Python Practical – Analysing a Movie Recommendation Engine with Tensorflow

Peter Moore

Student ID: 170000983

Course: AC52050

16th July 2018

## Understanding and Objectives

### Objectives[[1]](#footnote-1)

#### Background

This project takes a movie Recommendation Engine written in Google Brain’s[[2]](#endnote-1) TensorFlow[[3]](#endnote-2) and uses it to examine which users like what. That is, if the Tensorflow code recommends a movie, is that recommendation any good?

#### Objective

The primary goal of the project is three-fold:

1. To place users into categories
2. To take users from a given category and see what movies they like
3. To see how these compares to the recommendation engine

A secondary goal is to demonstrate a working knowledge of the Python [[4]](#endnote-3) programming language.

#### Success criteria

In terms of the primary goal: a successful project assesses the quality of the Tensorflow recommendation engine and delivers those results with statistical rigour.

The secondary goal is achieved by presenting code samples[[5]](#footnote-2) and a zip file. These are referenced throughout and reside in the appendix.

### Assessment of situation

#### Inventory of resources - Data

Code for a sample recommendation engine[[6]](#endnote-4) was provided by Andrew Cobley[[7]](#endnote-5); itself an adaptation of original code[[8]](#endnote-6) by Guocong Song[[9]](#endnote-7). The code takes movie rating data[[10]](#endnote-8) made open by MovieLens[[11]](#endnote-9), a website for movie recommendations.

The data consists of three files: “These files contain 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users who joined MovieLens in 2000.”[[12]](#footnote-3)

The data is relational.

#### Requirements

The original data needs to be fully-realised; initially this will be done using Python but ultimately the most appropriate technology will be used. For example, Python for deep learning; SQL[[13]](#endnote-10) for relational data and R[[14]](#endnote-11) for statistical analyses.

#### Assumptions

The project will take place under the following assumptions:

* The data provided is real.
* The data provided is complete.
* The deep learning output is taken as is. That is, whilst the results are questioned, techniques such as LIME as described by *Ribeiro et al* [[15]](#endnote-12) are left alone[[16]](#footnote-4).
* In the event that data is perceived to be "missing" an appropriate substitution technique may be used, and, in these cases, this will be stated.

#### Constraints

#### Constraints were budget and available technology. The budget was nil, and the available technology did not contain Nvidia graphics card. This limited the project to running on CPU.

#### This budgetary constraint led to an important data set constraint inasmuch as the project stays within the MovieLens 1m data set and does not reach for the (also available) 20M dataset[[17]](#endnote-13)

#### Terminology

* MovieLens: the MovieLens 1M dataset defined as a database
* Data File: one of:
  + Ratings: the data provided in the ratings.dat data file
  + Movies: movie provided in the movies.dat data file
  + Users: watches of Movie as per the users.dat file
* RBM: Restricted Boltzmann Machine
* Recommendation Engine: the specific recommendation engine built by the RBM
* Tensorflow: used here to describe the Tensorflow framework and as a shorthand to mean *derived from deep learning algorithm using Tensorflow*  (for example, in the phrase “the Tensorflow results”)
* Tensor: “a generalization of vectors and matrices to potentially higher dimensions”[[18]](#endnote-14)
* Graph: set of linear algebra transformations performed on Tensors in a pre-determined order (aka dataflow graph[[19]](#endnote-15))
* **Demographic:** a grouping of users based on their movie choice (e.g. the Tensorflow Demographic)
* **Tensorflow Demographic:** the definition of a User based on movies selected for them by Tensorflow
* **User Demographic:** the definition of a User based on non-Tensorflow categorisation

#### Risks

The project runs the risk of timeout and CPU burnout although these are considered small. Less emphatic is the risk posed by long running queries. This is considered real but manageable.

#### Contingencies

All data stored is backed up to iCloud[[20]](#endnote-16).

#### Data mining goals

The goal of the data mining is threefold:

* To investigate Tensorflow algorithms that recommend movies to users in terms of:
  + Quality of algorithm
  + Ease of adapting the code
  + Ability to store recommendations on a per user basis
* To place Users in a demographic and to do this in a manner agnostic to the Tensorflow algorithm; for example, by clustering.
* To form a full data comparison between the two sets.

#### Data mining success criteria

The accuracy of the tensorflow model is known via leverage of the Users information.

## Data understanding:

### Data collection and description

All of the MovieLens data was reviewed see ([Appendix One: Raw data](#Appendix One: Raw data))

The data was downloaded and loaded into Python via the code in [Appendix 2.1: Read data file](#Appendix 2.1: Read data file). This output the following info: the Data File as data frame, the number of records and a split into training and test data.

Each Data frame was observed to contain data relevant to its title. For example, the Movies table contained a recognisable film title. Furthermore, the Data Files seemed to reference one another via ID fields. For example, Ratings contained MovieID and UserID.

There were *exactly* 1,000,209 ratings records; 3,883 Movie records and 6,040 User records.

Finally, the Data files contained ancillary data that may have been of use. In particular, pertaining to:

* Age
* Occupation
* Year of release
* Genre
* And, indirectly, ZipCode

These are described in [Appendix Three: Ancillary data](#Appendix Three: Ancillary data)

### Data exploration, quality and preparation

Intra-file: The data contained no duplicates and was only occasionally messy where text encoding errors were observed. Of greater import was the mismatch between Maximum ID and length of file. The former being greater than the latter. This meant that there were missing ID values. In particular, with respect to Movies, this would have caused a problem if not spotted.

Inter-file: it was observed that the ID fields were consistent between databases. This meant the data was a relational structure making SQL Server a natural candidate whence it came to storage of the data.

## Data preparation:

### Selection of data

All of the data was considered of use and none jettisoned

### Cleaning of data

Apart from where explicitly stated below the data was left as is.

### Construction and storage of data in an appropriate model

Having been loaded to data frame, Ratings, Movies and Users were loaded into SQL server where they took the names: tblRating, tblMovie and tblUser with the below Python code[[21]](#footnote-5):

﻿datainout.loadtosqlstage(df =ratingstrainingdata, stagetablename="tblRating", ifexists="fail")

datainout.loadtosqlstage(df =moviestrainingdata, stagetablename="tblMovie", ifexists="fail")

datainout.loadtosqlstage(df =userstrainingdata, stagetablename="tblUser", ifexists="fail")

Age and Occupation took the form of IDs and as described in the MovieLens “read me” file[[22]](#endnote-17). These were turned into Python dictionaries per [Appendix 3.0: Age data as dictionary](#Appendix 3.0: Age data as dictionary) and [Appendix 3.1: Occupation data as dictionary](#Appendix 3.1: Occupation data as dictionary). In turn, they were turned into the SQL tables tblAge and tblOccupation via: [Appendix 3.2: Python code to transport dictionary to SQL](#Appendix 3.2: Python code to transport dictionary to SQL)

The year could be inferred from the movie title, for example: Grumpier Old Men (1995). This was easily extracted via the code in [Appendix 3.3: Python and SQL code to update release year](#Appendix 3.3: Python and SQL code to update release year).

Zip code formed a simpler problem, although some convention was applied here. Namely that there was no use for the four-part string at the end of a Zip Code[[23]](#footnote-6). At this point it was possible to convert the zip code to an integer and to create the two-digit zip in case it was useful. The Python code for this is at [Appendix 2.2: Clean user data](#Appendix 2.2: Clean user data).

Genre appeared as a pipe-separated list, for example: Adventure|Children's|Fantasy. This was dealt with via a mixture of Python and SQL that can be seen here: [Appendix 3.4: Python and SQL code to create Movie vs Genre look up](#Appendix 3.4: Python and SQL code to create Movie vs Genre look up) Two tables were created: tblMoviexGenre and tblGenre.

### Format of data i.e. the database schema

The database schema can be seen in [Appendix Four: database schema](#Appendix Four: database schema)

### Data interrogation

A few simple queries led to the below *superficial* observations. All of which are visualised in [Appendix Five: Visualisations](#Appendix Five: Visualisations)[[24]](#footnote-7)

#### Observations

* There are significantly more males than females
* Occupation varies a lot with age
* Choice of genre varies with gender, age and occupation

## Modelling (Tensorflow Demographic)

### Selection of modelling techniques

The algorithm supplied showed detectable errors in the original songgc code . In particular the variables aligned in ITEM\_NUM and USER\_NUM were incorrect. Although the code was passable[[25]](#footnote-8) in that it recommended movies; it was not as intuitive[[26]](#footnote-9) as the RBM code provided by Cognitive Class[[27]](#footnote-10). This code was co-opted to build the model and hereinafter, *Tensorflow model* refers to the RBM one.

### Test design

The original Tensorflow code was entirely inline. This was refactored so that the training part of the model be modular. The module created can be found in [Appendix Six: RBM model](#Appendix Six: RBM model). It is called from the main code by the following lines:

﻿import LearnModel

prv\_w, prv\_vb, prv\_hb, W, vb, hb, v0 = LearnModel.buildviewinghabits(sess,trX,hiddenUnits, visibleUnits)

What is happening? A Tensorflow session (sess) is being passed to a learning routine; alongside this are three variables that are explained now.

### Model build

The entirety of the model build code can be found in [Appendix Seven: Build model](#Appendix Seven: Build model). The important elements are:

trX: represents the collection of movie ratings grouped per User. In effect it is a list of each user rating contained within a list of each movie. The net effect is to provide a tensor of per user recommendations to the model. This is the training set.

visibleUnits: this represents the size of the visible layer of neuron

hiddenUnits: this represents the size of the hidden layer.

The choice of the size of these two variables I discussed now.

### Assessment of model

#### Method for model assessment

For input the model takes a visible layer of size 3,883 (the number of movies) and passes this to a hidden layer of length 40. It is this layer that learns the model’s features. Iterating to reduce the mean absolute error function per the line:

err\_sum = tf.reduce\_mean(err \* err)

The original code ran across 15 epochs, however, observation of the loss curve led to a belief that more gains could be attained by increasing this. The model seemed to have an asymptote at an error of about 0.33. (See figure 1). Therefore, the code was thus amended to 999 epochs but with a break condition of

﻿ if lasterr<0.033:

print("breaking at " + str(i))

break



Figure 1: loss curve showing error asymptotic to 0.33

Finally, a number of users needed to be assigned for testing. This number was set to 1000. This was because of the speed constraint (ideally it would have been higher).

#### Results for model assessment

Repeated iterations of the code with these parameters led to selections of movies that were self-similar.

#### Conclusion for model assessment

This in turn meant that the RBM results were stable enough to store.

#### Method for recommendations gathering

By merging the movies and ratings set, the below data frame was built:

﻿merged\_df = moviestrainingdata.merge(ratingstrainingdata, on='MovieID')

From this the user output by the trained model could be ascertained viz. ﻿

UserID = merged\_df.iloc[pu]["UserID"]

Looping through the training set then, it was possible to build a table of recommendations. (NB This vital piece of code appears in [Appendix Seven: Build model](#Appendix Seven: Build model) in red text on grey background)

#### Results for recommendations gathering

The resulting table, tblRecommendation, sits in SQL Server and is the repository for comparing recommendations. The data model can be seen in [Appendix 7.2: Recommendations data model](#Appendix 7.2: Recommendations data model) (figure A7.1).

#### Conclusion for recommendations gathering

With TensorFlow’s recommendations now stored, the next step was to create the User Demographic.

## Modelling (User Demographic)

### Selection of modelling techniques

Whilst it had been observed that the user demographics were superficially different; this was difficult to prove. For example, the below visualisation appears to show that different genders like different genres

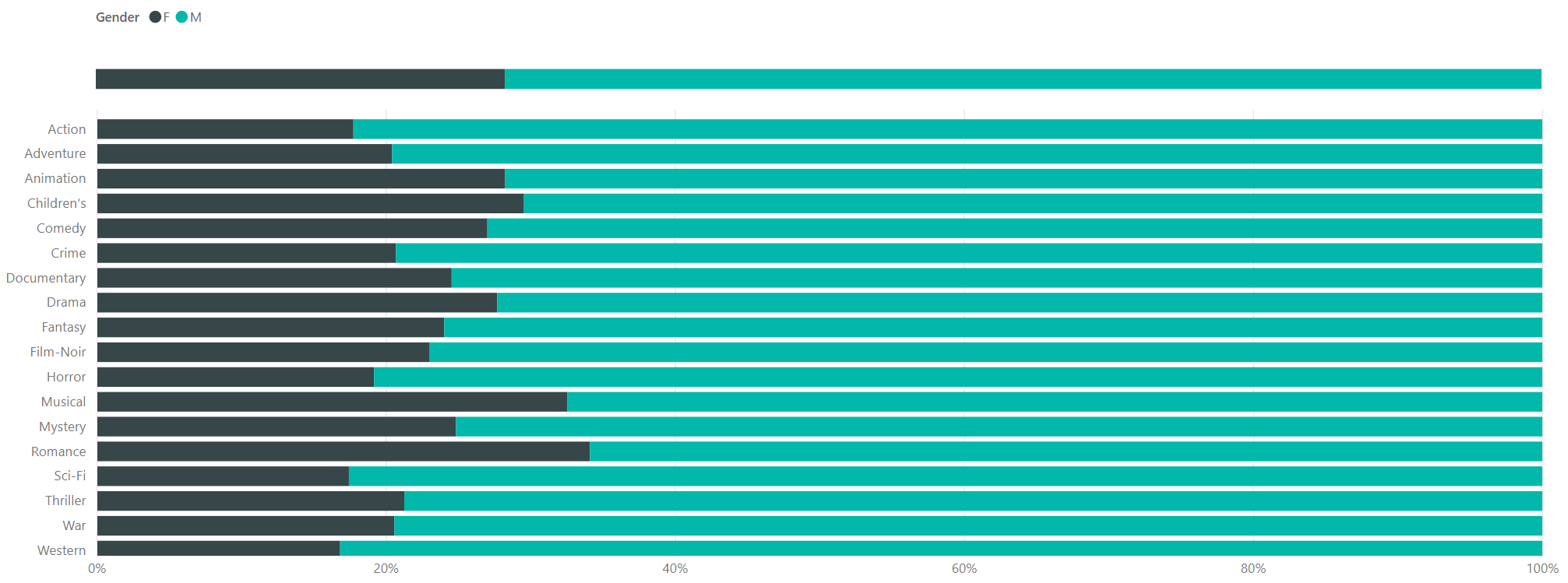


Figure 2: Genre vs Gender[[28]](#footnote-11)

However, setting a null hypothesis that there is no difference between them cannot be disproved at even 80% confidence (see [Appendix Eight: Chi-square tests for differences in User Demographic](#Appendix Eight: Chi-square tests for differences in User Demographic)). The same occurred for age and occupation.

Therefore, a more advanced series of models were needed for grouping the data.

### Model builds

#### Method for K-means

K-means is a method of separating continuous variables into a stated number of clusters. This was applied to the age, gender and occupation fields; across a varying number of clusters. (See [Appendix Nine: K-means code](#Appendix Nine: K-means code))

Initial signs were promising as the “elbow” curve dropped off rapidly; suggesting it was possible to get away with a small number of clusters. (See Figure 3):



Figure 3: “Elbow” curve for k-means clusters

#### Results K-means

Figure 4 shows the cluster centres across Z=Gender, X=Occupation and Y=Age.



Figure 4: K-means “Clusters”

The figure clearly shows that the data did not cluster.

#### Conclusion K-means

That the data did not cluster was not entirely surprising as K-means is designed to work with continuous data set[[29]](#footnote-12).

#### Method, Results and Conclusion for Principle Component Analysis

The idea of modifying the above K-means analysis with a Principle Component analysi was briefly and unsuccessfully explored; ultimately – and unsurprisingly – this befell the same fate as K-means, For completeness, it is included in [Appendix Nine Addendum: PCA](#Appendix Nine Addendum: PCA)

#### Method for Association Rules

The reason the above two models not work is that the data is not continuous, furthermore, the learning is unsupervised: there is simply no “master list” for each user’s favourite movies. Therefore, a method of unsupervised learning was needed. This was the association rules method.

Invoking Python’s mlxtend[[30]](#endnote-18) repository, association rules were compiled[[31]](#footnote-13). The code for this can be seen in [Appendix Ten: Association rules](#Appendix Ten: Association rules). The original code for this attempts to say “people who like this also like this” where the subject is the movie. This is subverted here as if to say “movies who like her also like him”; in this way users are placed into a demographic[[32]](#footnote-14). Results were formed by taking a minimum threshold of one pair; support of 20%; lift of over 2 and confidence of above 0.5[[33]](#footnote-15).

#### Results for Association Rules

The association rules results were analysed. The method came up with some “obvious” results (e.g. the Empire Strikes Back vs A New Hope). For further analysis, the results placed into the SQL structure shown in [Appendix 10.2: Association rules data model](#Appendix 10.2: Association rules data model) (figure A10.1))

#### Conclusion for Association Rules

An independent User Demographic has been created with which to compare the Tensorflow Demographic[[34]](#footnote-16).

## Overall assessment of model

#### Story so far

The Users have been split into two demographics: one from TensorFlow, another from association rules. These can be used to compare the two models and assess the quality of the Tensorflow predictions. In what follows a recommendation means a TensorFlow recommendation; antecedents and consequents come from the User demographic

#### Method

In the first instance the association rules are taken as the “truth” and the Tensorflow results compared against them. In order to make the data computable, a subset of 100 users with recommendations was assessed. Each user was assigned a status denoting the following:

|  |  |  |
| --- | --- | --- |
| Status | Movie has antecedents[[35]](#footnote-17) | Movie has no antecedents |
| User recommended movie | True positive | False positive |
| User not recommended movie | True negative | False negative |

#### Results

|  |  |  |
| --- | --- | --- |
| Status | Movie has antecedents[[36]](#footnote-18) | Movie has no antecedents |
| User recommended movie | 354 | 465 |
| User not recommended movie | 385292 | 1008 |

That is the Tensorflow model provided a true positive rate of 26% versus a false positive rate of 0.12%.

#### Conclusion

The Tensorflow model predicted movies with numerous false positives compared to other methods. However, when viewed in the aggregate there is some quality in the prediction.

## Evaluation/Reflection

This project found many ways *not* to assess the Tensorflow model before settling on something of a milquetoast conclusion. That is, the project could have been vastly improved by running the Tensorflow model at various thresholds. In the final analysis, however, there was only things left to vary came from the association rules. This was the wrong way around: The association rules grouped movies with users as an adjunct and, whilst this was the best way to group these users, it served the project less well than, say , running the RBM multiple times and collating those results. Ultimately that cost the project the crowning glory of a ROC curve[[37]](#footnote-19).

From a personal point standpoint, endeavour was high, but direction was poor. I spent a lot of time with docker, telnet and the songgc code only to end up using none of them. If I had my time again I would have started with the user groupings and placed the TensorFlow second. That is: it recommends things, go and test the recommendations. That said, I know a lot more TensorFlow now.

Finally, as an exercise in coding Python, this was useful. As I review thirty pages of appendix, I’m surprised. As a Python practical, I feel great progress has been made. Whilst this is also true in terms of assessment of the algorithm, I don’t feel this document reflects that. Sometimes the hardest lessons to learn, last the longest.

## Appendix Zero: Technical Set-up

Technical specifications of relevance:

### Hardware

All code performed on the following machine:

**Hardware Overview:**

Model Name: Mac mini

Model Identifier: Macmini7,1

Processor Name: Intel Core i7

Processor Speed: 3 GHz

Number of Processors: 1

Total Number of Cores: 2

L2 Cache (per Core): 256 KB

L3 Cache: 4 MB

Memory: 16 GB

### Software

All Python code was executed from a bespoke anaconda 4.5.4 environment with the following libraries. Unless stated explicitly, the .py files were executed from within Spyder 2.8 .

The SQL Server version was:

Microsoft SQL Server 2017 (RTM) - 14.0.1000.169 (X64)

Aug 22 2017 17:04:49

Copyright (C) 2017 Microsoft Corporation

Developer Edition (64-bit) on Windows 10 Pro 10.0 <X64> (Build 17134: ) (Hypervisor)

The SQL Server ran on Windows 10 Pro, running on Parallels Desktop 13 for Mac Pro Version 13.3.2 (43368)

The R code was ran on R-studio 1.1.423 and R version R version 3.4.3 (2017-11-30)

The following packages were utilised:

|  |  |
| --- | --- |
| Name | Version |
| \_r-mutex | 1.0.0 |
| alabaster | 0.7.11 |
| appnope | 0.1.0 |
| asn1crypto | 0.24.0 |
| astroid | 1.6.5 |
| babel | 2.6.0 |
| backcall | 0.1.0 |
| blas | 1.0 |
| bleach | 2.1.3 |
| bwidget | 1.9.11 |
| bzip2 | 1.0.6 |
| ca-certificates | 2018.03. |
| cairo | 1.14.12 |
| cctools | 895 |
| certifi | 2018.4.1 |
| cffi | 1.11.5 |
| chardet | 3.0.4 |
| clang | 4.0.1 |
| clang\_osx-64 | 4.0.1 |
| clangxx | 4.0.1 |
| clangxx\_osx-64 | 4.0.1 |
| cloudpickle | 0.5.3 |
| compiler-rt | 4.0.1 |
| cryptography | 2.2.2 |
| curl | 7.60.0 |
| cycler | 0.10.0 |
| cyrus-sasl | 2.1.26 |
| dbus | 1.13.2 |
| decorator | 4.3.0 |
| docutils | 0.14 |
| entrypoints | 0.2.3 |
| expat | 2.2.5 |
| font-ttf-dejavu-sans-mono | 2.37 |
| font-ttf-inconsolata | 2.001 |
| font-ttf-source-code-pro | 2.030 |
| font-ttf-ubuntu | 0.83 |
| fontconfig | 2.12.6 |
| fonts-anaconda | 1 |
| freetype | 2.8 |
| gettext | 0.19.8.1 |
| gfortran\_osx-64 | 4.8.5 |
| glib | 2.56.1 |
| graphite2 | 1.3.11 |
| gsl | 2.4 |
| harfbuzz | 1.7.6 |
| html5lib | 1.0.1 |
| icu | 58.2 |
| idna | 2.7 |
| imagesize | 1.0.0 |
| intel-openmp | 2018.0.3 |
| ipykernel | 4.8.2 |
| ipython | 6.4.0 |
| ipython\_genutils | 0.2.0 |
| isort | 4.3.4 |
| jedi | 0.12.0 |
| jinja2 | 2.10 |
| jpeg | 9b |
| jsonschema | 2.6.0 |
| jupyter\_client | 5.2.3 |
| jupyter\_core | 4.4.0 |
| kiwisolver | 1.0.1 |
| krb5 | 1.14.2 |
| lazy-object-proxy | 1.3.1 |
| ld64 | 274.2 |
| libcurl | 7.60.0 |
| libcxx | 4.0.1 |
| libcxxabi | 4.0.1 |
| libedit | 3.1 |
| libffi | 3.2.1 |
| libgfortran | 3.0.1 |
| libiconv | 1.15 |
| libntlm | 1.4 |
| libpng | 1.6.34 |
| libprotobuf | 3.5.2 |
| libsodium | 1.0.16 |
| libssh2 | 1.8.0 |
| libtiff | 4.0.9 |
| libxml2 | 2.9.8 |
| llvm | 4.0.1 |
| llvm-lto-tapi | 4.0.1 |
| llvm-openmp | 4.0.1 |
| markupsafe | 1.0 |
| matplotlib | 2.2.2 |
| mccabe | 0.6.1 |
| mistune | 0.8.3 |
| mkl | 2018.0.3 |
| mkl\_fft | 1.0.1 |
| mkl\_random | 1.0.1 |
| mlxtend | 0.12.0 |
| nbconvert | 5.3.1 |
| nbformat | 4.4.0 |
| ncurses | 6.0 |
| notebook | 5.5.0 |
| numpy | 1.12.1 |
| numpy-base | 1.14.5 |
| numpydoc | 0.8.0 |
| openssl | 1.0.2o |
| packaging | 17.1 |
| pandas | 0.23.1 |
| pandoc | 2.2.1 |
| pandocfilters | 1.4.2 |
| pango | 1.41.0 |
| parso | 0.2.1 |
| patsy | 0.5.0 |
| pcre | 8.42 |
| pexpect | 4.6.0 |
| pickleshare | 0.7.4 |
| pip | 10.0.1 |
| pixman | 0.34.0 |
| prompt\_toolkit | 1.0.15 |
| protobuf | 3.5.2 |
| psutil | 5.4.6 |
| ptyprocess | 0.6.0 |
| pycodestyle | 2.4.0 |
| pycparser | 2.18 |
| pyflakes | 2.0.0 |
| pygments | 2.2.0 |
| pylint | 1.9.2 |
| pyodbc | 4.0.23 |
| pyopenssl | 18.0.0 |
| pyparsing | 2.2.0 |
| pyqt | 5.9.2 |
| pysocks | 1.6.8 |
| python | 3.6.5 |
| python-dateutil | 2.7.3 |
| python.app | 2 |
| pytz | 2018.4 |
| pyzmq | 17.0.0 |
| qt | 5.9.6 |
| qtawesome | 0.4.4 |
| qtconsole | 4.3.1 |
| qtpy | 1.4.2 |
| r-assertthat | 0.2.0 |
| r-backports | 1.1.2 |
| r-base | 3.4.3 |
| r-base64enc | 0.1\_3 |
| r-bh | 1.65.0\_1 |
| r-bindr | 0.1 |
| r-bindrcpp | 0.2 |
| r-bit | 1.1\_12 |
| r-bit64 | 0.9\_7 |
| r-bitops | 1.0\_6 |
| r-blob | 1.1.0 |
| r-boot | 1.3\_20 |
| r-broom | 0.4.3 |
| r-callr | 1.0.0 |
| r-caret | 6.0\_78 |
| r-catools | 1.17.1 |
| r-cellranger | 1.1.0 |
| r-class | 7.3\_14 |
| r-cli | 1.0.0 |
| r-clipr | 0.4.0 |
| r-cluster | 2.0.6 |
| r-codetools | 0.2\_15 |
| r-colorspace | 1.3\_2 |
| r-config | 0.2 |
| r-crayon | 1.3.4 |
| r-curl | 3.1 |
| r-cvst | 0.2\_1 |
| r-data.table | 1.10.4\_3 |
| r-dbi | 0.7 |
| r-dbplyr | 1.1.0 |
| r-ddalpha | 1.3.1 |
| r-deoptimr | 1.0\_8 |
| r-dichromat | 2.0\_0 |
| r-digest | 0.6.13 |
| r-dimred | 0.1.0 |
| r-dplyr | 0.7.4 |
| r-drr | 0.0.2 |
| r-essentials | 3.4.3 |
| r-evaluate | 0.10.1 |
| r-forcats | 0.2.0 |
| r-foreach | 1.4.4 |
| r-foreign | 0.8\_69 |
| r-formatr | 1.5 |
| r-ggplot2 | 2.2.1 |
| r-glmnet | 2.0\_13 |
| r-glue | 1.2.0 |
| r-gower | 0.1.2 |
| r-gtable | 0.2.0 |
| r-haven | 1.1.0 |
| r-hexbin | 1.27.1 |
| r-highr | 0.6 |
| r-hms | 0.4.0 |
| r-htmltools | 0.3.6 |
| r-htmlwidgets | 0.9 |
| r-httpuv | 1.3.5 |
| r-httr | 1.3.1 |
| r-ipred | 0.9\_6 |
| r-irdisplay | 0.4.4 |
| r-irkernel | 0.8.11 |
| r-iterators | 1.0.9 |
| r-jsonlite | 1.5 |
| r-kernlab | 0.9\_25 |
| r-kernsmooth | 2.23\_15 |
| r-knitr | 1.18 |
| r-labeling | 0.3 |
| r-lattice | 0.20\_35 |
| r-lava | 1.5.1 |
| r-lazyeval | 0.2.1 |
| r-lubridate | 1.7.1 |
| r-magrittr | 1.5 |
| r-maps | 3.2.0 |
| r-markdown | 0.8 |
| r-mass | 7.3\_48 |
| r-matrix | 1.2\_12 |
| r-mgcv | 1.8\_22 |
| r-mime | 0.5 |
| r-miniui | 0.1.1 |
| r-mnormt | 1.5\_5 |
| r-modelmetrics | 1.1.0 |
| r-modelr | 0.1.1 |
| r-mongolite | 1.4 |
| r-munsell | 0.4.3 |
| r-nlme | 3.1\_131 |
| r-nnet | 7.3\_12 |
| r-numderiv | 2016.8\_1 |
| r-odbc | 1.1.2 |
| r-openssl | 0.9.9 |
| r-packrat | 0.4.8\_1 |
| r-pbdzmq | 0.3\_0 |
| r-pillar | 1.0.1 |
| r-pkgconfig | 2.0.1 |
| r-pki | 0.1\_5.1 |
| r-plogr | 0.1\_1 |
| r-plyr | 1.8.4 |
| r-prodlim | 1.6.1 |
| r-profvis | 0.3.4 |
| r-psych | 1.7.8 |
| r-purrr | 0.2.4 |
| r-quantmod | 0.4\_12 |
| r-r6 | 2.2.2 |
| r-randomforest | 4.6\_12 |
| r-rappdirs | 0.3.1 |
| r-rbokeh | 0.6.3 |
| r-rcolorbrewer | 1.1\_2 |
| r-rcpp | 0.12.14 |
| r-rcpproll | 0.2.2 |
| r-rcurl | 1.95\_4.9 |
| r-readr | 1.1.1 |
| r-readxl | 1.0.0 |
| r-recipes | 0.1.1 |
| r-recommended | 3.4.3 |
| r-rematch | 1.0.1 |
| r-repr | 0.12.0 |
| r-reprex | 0.1.1 |
| r-reshape2 | 1.4.3 |
| r-rjava | 0.9\_9 |
| r-rjdbc | 0.2\_5 |
| r-rjsonio | 1.3\_0 |
| r-rlang | 0.1.6 |
| r-rmarkdown | 1.8 |
| r-robustbase | 0.92\_8 |
| r-rodbc | 1.3\_15 |
| r-rpart | 4.1\_11 |
| r-rprojroot | 1.3\_1 |
| r-rsconnect | 0.8.5 |
| r-rstudioapi | 0.7 |
| r-rvest | 0.3.2 |
| r-scales | 0.5.0 |
| r-selectr | 0.3\_1 |
| r-sfsmisc | 1.1\_1 |
| r-shiny | 1.0.5 |
| r-sourcetools | 0.1.6 |
| r-sparklyr | 0.8.2 |
| r-spatial | 7.3\_11 |
| r-stringi | 1.1.6 |
| r-stringr | 1.2.0 |
| r-survival | 2.41\_3 |
| r-tibble | 1.4.1 |
| r-tidyr | 0.7.2 |
| r-tidyselect | 0.2.3 |
| r-tidyverse | 1.2.1 |
| r-timedate | 3042.101 |
| r-ttr | 0.23\_2 |
| r-utf8 | 1.1.2 |
| r-uuid | 0.1\_2 |
| r-viridislite | 0.2.0 |
| r-whisker | 0.3\_2 |
| r-withr | 2.1.1 |
| r-xml2 | 1.1.1 |
| r-xtable | 1.8\_2 |
| r-xts | 0.10\_1 |
| r-yaml | 2.1.16 |
| r-zoo | 1.8\_0 |
| readline | 7.0 |
| requests | 2.19.1 |
| rope | 0.10.7 |
| rstudio | 1.1.423 |
| scikit-learn | 0.19.1 |
| scipy | 1.1.0 |
| seaborn | 0.8.1 |
| send2trash | 1.5.0 |
| setuptools | 39.2.0 |
| simplegeneric | 0.8.1 |
| sip | 4.19.8 |
| six | 1.11.0 |
| snowballstemmer | 1.2.1 |
| sphinx | 1.7.5 |
| sphinxcontrib | 1.0 |
| sphinxcontrib-websupport | 1.1.0 |
| spyder | 3.2.8 |
| sqlalchemy | 1.2.8 |
| sqlite | 3.23.1 |
| statsmodels | 0.9.0 |
| tensorflow | 1.1.0 |
| terminado | 0.8.1 |
| testpath | 0.3.1 |
| tk | 8.6.7 |
| tktable | 2.10 |
| tornado | 5.0.2 |
| traitlets | 4.3.2 |
| unixodbc | 2.3.6 |
| urllib3 | 1.23 |
| wcwidth | 0.1.7 |
| webencodings | 0.5.1 |
| werkzeug | 0.14.1 |
| wheel | 0.31.1 |
| wrapt | 1.10.11 |
| xlrd | 1.1.0 |
| xz | 5.2.4 |
| zeromq | 4.2.5 |
| zlib | 1.2.11 |

## Appendix One: Raw data

Sample MovieLens data:

### Appendix 1.1: Ratings Sample

#### Python code:

ratingstrainigndata.head(5)

#### Data

|  |  |  |  |
| --- | --- | --- | --- |
| UserID | MovieID | Rating | Timestamp |
| 1 | 1 | 5 | 978824268 |
| 1 | 48 | 5 | 978824351 |
| 1 | 150 | 5 | 978301777 |
| 1 | 260 | 4 | 978300760 |
| 1 | 527 | 5 | 978824195 |

Figure A1.1: Sample Ratings Data

### Appendix 1.2: Movies Sample

#### Python code:

moviestrainigndata.head(5)

#### Data

|  |  |  |  |
| --- | --- | --- | --- |
| MovieID | Title | Genres | ReleaseYear |
| 1 | Toy Story | Animation|Children's|Comedy | 1995 |
| 2 | Jumanji | Adventure|Children's|Fantasy | 1995 |
| 3 | Grumpier Old Men | Comedy|Romance | 1995 |
| 4 | Waiting to Exhale | Comedy|Drama | 1995 |
| 5 | Father of the Bride Part II | Comedy | 1995 |

Figure A1.2: Sample Movies Data

### Appendix 1.3: User Sample

#### Python code:

userestrainigndata.head(5)

#### Data

(grey fields added later)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| UserID | Gender | AgeID | OccupationID | ZipCode | ZipShort | ZipcodePrimary |
| 1 | F | 1 | 10 | 48067 | 48 | 48067 |
| 2 | M | 56 | 16 | 70072 | 70 | 70072 |
| 3 | M | 25 | 15 | 55117 | 55 | 55117 |
| 4 | M | 45 | 7 | 02460 | 2 | 2460 |
| 5 | M | 25 | 20 | 55455 | 55 | 55455 |

Figure A1.3: Sample Users Data

## Appendix Two: Data movement code

This code was used to move data between locations and to manipulate data once it had got to said location (for example to make a Zip Code numeric). It is largely original code. It can be found in the accompanying zip file at datainout.py

### Appendix 2.0: Imports

import numpy as np

import pandas as pd

#code to take a data frame (df) and place in a staging area of a named connection

import sqlalchemy

import pyodbc

#read .dat or .csv file with a specified schema

### Appendix 2.1: Manipulate data file

#### Appendix 2.1.0: read data file

def readdatfile(filename, sep, dataschema):

colHeaders=list(dataschema.keys())

datafromfile = pd.read\_csv(filepath\_or\_buffer=filename,sep=sep,header=None,names=colHeaders, dtype=dataschema,engine='python')

rowcount=len(datafromfile)

#add a local id to account for missing IDs

# datafromfile["LocalInd"] = datafromfile.index

return datafromfile, rowcount

#get a dat file, read it (using above function) and split it into trainingproportion

#return training set, test set and the redefined schema

#NB redfined schema incudes row counts

#### Appendix 2.1.1: split data file

def splitdatdata(trainingproportion, datFILE, datSCHEMA):

dfdat, dfdatrows = readdatfile(filename=datFILE, sep="::", dataschema=datSCHEMA)

# newind=np.random.permutation(dfdatrows)#random permutation of indices to reseed

# dfdat = dfdat.iloc[newind].reset\_index(drop=True)

splitind = int(dfdatrows\*trainingproportion)

dfdattrain = dfdat[0:splitind]

dfdattest = dfdat[splitind:]#.reset\_index(drop=True)

dfSchema = datSCHEMA

def endwithcount(x): return x.endswith("ID") #local funtion to isolate "count" fields

for counted in filter(endwithcount,list(dfSchema.keys())):

countresult = len(dfdat.groupby(counted)) #loop to get count fields

dfSchema[counted+"Count"] = countresult

return dfdattrain,dfdattest,dfdatrows, dfSchema

### Appendix 2.2: Clean user data

#code for specific user file functions

def treatuserfile(userstrainingdata):

#add short two digit zip

userstrainingdata["ZipShort"] = userstrainingdata.ZipCode.str.slice(0, 2).astype(np.int32)

userstrainingdata["ZipcodePrimary"] = userstrainingdata.ZipCode.str.slice(0, 5).astype(np.int32)

return userstrainingdata

### Appendix 2.3: Talk to SQL

#### Appendix 2.3.0: Does a table exist

#function to check if a table exists

def tableexists(stagetablename ,connectionstring='DSN=MYMSSQL\_MOVIES;UID=sa;PWD=AnyOldPassword'):

extant = False

params=stagetablename

conn = pyodbc.connect(connectionstring, autocommit=True)

crsr = conn.cursor()

rows = crsr.execute("select name from sys.tables where name = ?", params).fetchall()

crsr.close()

conn.close()

if len(rows)> 0:

extant = True

return extant

#### Appendix 2.3.1: Does a field exist

#function to check if a field exists

def fieldexists(stagetablename, stagefieldname,connectionstring='DSN=MYMSSQL\_MOVIES;UID=sa;PWD=AnyOldPassword'):

extant = False

params=[stagetablename, stagefieldname]

conn = pyodbc.connect(connectionstring, autocommit=True)

crsr = conn.cursor()

rows = crsr.execute("SELECT t.name, c.name FROM sys.tables t INNER JOIN sys.columns AS c ON c.object\_id = t.object\_id WHERE t.name = ? AND c.name = ?;", params).fetchall()

crsr.close()

conn.close()

if len(rows)> 0:

extant = True

return extant

#### Appendix 2.3.2: Manipulate SQL table

def loadtosqlstage(df,

stagetablename,

connectionstring="mssql+pyodbc://sa:AnyOldPassword@MYMSSQL\_Movies",

ifexists='replace',

index=False,

indexlabel=None,

dtype=None):

engine = sqlalchemy.create\_engine(connectionstring)

BuildDF = True

if ifexists == "fail":#if set to failure when table exists the don't build

if tableexists(stagetablename):

BuildDF = False

# write the DataFrame to a table in the sql database

if BuildDF:

df.to\_sql(name=stagetablename, con = engine, if\_exists=ifexists, index=index, index\_label=indexlabel,dtype=dtype)

#### Appendix 2.3.3: Run SQL from text file

#function to run SQL from a text file

#indirectly inspired by: https://stackoverflow.com/questions/38856534/execute-sql-file-with-multiple-statements-separated-by-using-pyodbc

def runsqltxt(script,connectionstring='DSN=MYMSSQL\_MOVIES;UID=sa;PWD=AnyOldPassword'):

conn = pyodbc.connect(connectionstring, autocommit=True)

with open(script,'r') as actions:

sqlScript = actions.read()

print(sqlScript)

for statement in sqlScript.split(';'):

with conn.cursor() as cur:

cur.execute(statement)

conn.close()

#### Appendix 2.3.0: Get SQL data and turn into pandas dataframe

#function to make a dataframe from a sql statement

#indirectly inspired by: https://stackoverflow.com/questions/39835770/move-data-from-pyodbc-to-pandas

def sqldf(sql,connectionstring='DSN=MYMSSQL\_MOVIES;UID=sa;PWD=AnyOldPassword'):

conn = pyodbc.connect(connectionstring, autocommit=True)

df = pd.read\_sql(sql,conn)

return df

## Appendix Three: Ancillary data

### Appendix 3.0: Age data as dictionary

﻿dictAge={

1:"Under 18",

18:"18-24",

25:"25-34",

35:"35-44",

45:"45-49",

50:"50-55",

56:"56+"}

### Appendix 3.1: Occupation data as dictionary

dictOccupation={

0:"other or not specified",

1:"academic/educator",

2:"artist",

3:"clerical/admin",

4:"college/grad student",

5:"customer service",

6:"doctor/health care",

7:"executive/managerial",

8:"farmer",

9:"homemaker",

10:"K-12 student",

11:"lawyer",

12:"programmer",

13:"retired",

14:"sales/marketing",

15:"scientist",

16:"self-employed",

17:"technician/engineer",

18:"tradesman/craftsman",

19:"unemployed",

20:"writer"}

### Appendix 3.2: Python code to transport dictionary to SQL

This code utilised [Appendix 2.3.2: Manipulate SQL table](#Appendix 2.3.2: Manipulate SQL table)

﻿#load Age data

dfAge = pd.DataFrame.from\_dict(data=dictAge, orient='index',columns=["Age"])

datainout.loadtosqlstage(df=dfAge, stagetablename="tblAge", ifexists="fail",index=True,indexlabel="AgeID")

#load occupation data

dfOccupation = pd.DataFrame.from\_dict(data=dictOccupation, orient='index',columns=["Occupation"])

datainout.loadtosqlstage(df=dfOccupation, stagetablename="tblOccupation", ifexists="fail",index=True,indexlabel="OccupationID")

### Appendix 3.3: Python and SQL code to update release year

This code was executed via the code in [Appendix 2.3.3: Run SQL from text file](#Appendix 2.3.3: Run SQL from text file)

﻿Python

#get release year for movie intra-SQL based on what we already have

if not datainout.fieldexists("tblMovie", "ReleaseYear"):

PATH\_YEARSQL = os.path.join(PATH\_ME, "Split Movie Year.txt")

datainout.runsqltxt(script=PATH\_YEARSQL)

SQL

ALTER TABLE tblMovie ADD ReleaseYear SMALLINT;

UPDATE tm

SET tm.Title = REPLACE(tm.Title, u.ShortYear, '')

, tm.ReleaseYear = REPLACE(REPLACE(u.ShortYear, '(', ''), ')', '')

FROM dbo.tblMovie tm

INNER JOIN

(

SELECT MovieID

, REVERSE(LEFT(REVERSE(RTRIM(Title)), 6)) AS ShortYear

FROM dbo.tblMovie

) u

ON tm.MovieID = u.MovieID;

### Appendix 3.4: Python and SQL code to create Movie vs Genre look up

This code was executed via the code in [Appendix 2.3.3: Run SQL from text file](#Appendix 2.3.3: Run SQL from text file)

Python

﻿#build a genre table intra-SQL based on what we already have

if not datainout.tableexists("tblGenre"):

PATH\_GENRESQL = os.path.join(PATH\_ME, "Build Movie x Genre.txt")

datainout.runsqltxt(script=PATH\_GENRESQL)

SQL

DROP TABLE IF EXISTS dbo.tblGenre;

SELECT DISTINCT

IDENTITY(INT, 1, 1) AS GenreID

, ulsd.value AS Genre

INTO dbo.tblGenre

FROM tblMovie m

CROSS APPLY STRING\_SPLIT(m.Genres, '|') AS ulsd;

DROP TABLE IF EXISTS dbo.tblMoviexGenre;

SELECT IDENTITY(INT, 1, 1) AS MoviexGenreID

, m.MovieID

, g2.GenreID

INTO dbo.tblMoviexGenre

FROM dbo.tblMovie m

CROSS APPLY STRING\_SPLIT(m.Genres, '|') AS ulsd

INNER JOIN dbo.tblGenre AS g2

ON ulsd.value = g2.Genre

ORDER BY m.MovieID

, g2.GenreID;

## Appendix Four: database schema

Once the data was extracted using Python, it was loaded into SQL Server where the tables were arranged thus:

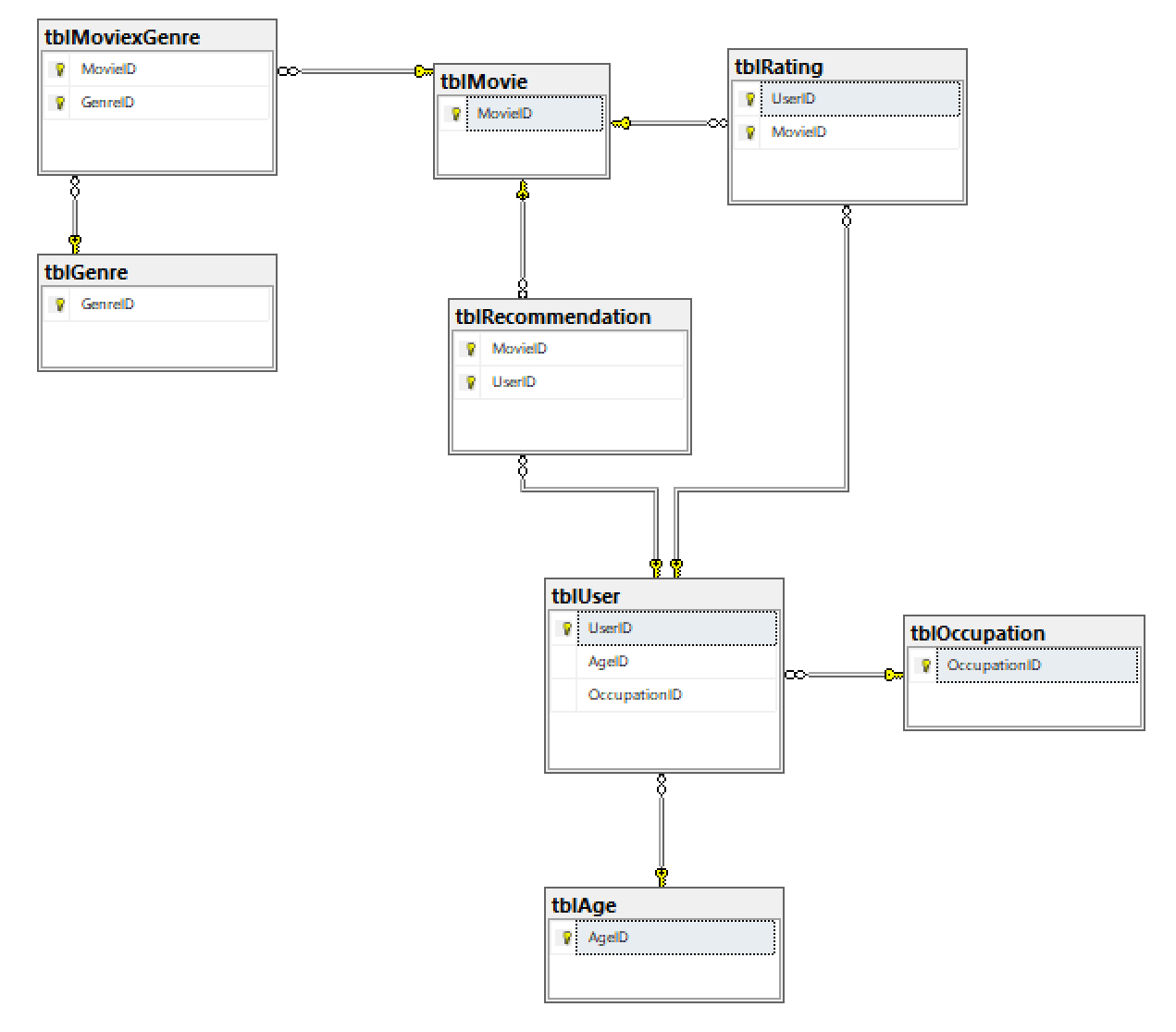


Figure A4.0: Database Schema

## Appendix Five: Visualisations

### Appendix 5.1: Variation by age and gender

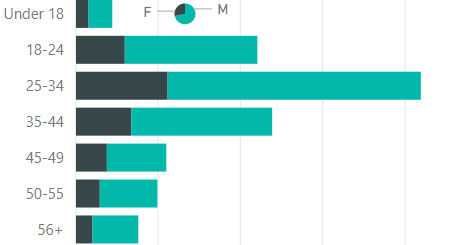


Figure A5.1: Variation by age and gender

### Appendix 5.2: Variation by age and occupation

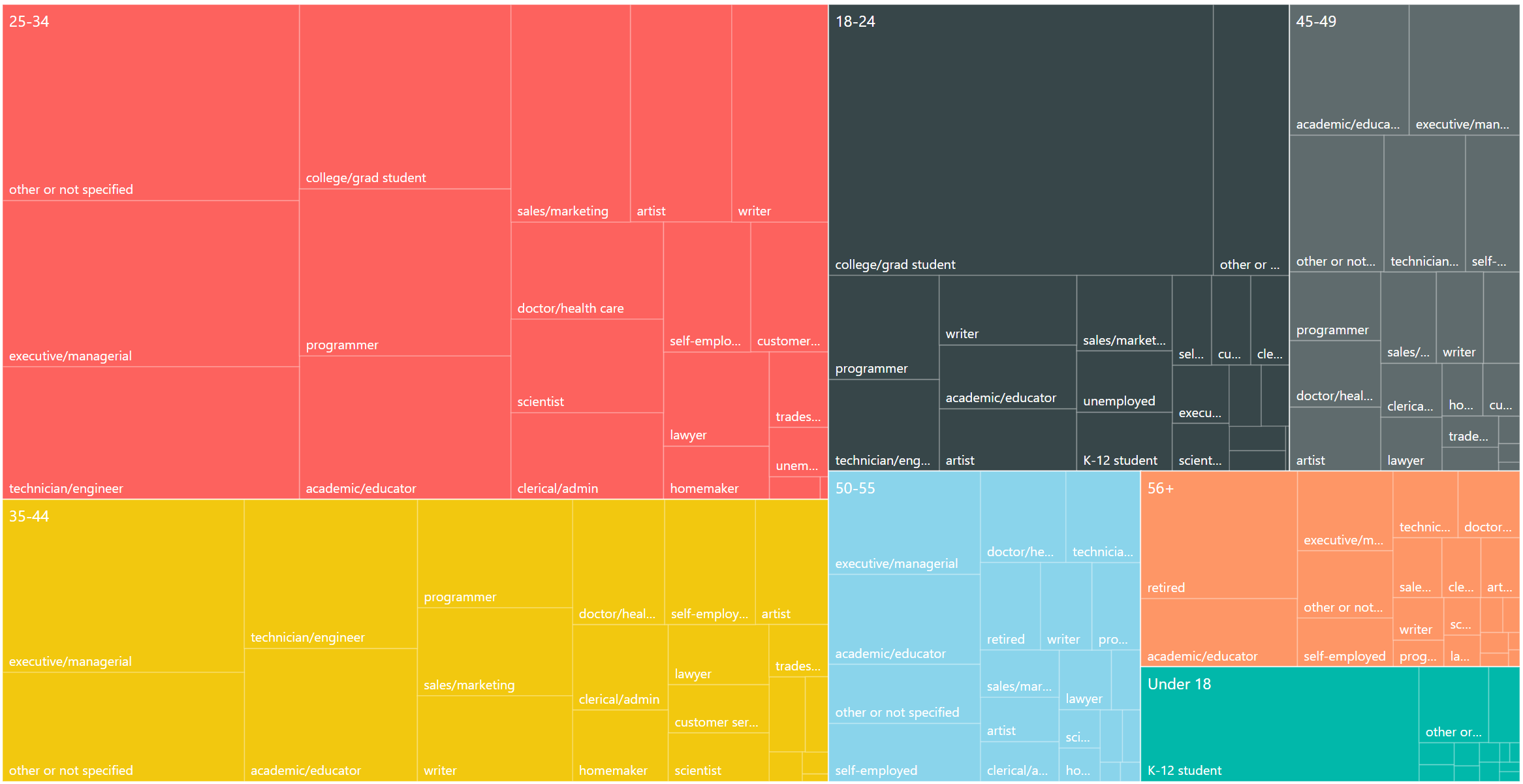


Figure A5.2: Variation by age and occupation

### Appendix 5.3: Variation by gender and genre

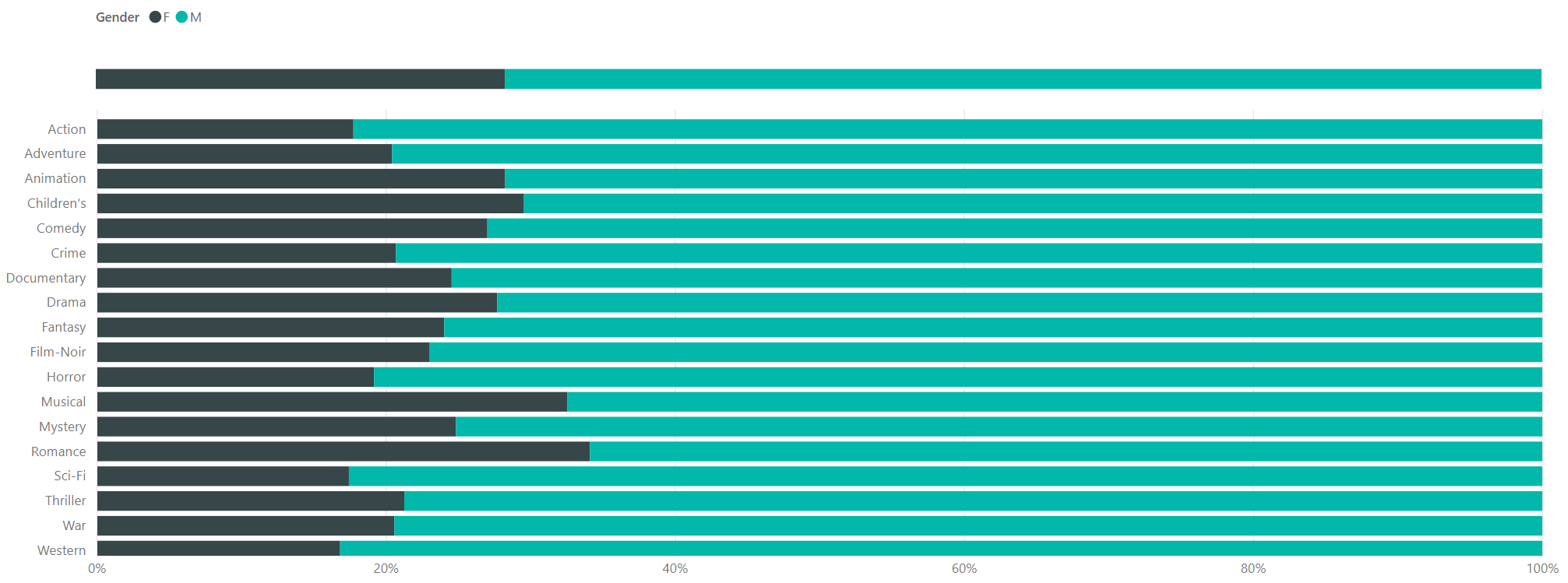


Figure A5.3: Variation by genre and gender

### Appendix 5.4: Variation by age and genre

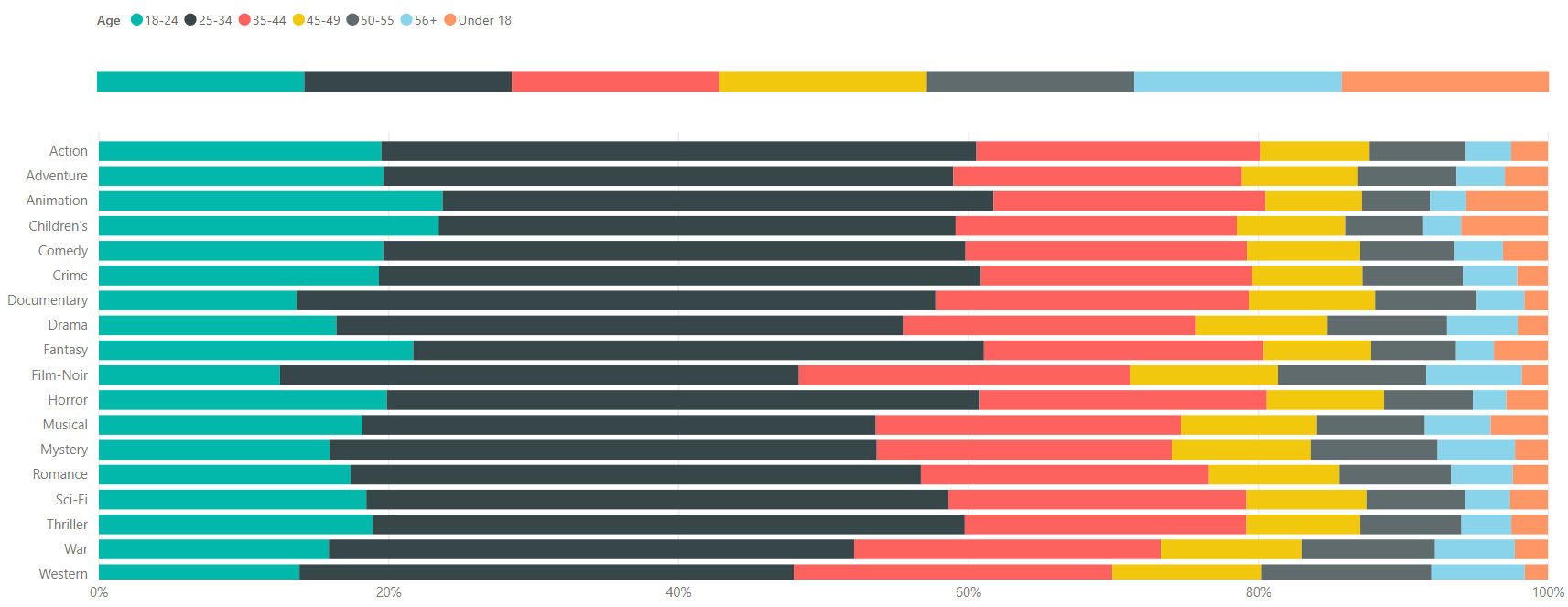


Figure A5.4: Variation by age and genre

### Appendix 5.5: Variation by occupation and genre

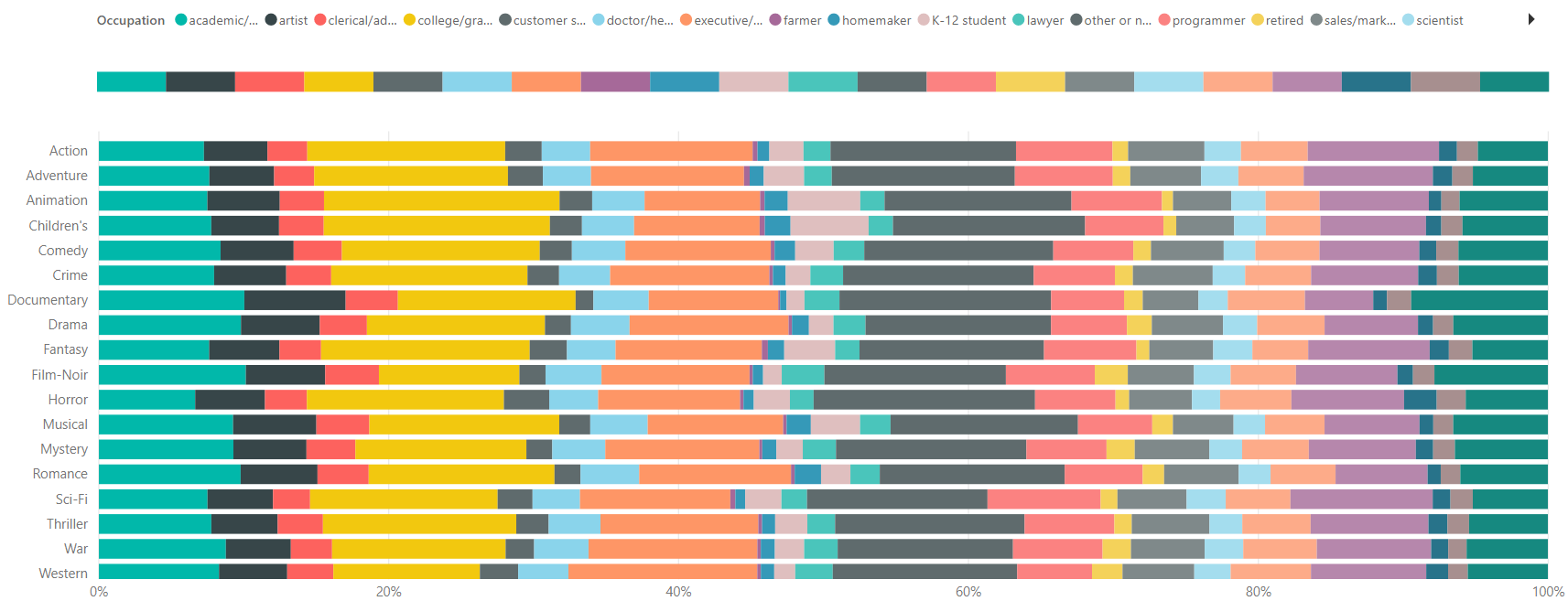


Figure A5.5: Variation by occupation and genre

## Appendix Six: RBM model

The below code form the engine room of the Tensorflow model.

It appears in the zip fie as LearnModel.py

"""

Parts of the code herein were

Inspired by original code at cognitiveclass.ai

it can be found here (but you may have go sign up):

https://courses.cognitiveclass.ai/courses/course-v1:CognitiveClass+ML0120ENv2+2018/courseware/89227024130b43f684d95376901b65c8/e7c36d2c4c6840fe8b81b97147ea9c16/

I have marked \_their\_ code with #CCAI.

I have marked \_their comments\_ with #CCOM

"""

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

"""

CCOM

Next, let's start building our RBM with Tensorflow.

We'll begin by first determining the amount of hidden layers

and then creating placeholder variables for storing our visible layer biases,

hidden layer biases and weights that connect the hidden layer with the visible one.

We will be arbitrarily setting the amount of hidden layers to 20.

You can freely set this value to any number you want since each neuron in the hidden

layer will end up learning a feature.

"""

def buildviewinghabits(sess,trX,hiddenUnits, visibleUnits):

vb = tf.placeholder("float", [visibleUnits]) #CCAI: Number of unique movies

hb = tf.placeholder("float", [hiddenUnits]) #CCAI: Number of features we're going to learn

W = tf.placeholder("float", [visibleUnits, hiddenUnits])

"""

CCOM

We then move on to creating the visible and hidden layer units

and setting their activation functions. In this case, we will

be using the tf.sigmoid and tf.relu functions as nonlinear activations

since it's what is usually used in RBM's.

"""

#CCAI: Phase 1: Input Processing

v0 = tf.placeholder("float", [None, visibleUnits])

\_h0= tf.nn.sigmoid(tf.matmul(v0, W) + hb)

h0 = tf.nn.relu(tf.sign(\_h0 - tf.random\_uniform(tf.shape(\_h0))))

"""

CCOM

initialize our variables

"""

#CCAI: Current weight

cur\_w = np.zeros([visibleUnits, hiddenUnits], np.float32)

#CCAI: Current visible unit biases

cur\_vb = np.zeros([visibleUnits], np.float32)

#CCAI: Current hidden unit biases

cur\_hb = np.zeros([hiddenUnits], np.float32)

#CCAI: Phase 2: Reconstruction

\_v1 = tf.nn.sigmoid(tf.matmul(h0, tf.transpose(W)) + vb)

v1 = tf.nn.relu(tf.sign(\_v1 - tf.random\_uniform(tf.shape(\_v1))))

h1 = tf.nn.sigmoid(tf.matmul(v1, W) + hb)

"""

CCOM

Now we set the RBM training parameters and functions.

"""

#CCAI: Learning rate

alpha = 1.0

#CCAI: Create the gradients

w\_pos\_grad = tf.matmul(tf.transpose(v0), h0)

w\_neg\_grad = tf.matmul(tf.transpose(v1), h1)

#CCAI: Calculate the Contrastive Divergence to maximize

CD = (w\_pos\_grad - w\_neg\_grad) / tf.to\_float(tf.shape(v0)[0])

#CCAI: Create methods to update the weights and biases

update\_w = W + alpha \* CD

update\_vb = vb + alpha \* tf.reduce\_mean(v0 - v1, 0)

update\_hb = hb + alpha \* tf.reduce\_mean(h0 - h1, 0)

"""

CCOM

And set the error function, which in this case will be the Mean Absolute Error Function.

"""

err = v0 - v1

err\_sum = tf.reduce\_mean(err \* err)#PM just reducing mena square error

"""

CCOM (PM note: numbers subject to change)

Now we train the RBM with

15 epochs with each epoch using

10 batches with size 100.

After training, we print out a graph with the error by epoch.

"""

epochs = 999#PM changed from 15 to try and remove inconsistency in results

batchsize = 100

errors = []

for i in range(epochs):

for start, end in zip( range(0, len(trX), batchsize), range(batchsize, len(trX), batchsize)):

batch = trX[start:end]

# prv\_w = cur\_w #order of operation reversed because

# prv\_vb = cur\_vb

# prv\_hb = cur\_hb

cur\_w = sess.run(update\_w, feed\_dict={v0: batch, W: cur\_w, vb: cur\_vb, hb: cur\_hb})

cur\_vb = sess.run(update\_vb, feed\_dict={v0: batch, W: cur\_w, vb: cur\_vb, hb: cur\_hb})

cur\_hb = sess.run(update\_hb, feed\_dict={v0: batch, W: cur\_w, vb: cur\_vb, hb: cur\_hb})

errors.append(sess.run(err\_sum, feed\_dict={v0: trX, W: cur\_w, vb: cur\_vb, hb: cur\_hb}))

lasterr=errors[-1]

if lasterr<0.033:

print("breaking at " + str(i))

break

print (lasterr)

plt.plot(errors)

plt.ylabel('Error')

plt.xlabel('Epoch')

plt.show()

return cur\_w, cur\_vb, cur\_hb, W, vb, hb, v0

## Appendix Seven: Build model

The below code which is original except where stated.

Seen holistically, it collects all of the data from the Data File, builds the SQL database and then trains the Tensorflow model. (note the calling Learn Model halfway through)

### Appendix 7.1: Training Python code

"""

Parts of the code herein were

inspried by original code at cognitiveclass.ai

it can be found here (but you may have go sign up):

https://courses.cognitiveclass.ai/courses/course-v1:CognitiveClass+ML0120ENv2+2018/courseware/89227024130b43f684d95376901b65c8/e7c36d2c4c6840fe8b81b97147ea9c16/

I have marked \_their\_ code with #CCAI.

I have marked \_their comments\_ with #CCOM

"""

#get tensorflow

import tensorflow as tf

#get others

import numpy as np

import pandas as pd

import os

#import sqlalchemy

#change working directory

PATH\_FILE = "/Users/petermoore/Documents/GitHub/Movies/Trainspotting Three"#os.path.dirname(os.path.realpath(\_\_file\_\_))

os.chdir(PATH\_FILE)

import datainout

PATH\_ME = os.getcwd()

PATH\_DD = os.path.join(PATH\_ME, "downloaded data")

PATH\_LOG = os.path.join(PATH\_ME,"Log")

PATH\_RATINGS = os.path.join(PATH\_DD,"ratings.dat")

PATH\_MOVIES = os.path.join(PATH\_DD,"movies.dat")

PATH\_USERS = os.path.join(PATH\_DD,"users.dat")

DEVICE = "/cpu:0"

#PM dictionaries for schemas

#numpy schema for python

RATINGSSCHEMA={"UserID":np.int32, "MovieID":np.int32, "Rating":np.int32, "Timestamp":np.int32}#NB names changed to match readme schema: UserID::MovieID::Rating::Timestamp

MOVIESSCHEMA={"MovieID":np.int32, "Title":np.object, "Genres":np.object}#NB names changed to match readme schema: MovieID::Title::Genres

USERSSCHEMA={"UserID":np.int32, "Gender":np.object, "AgeID":np.float32, "OccupationID":np.int32, "ZipCode":np.object}#NB names changed to match readme schema: UserID::Gender::Age::Occupation::Zip-code

##sql schema for DB

#RATINGSSCHEMA\_SQL={"UserID":sqlalchemy.types.INTEGER(), "MovieID":sqlalchemy.types.INTEGER(), "Rating":sqlalchemy.types.INTEGER(), "Timestamp":sqlalchemy.types.INTEGER()}#NB names changed to match readme schema: UserID::MovieID::Rating::Timestamp

#MOVIESSCHEMA\_SQL={"MovieID":sqlalchemy.types.INTEGER(), "Title":sqlalchemy.types.NVARCHAR(length=255), "Genres":sqlalchemy.types.NVARCHAR(length=255)}#NB names changed to match readme schema: MovieID::Title::Genres

#USERSSCHEMA=\_SQL={"UserID":sqlalchemy.types.INTEGER(), "Gender":sqlalchemy.types.NVARCHAR(length=255), "Age":sqlalchemy.types.Float(precision=3, asdecimal=True), "Occupation":sqlalchemy.types.INTEGER(), "ZipCode":sqlalchemy.types.NVARCHAR(length=255)}#NB names changed to match readme schema: UserID::Gender::Age::Occupation::Zip-code

#get ratings, movies and users data according to schema and...

TRAININGPROPORTION = 1.0#...training proportion (initially set ot 1.0 i.e. NO TEST DATA)

#this is because of the way CCAI splits the data

ratingstrainingdata, ratingstestdata, ratingsrowcount, ratingsschemaout = datainout.splitdatdata(trainingproportion=TRAININGPROPORTION, datFILE=PATH\_RATINGS, datSCHEMA=RATINGSSCHEMA)

moviestrainingdata, moviestestdata, moviesrowcount, moviesschemaout = datainout.splitdatdata(trainingproportion=TRAININGPROPORTION, datFILE=PATH\_MOVIES, datSCHEMA=MOVIESSCHEMA)

userstrainingdata, userstestdata, usersrowcount, usersschemaout = datainout.splitdatdata(trainingproportion=TRAININGPROPORTION, datFILE=PATH\_USERS, datSCHEMA=USERSSCHEMA)

#sort out problematic user fields (e.g. Zip code)

userstrainingdata = datainout.treatuserfile(userstrainingdata)

#load to a sql database

datainout.loadtosqlstage(df =ratingstrainingdata, stagetablename="tblRating", ifexists="fail")

datainout.loadtosqlstage(df =moviestrainingdata, stagetablename="tblMovie", ifexists="fail")

datainout.loadtosqlstage(df =userstrainingdata, stagetablename="tblUser", ifexists="fail")

#get release year for movie intra-SQL based on what we already have

if not datainout.fieldexists("tblMovie", "ReleaseYear"):

PATH\_YEARSQL = os.path.join(PATH\_ME, "Split Movie Year.txt")

datainout.runsqltxt(script=PATH\_YEARSQL)

#build age and occupation based off readme data

dictAge={

1:"Under 18",

18:"18-24",

25:"25-34",

35:"35-44",

45:"45-49",

50:"50-55",

56:"56+"}

dictOccupation={

0:"other or not specified",

1:"academic/educator",

2:"artist",

3:"clerical/admin",

4:"college/grad student",

5:"customer service",

6:"doctor/health care",

7:"executive/managerial",

8:"farmer",

9:"homemaker",

10:"K-12 student",

11:"lawyer",

12:"programmer",

13:"retired",

14:"sales/marketing",

15:"scientist",

16:"self-employed",

17:"technician/engineer",

18:"tradesman/craftsman",

19:"unemployed",

20:"writer"}

dfAge = pd.DataFrame.from\_dict(data=dictAge, orient='index',columns=["Age"])

datainout.loadtosqlstage(df=dfAge, stagetablename="tblAge", ifexists="fail",index=True,indexlabel="AgeID")

dfOccupation = pd.DataFrame.from\_dict(data=dictOccupation, orient='index',columns=["Occupation"])

datainout.loadtosqlstage(df=dfOccupation, stagetablename="tblOccupation", ifexists="fail",index=True,indexlabel="OccupationID")

#build a genre table intra-SQL based on what we already have

if not datainout.tableexists("tblGenre"):

PATH\_GENRESQL = os.path.join(PATH\_ME, "Build Movie x Genre.txt")

datainout.runsqltxt(script=PATH\_GENRESQL)

#to this line the code is original, from this point the code echoes CCAI more closely

#############################START OF CCAI TRAINING MODEL#############################

"""

CCOM

For our model, the input is going to contain X neurons, where X is

the amount of movies in our dataset. Each of these neurons will possess

a normalized rating value varying from 0 to 1 -- 0 meaning that a user

has not watched that movie and the closer the value is to 1, the more

the user likes the movie that neuron's representing. These normalized

values, of course, will be extracted and normalized from the ratings

dataset.

After passing in the input, we train the RBM on it and have the hidden

layer learn its features. These features are what we use to reconstruct

the input, which in our case, will predict the ratings for movies that

the input hasn't watched, which is exactly what we can use to recommend

movies!

We will now begin to format our dataset to follow the model's expected input

"""

#get a new unborken ID (there are missing MovieIDs)

moviestrainingdata['List Index'] = moviestrainingdata.index #PM duplicate with my code to be tidied if time

#CCAI Merging movies\_df with ratings\_df by MovieID

merged\_df = moviestrainingdata.merge(ratingstrainingdata, on='MovieID')

#CCAI Dropping unecessary columns

merged\_df = merged\_df.drop('Timestamp', axis=1).drop('Title', axis=1).drop('Genres', axis=1)

#Group up by UserID

userGroup = merged\_df.groupby('UserID')

"""

CCOM

Now, we can start formatting the data into input for the RBM.

We're going to store the normalized users ratings into a list of lists called trX.

"""

#CCAI: Amount of users used for training

amountOfUsedUsers = 1000 #PM changed from 1000

#CCAI: Creating the training list

***trX = []***

***#CCAI: For each user in the group***

***for userID, curUser in userGroup:***

***#CCAI: Create a temp that stores every movie's rating***

***temp = [0]\*moviesrowcount#PM changed from len(movies\_df)***

***#CCAI: For each movie in curUser's movie list***

***for num, movie in curUser.iterrows():***

***#CCAI: Divide the rating by 5 and store it***

***temp[movie['List Index']] = movie['Rating']/5.0***

***#CCAI: Now add the list of ratings into the training list***

***trX.append(temp)***

***#CCAI: Check to see if we finished adding in the amount of users for training***

***if amountOfUsedUsers == 0:***

***break***

***amountOfUsedUsers -= 1***

"""

PM

NB up to this point we haven't done anything in Tensorflow,

we've just rearranged the data so that users movies and ratings

are iterable

"""

hiddenUnits = 40 #PM changed from 20

visibleUnits = moviesrowcount#PM changed from len(movies\_df)

#CCAI: Phase 1: Input Processing

sess = tf.Session()

sess.run(tf.global\_variables\_initializer())

***import LearnModel***

***prv\_w, prv\_vb, prv\_hb, W, vb, hb, v0 = LearnModel.buildviewinghabits(sess,trX,hiddenUnits, visibleUnits)***

"""

CCOM

Recommendation

We can now predict movies that an arbitrarily selected user might like.

This can be accomplished by feeding in the user's watched movie

preferences into the RBM and then reconstructing the input.

The values that the RBM gives us will attempt to estimate the user's preferences

for movies that he hasn't watched based on the preferences of the users that

the RBM was trained on.

"""

#PM Selecting the input user coverted to loop

#usrs =[]

for pu in range(len(trX)):

#CCIA: Feeding in the user and reconstructing the input

UserID = merged\_df.iloc[pu]["UserID"]

hh0 = tf.nn.sigmoid(tf.matmul(v0, W) + hb)

vv1 = tf.nn.sigmoid(tf.matmul(hh0, tf.transpose(W)) + vb)

feed = sess.run(hh0, feed\_dict={ v0: [trX[pu]], W: prv\_w, hb: prv\_hb})

rec = sess.run(vv1, feed\_dict={ hh0: feed, W: prv\_w, vb: prv\_vb})

#CCIA: We can then list the 20 most recommended movies for our mock user by sorting it by their scores given by our model.

scored\_movies\_df = moviestrainingdata #PM names changed to a generic name

scored\_movies\_df["Recommendation Score"] = rec[0]

px = scored\_movies\_df.sort\_values(["Recommendation Score"], ascending=False).head(20)

px["UserID"] = UserID

datainout.loadtosqlstage(df=px, stagetablename="tblRecommendation", ifexists="append")

if pu % 25 == 0:

print("Now loaded through " + str(pu) + " iterations")

print("fin")

### Appendix 7.2: Recommendations data model

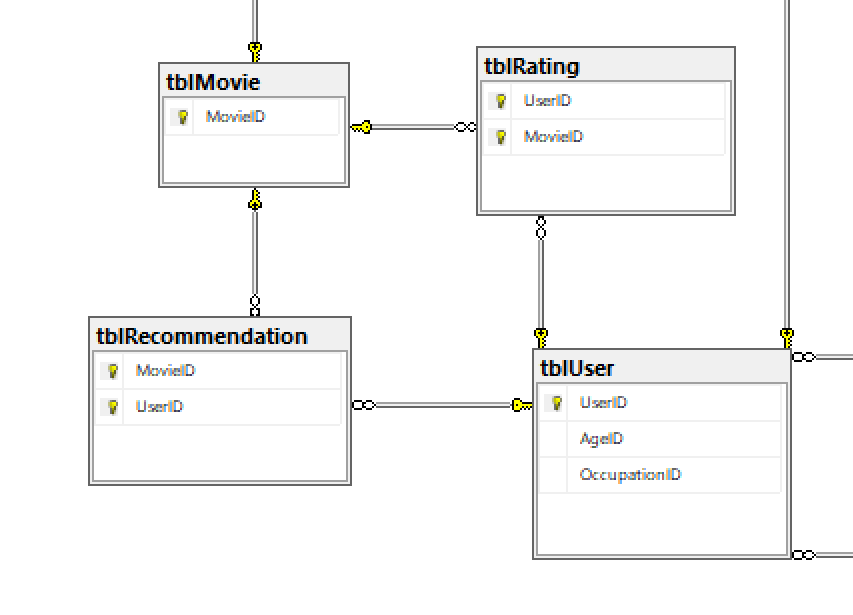


Figure A 7.1: Recommendation data model

## Appendix Eight: Chi-square tests for differences in User Demographic

The below tests for significant differences in the basic user demographic:

### Appendix 8.1: SQL/Python code to isolate Gender vs Genre

NB this code uses SQL Server 2017’s Python engine

SET QUOTED\_IDENTIFIER ON

SET ANSI\_NULLS ON

GO

CREATE PROCEDURE dbo.usp\_PM\_XTab\_GenderxGenre

AS

BEGIN

DECLARE @PySQL NVARCHAR(512)

= 'SELECT vmgd.Gender

, vmgd.Genre

, COUNT(DISTINCT vmgd.MovieID) AS MovieCount

FROM dbo.vMoviesxGenreDenormalised AS vmgd

GROUP BY vmgd.Gender

, vmgd.Genre ';

EXEC sys.sp\_execute\_external\_script @language = N'Python'

, @script = N'

OutputDataSet = PyInput.pivot( index="Genre", columns="Gender", values = "MovieCount" ).reset\_index(drop=True)

'

, @input\_data\_1\_name = N'PyInput'

, @input\_data\_1 = @PySQL

WITH RESULT SETS

(

(

Female SMALLINT,

Male SMALLINT

)

);

END;

GO

### Appendix 8.2: Chi-square test in R

library("odbc")

library("dplyr")

con <- dbConnect(odbc(),

Driver = "/usr/local/lib/libtdsodbc.so",

Server = "10.37.129.10",

Database = "Movies",

UID = "sa",

PWD = "AnyOldPassword",

Port = 5171)

result <- dbSendQuery(con, "EXEC dbo.usp\_PM\_XTab\_GenderxGenre")

rest <- dbFetch(result)

chisq.test(rest)

## Appendix Nine: K-means code

﻿#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Mon Jul 2 12:34:58 2018

@author: petermoore

PCA code inpired by http://scikit-learn.org/stable/auto\_examples/decomposition/plot\_pca\_iris.html

and http://jotterbach.github.io/2016/03/24/Principal\_Component\_Analysis/

"elbow" curve inspired by http://www.michaeljgrogan.com/k-means-clustering-python-sklearn/

The following quote about Zip Codes comes from https://people.howstuffworks.com/usps4.htm

"The first digit represents the state. Numbers increase as you move west. Several states share each digit — 2,

for example, represents the District of Columbia, Maryland, North Carolina, South Carolina, Virginia and West Virginia."

"""

import numpy as np

from matplotlib import pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

from sklearn.cluster import KMeans

import datainout

plt.rcParams['figure.figsize'] = (5,4)

BASEDIR="/Users/petermoore/Documents/GitHub/Movies/Trainspotting Two";MOVIEDIR = BASEDIR+"/ml-1m";USERSFILE = MOVIEDIR+"/users.dat"

USERSSCHEMA={"UserID":np.int32, "Gender":np.object, "Age":np.float32, "Occupation":np.int32, "ZipCode":np.object}#NB names changed to match readme schema: UserID::Gender::Age::Occupation::Zip-code

userstrainingdata, userstestdata, usersrowcount, usersschemaout = datainout.splitdatdata(trainingproportion=1.0, datFILE=USERSFILE, datSCHEMA=USERSSCHEMA)

r,c=userstrainingdata.shape

userlocal = userstrainingdata

userlocal.insert(loc=c, column="ShortZip", value=userstrainingdata.ZipCode.str.slice(0, 2).astype(np.int32), allow\_duplicates=False)# first two digits of zip code

userlocal["GenderN"] = 0.0

def normalizedata(Y): return (Y - Y.mean()) / (Y.max() - Y.min())

userlocal.loc[userlocal["Gender"] == "M","GenderN"] = 0 # make male and female numeric (because pca likes floats)

userlocal.loc[userlocal["Gender"] == "F","GenderN"] = 1

userlocal["AgeN"] = normalizedata(userlocal["Age"])

userlocal["OccupationN"] = normalizedata(userlocal["Occupation"])

userlocal["ShortZipN"] = normalizedata(userlocal["ShortZip"])

X=userlocal[["GenderN", "Occupation", "Age"]]

#try some Kmeans

ntrials = 20

Nc = range(1,ntrials+1)

#fieldlist = ["GenderN", "OccupationN", "AgeN", "ShortZipN"]

ltitle = "Elbow curve for: "

for l in list(X):#fieldlist:

Y = X[[l]]

kmeans = [KMeans(n\_clusters=i) for i in Nc]

score = [kmeans[i].fit(Y).score(Y) for i in range(len(kmeans))]

plt.plot(Nc,score, label =l)

plt.xlabel('Number of Clusters')

plt.ylabel('Score')

ltitle = ltitle + l +','

plt.title(ltitle)

leg = plt.legend(loc='best', ncol=2, mode="expand", shadow=True, fancybox=True)

leg.get\_frame().set\_alpha(0.5)

plt.show()

X=X.values#copnvert to numpy array

fig=plt.figure()

ax = Axes3D(fig)

ax.scatter(X[:,0], X[:,1], X[:,2])

k = 3

kmeans = KMeans(n\_clusters=k)

kmeans = kmeans.fit(X)

labels = kmeans.predict(X)

C=kmeans.cluster\_centers\_

fig=plt.figure()

ax = Axes3D(fig)

ax.scatter(X[:,0], X[:,1], X[:,2])

ax.scatter(C[:,0], C[:,1], C[:,2], marker="\*",c="#050505", s=1000)

### Appendix Nine Addendum: PCA

Brief attempt to get PCA working:

pca = PCA(n\_components=1).fit(Y)

pca\_d = pca.transform(Y)

pca\_c = pca.transform(X)

kmeans=KMeans(n\_clusters=3)

kmeansoutput=kmeans.fit(Y)

kmeansoutput

pl.figure('3 Cluster K-Means')

pl.scatter(pca\_c[:, 0], pca\_d[:, 0], c=kmeansoutput.labels\_)

pl.xlabel('age')

pl.ylabel('zip')

pl.title('3 Cluster K-Means')

pl.show()

## Appendix Ten: Association rules

The following code uses association to link users who liked the same movies. The followhg data model was used to store the output.

### Appendix 10.1: Association rules Python code

### ﻿#!/usr/bin/env python3

### # -\*- coding: utf-8 -\*-

### """

### Created on Tue Jul 3 06:06:34 2018

### @author: petermoore

### Inspired by http://pbpython.com/market-basket-analysis.html

### """

### import os

### #import numpy as np

### import pandas as pd

### pd.set\_option('display.max\_rows', 500)

### pd.set\_option('display.max\_columns', 5)

### #change working directory

### PATH\_FILE = "/Users/petermoore/Documents/GitHub/Movies/Trainspotting Three"#os.path.dirname(os.path.realpath(\_\_file\_\_))

### os.chdir(PATH\_FILE)

### import datainout

### PATH\_ME = os.getcwd()

### #read amalgamated data from data model

### sql="""SELECT tu.UserID

### , tr.Rating

### , tm.MovieID

### , tm.Title

### , ta.Age

### FROM dbo.tblRating AS tr

### INNER JOIN dbo.tblMovie AS tm

### ON tm.MovieID = tr.MovieID

### INNER JOIN dbo.tblUser AS tu

### ON tu.UserID = tr.UserID

### INNER JOIN dbo.tblAge ta

### ON ta.AgeID = tu.AgeID

### INNER JOIN dbo.tblRecommendation trm

### ON trm.UserID = tu.UserID

### AND trm.MovieID = tm.MovieID"""

### #get dataframe from sql statement

### df = datainout.sqldf(sql)

### #this is very close to the code at http://pbpython.com/market-basket-analysis.html

### basket = (df#[df['Age'] =="18-24"]

### .groupby(['UserID', 'MovieID'])['Rating']

### .mean().unstack().reset\_index().fillna(0) #take the mean, just in case some one has rated the same movie twice

### .set\_index('UserID'))

### #invoke the mlxtend repository

### from mlxtend.frequent\_patterns import apriori

### from mlxtend.frequent\_patterns import association\_rules

### #function a direct copy from http://pbpython.com/market-basket-analysis.html

### """

### from http://pbpython.com/market-basket-analysis.html

### There are a lot of zeros in the data but we also need to make sure

### any positive values are converted to a 1 and anything less the 0 is set to 0.

### This step will complete the one hot encoding of the data and

### remove the postage column (since that charge is not one we wish to explore

### """

### def encode\_units(x):

### if x <= 0:

### return 0

### if x>=1:

### return 1

### #from http://pbpython.com/market-basket-analysis.html

### basket\_sets = basket.applymap(encode\_units)

### #get items with support of at least 20% (number kept low to get some data)

### *frequent\_itemsets = apriori(basket\_sets,min\_support=0.07,use\_colnames=True)*

### *#get the association rules*

### *rules = association\_rules(frequent\_itemsets, metric = "lift", min\_threshold=1)*

### #only retain those rules with a lift over 1.2 and over 20% confidence

### rules2 = rules[ (rules['lift'] >= 2.0) &

### (rules['confidence'] >= 0.2)]

### #import association rules to SQL

### import pyodbc

### conn = pyodbc.connect('DSN=MYMSSQL\_MOVIES;UID=sa;PWD=l00katy0urd%t%a', autocommit=True)

### crsr = conn.cursor()

### crsr.execute("DELETE FROM dbo.tblRuleAsssociationxRulePrecedent")

### crsr.execute("DELETE FROM dbo.tblRuleAsssociation")

### crsr.close()

### insertmajor = """

### EXEC dbo.usp\_PM\_RuleAsssociation\_Insert @antecedentsupport = ?

### , @consequentsupport = ?

### , @support = ?

### , @confidence = ?

### , @lift = ?

### , @leverage = ?

### , @conviction = ?

### """

### insertminor = """

### EXEC dbo.usp\_PM\_RuleAsssociationxRulePrecedent\_Insert

### @RuleAsssociationFK = ?

### , @RulePrecedentFK = ?

### , @MovieFK =?

### """

### for index, row in rules2.iterrows():

### antecedants=(row['antecedants'])

### consequents=(row['consequents'])

### paramsmajor=[row['antecedent support'],row['consequent support'],row['support'],

### row['confidence'],row['lift'],row['leverage'],row['conviction']]

### with conn.cursor() as cur:

### RuleAsssociationID = cur.execute(insertmajor, paramsmajor).fetchall()

### RuleAsssociationID = [int(L[0]) for L in RuleAsssociationID]

### RuleAsssociationID = (int(RuleAsssociationID[0]))

### for \_p in list(antecedants):

### paramsminor = [RuleAsssociationID,1,\_p]

### cur.execute(insertminor, paramsminor)

### for \_q in list(consequents):

### paramsminor = [RuleAsssociationID,2,\_q]

### cur.execute(insertminor, paramsminor)

### conn.close()

### Appendix 10.2: Association rules data model

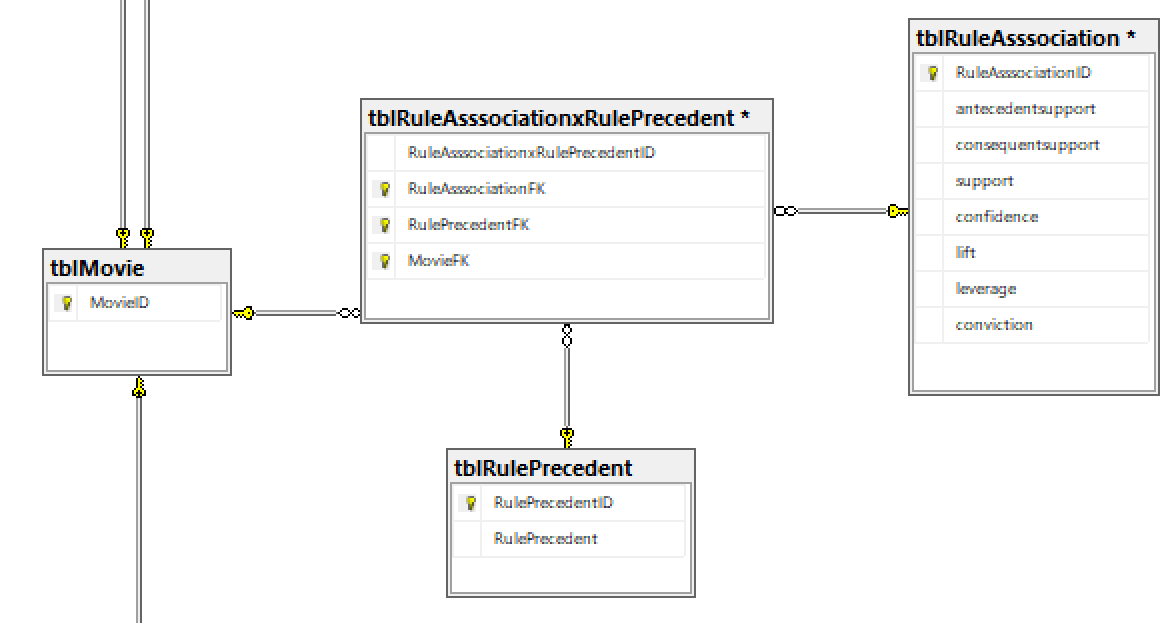


Figure A 10.1: Rule association data model

## Appendix Eleven: collaborative filtering

Code for a not-implemented collaborative filtering model.

"""

Created on Tue Jul 3 06:06:34 2018

@author: petermoore

Inspired by http://www.awesomestats.in/python-recommending-movies/

"""

import os

#import numpy as np

import numpy as np

import pandas as pd

pd.set\_option('display.max\_rows', 500)

pd.set\_option('display.max\_columns', 500)

#change working directory

PATH\_FILE = "/Users/petermoore/Documents/GitHub/Movies/Trainspotting Three"#os.path.dirname(os.path.realpath(\_\_file\_\_))

os.chdir(PATH\_FILE)

import datainout

PATH\_ME = os.getcwd()

sql="""SELECT tu.UserID

, tr.Rating

, tm.MovieID

, tm.Title

, ta.Age

FROM dbo.tblRating AS tr

INNER JOIN dbo.tblMovie AS tm

ON tm.MovieID = tr.MovieID

INNER JOIN dbo.tblUser AS tu

ON tu.UserID = tr.UserID

INNER JOIN dbo.tblAge ta

ON ta.AgeID = tu.AgeID

INNER JOIN dbo.tblRecommendation trm

ON trm.UserID = tu.UserID

AND trm.MovieID = tm.MovieID;"""

df = datainout.sqldf(sql)

rating\_df = datainout.sqldf("select \* from tblRating")

movies\_df = datainout.sqldf("select \* from tblMovie")

rating\_df.drop( "Timestamp", inplace = True, axis = 1 )

from sklearn.metrics import pairwise\_distances

#Create the pivot tablen(straight from http://www.awesomestats.in/python-recommending-movies/)

user\_movies\_df = rating\_df.pivot( index='UserID', columns='MovieID', values = "Rating" ).reset\_index(drop=True)

user\_movies\_df.fillna( 0, inplace = True )

user\_sim = 1 - pairwise\_distances(user\_movies\_df.as\_matrix(), metric="cosine" )

np.fill\_diagonal( user\_sim, 0 )

user\_sim\_df = pd.DataFrame(user\_sim)

user\_sim\_df.set\_index(index)

user\_sim\_dfT = user\_sim\_df.T

type(user\_sim\_dfT)

user\_sim\_dfT.shape

datainout.loadtosqlstage(df=user\_sim\_df, stagetablename="tblUserxUserCollaborativeFiltering", ifexists="replace")

## Appendix Twelve: SQL code for “ROC curve”

DROP PROCEDURE IF EXISTS dbo.usp\_PM\_ROC\_Curve;

GO

CREATE PROCEDURE dbo.usp\_PM\_ROC\_Curve

AS

SELECT ra.RuleAsssociationID

, ra.antecedentsupport

, ra.consequentsupport

, ra.support

, ra.confidence

, ra.lift

, ra.leverage

, ra.conviction

, x.MovieID AS AntecendantMovieID

, y.MovieID AS ConsequentMovieID

, x.Title AS AntecendantTitle

, y.Title AS ConsequentTitle

INTO dbo.#MovieTruth

FROM dbo.tblRuleAsssociation AS ra

INNER JOIN

(SELECT \* FROM dbo.vPrecedent WHERE RulePrecedentID = 1) x --Antecedant

ON x.RuleAsssociationID = ra.RuleAsssociationID

INNER JOIN

(SELECT \* FROM dbo.vPrecedent WHERE RulePrecedentID = 2) y --Consequent

ON y.RuleAsssociationID = ra.RuleAsssociationID;

SELECT usr.UserID

, tm.MovieID

INTO dbo.#TensorRecommends

FROM dbo.tblRecommendation AS rec

INNER JOIN dbo.tblUser usr

ON usr.UserID = rec.UserID

INNER JOIN dbo.tblMovie AS tm

ON tm.MovieID = rec.MovieID

WHERE usr.UserID IN

(

SELECT UserID FROM dbo.tblUserSample

);

SELECT usr.UserID

, tm.MovieID

INTO dbo.#AlreadySeen

FROM dbo.tblRating AS rat

INNER JOIN dbo.tblUser usr

ON usr.UserID = rat.UserID

INNER JOIN dbo.tblMovie AS tm

ON tm.MovieID = rat.MovieID

WHERE usr.UserID IN

(

SELECT UserID FROM dbo.tblUserSample

);

UPDATE tusm

SET tusm.Recommended = 1

FROM dbo.tblUserSamplexMovie AS tusm

INNER JOIN dbo.tblUser AS tu

ON tu.UserID = tusm.UserID

INNER JOIN dbo.tblMovie AS tm

ON tm.MovieID = tusm.MovieID

INNER JOIN dbo.#TensorRecommends AS tr

ON tr.UserID = tu.UserID

AND tr.MovieID = tm.MovieID;

UPDATE tusm

SET tusm.HasAntecendants = 1

, tusm.Lift = mt.lift

FROM dbo.tblUserSamplexMovie AS tusm

INNER JOIN dbo.tblUser AS tu

ON tu.UserID = tusm.UserID

INNER JOIN dbo.tblMovie AS tm

ON tm.MovieID = tusm.MovieID

INNER JOIN dbo.#MovieTruth mt

ON tm.MovieID = mt.ConsequentMovieID --movie in row is the consequential movie

INNER JOIN dbo.#AlreadySeen AS ase

ON mt.AntecendantMovieID = ase.MovieID -- looking for the antecendent movie

AND ase.UserID = tu.UserID; --to exist for that user

SELECT uxs.Lift

, SUM( CASE

WHEN Recommended = 1

AND HasAntecendants = 1 THEN

1.0

ELSE

0.0

END

) AS TruePositive

, SUM( CASE

WHEN Recommended = 1

AND HasAntecendants = 0 THEN

1.0

ELSE

0.0

END

) AS FalsePositive

, SUM( CASE

WHEN Recommended = 0

AND HasAntecendants = 1 THEN

1.0

ELSE

0.0

END

) AS FalseNegative

, SUM( CASE

WHEN Recommended = 0

AND HasAntecendants = 0 THEN

1.0

ELSE

0.0

END

) AS TrueNegative

, CAST(0 AS FLOAT) AS FPR

, CAST(0 AS FLOAT) AS TPR

INTO dbo.#StayPositive

FROM dbo.tblUserSamplexMovie uxs

GROUP BY uxs.Lift;

UPDATE dbo.#StayPositive

SET TPR = TruePositive / (TruePositive + FalseNegative)

, FPR = FalsePositive / (FalsePositive + TrueNegative)

WHERE (TruePositive + FalseNegative) > 0

AND (FalsePositive + TrueNegative) > 0;

SELECT \*

FROM dbo.#StayPositive;

DROP TABLE dbo.#MovieTruth;

DROP TABLE dbo.#TensorRecommends;

DROP TABLE dbo.#AlreadySeen;

DROP TABLE dbo.#StayPositive;

## References

1. For a detailed understanding of the data mining objectives see: [Appendix 0: Business Objectives](#_Appendix_0:_Business) [↑](#footnote-ref-1)
2. https://ai.google/research/teams/brain [↑](#endnote-ref-1)
3. https://www.tensorflow.org [↑](#endnote-ref-2)
4. https://www.python.org [↑](#endnote-ref-3)
5. Author’s note: code samples are a hybrid of brand new code and pre-existing code; the distinctions are clearly highlighted in the code comments wherein references to ancillary external sources may also be found. [↑](#footnote-ref-2)
6. https://github.com/acobley/TF-recomm [↑](#endnote-ref-4)
7. https://github.com/acobley [↑](#endnote-ref-5)
8. https://github.com/songgc/TF-recomm/ [↑](#endnote-ref-6)
9. https://github.com/songgc [↑](#endnote-ref-7)
10. https://grouplens.org/datasets/movielens/ [↑](#endnote-ref-8)
11. <https://movielens.org> [↑](#endnote-ref-9)
12. http://files.grouplens.org/datasets/movielens/ml-1m-README.txt [↑](#footnote-ref-3)
13. https://www.microsoft.com/en-gb/sql-server/sql-server-2017 [↑](#endnote-ref-10)
14. https://www.r-project.org [↑](#endnote-ref-11)
15. Ribeiro, M.T., Singh, S. and Guestrin, C., 2016, August. Why should I trust you?: Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 1135-1144). ACM. [↑](#endnote-ref-12)
16. This is beyond the scope of the project but still a deeply non-trivial point; as Ribeiro et al argue: “trust is crucial for effective human interaction with machine learning systems…explaining individual predictions is important in assessing trust” I do not explain my decisions here! [↑](#footnote-ref-4)
17. http://grouplens.org/datasets/movielens/10m/ [↑](#endnote-ref-13)
18. https://www.tensorflow.org/guide/tensors [↑](#endnote-ref-14)
19. https://www.tensorflow.org/guide/graphs [↑](#endnote-ref-15)
20. https://www.apple.com/uk/icloud/ [↑](#endnote-ref-16)
21. Placed in main body as fundamental [↑](#footnote-ref-5)
22. http://files.grouplens.org/datasets/movielens/ml-1m-README.txt [↑](#endnote-ref-17)
23. This seems fair it is unlikely that viewer habits change depending on which floor of a building one is on [↑](#footnote-ref-6)
24. In addition, the pbix file that created them is included the accompanying zip file [↑](#footnote-ref-7)
25. Author’s note: just for the record I did indeed get the Telnet session working [↑](#footnote-ref-8)
26. Author’s note: this decision was made entirely on the basis of ease of *reading* code and was a direct consequence of the Zoom session held between the group wherein it was referenced by Ged Walls [↑](#footnote-ref-9)
27. <http://cognitiveclass.ai> (formerly Big Data University) [↑](#footnote-ref-10)
28. This is a recreation of [Appendix 5.3: Variation by gender and genre](#Appendix 5.3: Variation by gender and genre) [↑](#footnote-ref-11)
29. Here we had used AgeID (which can just about be considered continuous); and occupation ID (which can’t) [↑](#footnote-ref-12)
30. https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwirloHb06bcAhXjKsAKHRJVCbEQFggpMAA&url=https%3A%2F%2Fgithub.com%2Frasbt%2Fmlxtend&usg=AOvVaw3AlsHBT\_HS5ejzalNzQoDo [↑](#endnote-ref-18)
31. The code produced veers quite far from but as inspired by: <http://pbpython.com/market-basket-analysis.html> [↑](#footnote-ref-13)
32. The salient code in the appendix is italicised [↑](#footnote-ref-14)
33. Loosely these terms mean the following: Support is how often, Confidence is how reliable the rule. And lift is the distance from independence. [↑](#footnote-ref-15)
34. A similar method to association rules is called collaborative filtering as seen here: ﻿<http://www.awesomestats.in/python-recommending-movies/> This was assessed but ultimately came second to association rules. The code can be found here: [Appendix Eleven: collaborative filtering](#Appendix Eleven: collaborative filtering) [↑](#footnote-ref-16)
35. Has antecedents *that the user has seen* [↑](#footnote-ref-17)
36. Has antecedents *that the user has seen* [↑](#footnote-ref-18)
37. One can build a ROC curve from this data but it is user agnostic and thus irrelevant (or relevant for something else) [↑](#footnote-ref-19)