because some of our attributes are stored as lists and for processing purposes they needed to be their own rows.

```
# Create unique rows for each genre
movieData <- unnest(movieData,Genre)
movieData <- unnest(movieData, Writer)
#movieData <- unnest(movieData, Actors)
movieData <- unnest(movieData, Country)
```

Here we remove the rows with the NA genre because these cannot be classified:

```
movieData <- movieData[!(is.na(as.factor(movieData$Genre))),]
movieData <- movieData[!movieData$Genre == "N/A",]
movieData$Genre <- as.factor(movieData$Genre)
```

Here we cast the string as factors so that the classification algorithms have the correct parameters:

```
# 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)
```

Like in project I we assign training and test sets:

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

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(obliqueModelPredictionsMovi


#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)
```

Below prints the accuracies for each method used above in a table:

```
accRipMovie <- ripCMMovie$overall[1]
accC45Movie <- c45CMMovie$overall[1]
accNBMovie <- nbCMMovie$overall[1]
accKNNMovie <- knnCMMovie$overall[1]

accuracyMatrix <- matrix(c(accRipMovie, accC45Movie, accNBMovie, accKNNMovie), ncol = 1, byrow = TRUE)
colnames(accuracyMatrix) <- c("Movie Data Accuracies")
rownames(accuracyMatrix) <- c("Ripper", "C4.5", "Naive Bayes", "KNN")
accuracyMatrix <- as.table(accuracyMatrix)
print(accuracyMatrix)
```

‘‘‘

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).

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

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

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.

– Insert picture here –

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

```
# 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'.

– Insert other distance matrix photo –

Results

Genre Classification

The results of the classifications are shown below.

Clustering

The result of the cluster assignments are shown below.

– Insert analysis of kCluster output (original)– – withinss, centers, size, etc –

– Insert analysis of 2nd kCluster output ("good" one)– – withinss, centers, size, etc –

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 imporve the results as much as possible include trial and errror 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.

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.

```
# count unique genres
length(unique(dataPruned))
```

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

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 http://www.imdb.com
- IMDbPY - Python API and Materials for IMDB searches http://www.imdbpy.sourceforge.net
- The Open Movie Database http://www.omdbapi.com/

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