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 done 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 into the following major categories:

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

Literature Review

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 listed 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") }
  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 provided 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
## Author: Dax Gerts
## Description: use OMDb API to rapidly build a clean JSON IMDb data sample

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
## 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

Each variable was examined to make sure that it had a

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 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.

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. 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 (for coherent use across all project members)

```
# Clean data set, set proper types, drop invalid rows,
preprocessing <- function(data) {
  # Step 1: Variable type checking
```

```r
# 1.1 Plot - char (fine as is)
# 1.2 Rated - factor (49 levels) (sparse)
data$Rated <- as.factor(data$Rated)
summary(data$Rated)

# 1.3 Title - char (fine as is)

# 1.4 Writer - factor (>10000 levels)
data$Writer <- as.factor(data$Writer)
summary(data$Writer)

# 1.5 Actors - list (will require more processing later)
data$Actors <- strsplit(data$Actors,", ")
data$Actors <- sapply(data$Actors,'[',seq(max(sapply(data$Actors,length))),simplify=FALSE)
data$Actors[1:5]

# 1.6 Type - factor
data$Type <- as.factor(data$Type)
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)
levels(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 (parse for numeric values later, maybe)

# 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 (unfinished)
  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)
  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 Prepare valid data

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

  # 2.2 Drop bad/"N/A" entries
  data <- data[!(is.na(as.factor(data$Title))),]

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

  # 2.4 Return output table
  data
}
```

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.

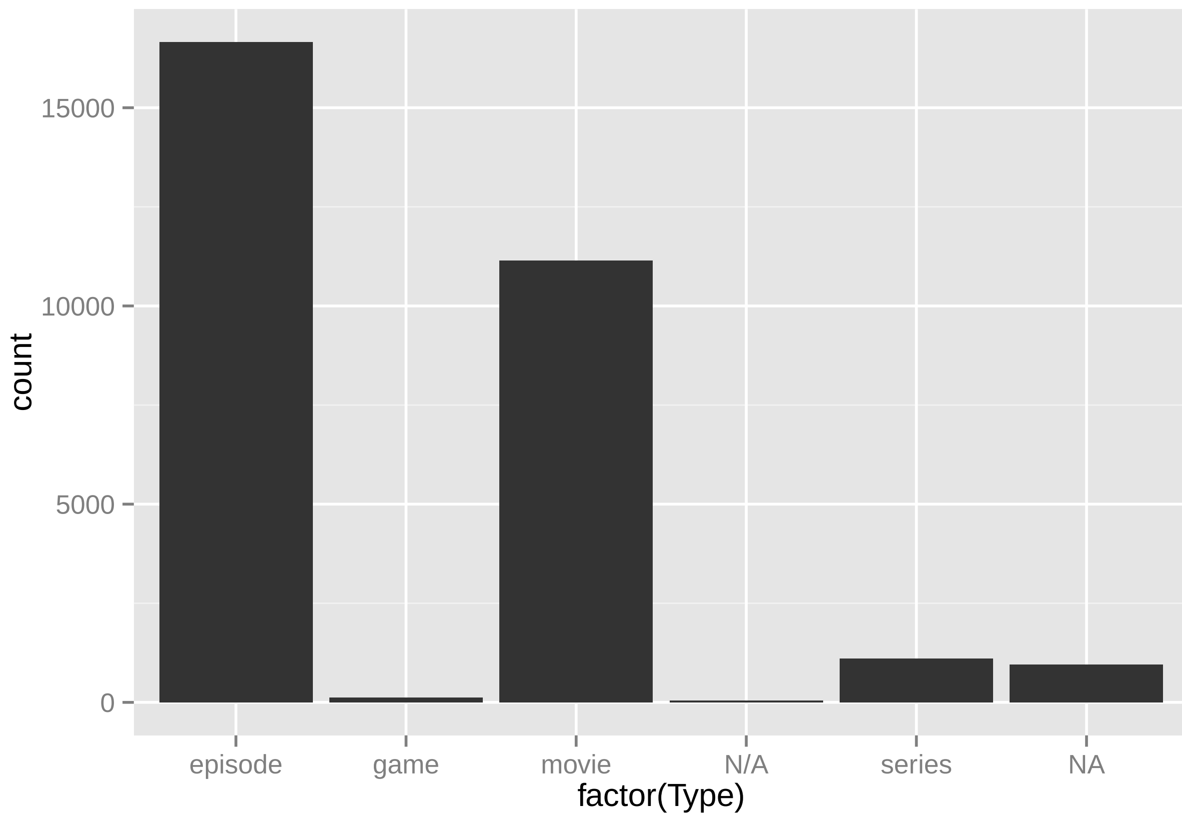
Before Preprocessing

```
# 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")
```



After Preprocessing

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

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

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

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

```
## episode    game   movie     N/A  series
##       0       0   11138       0       0
```

Method and Materials

Results

Discussion

Conclusion

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/