PH125.9x - HarvardX Capstone Project : MovieLens

HM

9/10/2020

Contents

Introduction

Description of the Dataset

The dataset we are going to work with is the Movielens 10M dataset. It consists of ten million movie ratings. We use the provided code to create two datasets: one for training and one for test, each of them containing the following features:

Variable	Description (Type)
`userId`	Used to identify users (Integer)
`movieId`	Used to identify movies (Integer)
`rating`	Rating of the movie by user (Double)
`timestamp`	Timestamp of the date of the rating (Integer)
`title`	Movie Title and Year (Character)
`genres`	Genres of the movie (Character)

Lets visualize the first few rows of the training set `edx` to get an idea of what we are dealing with:

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	231	5	838983392	Dumb & Dumber (1994)	Comedy
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi

There are 9000057 items in the training dataset `edx` and 999997 items in the test dataset `validation`. In total, the full dataset contains 10000054 items.

Analyse the Data

Though training set `edx` contains 999997 items, it only contains 10677 distinct movies and 69878 distinct users.

The `edx` dataset as well as the `validation` dataset have been "cleaned" in such a way that movies or users only appearing once have been removed.

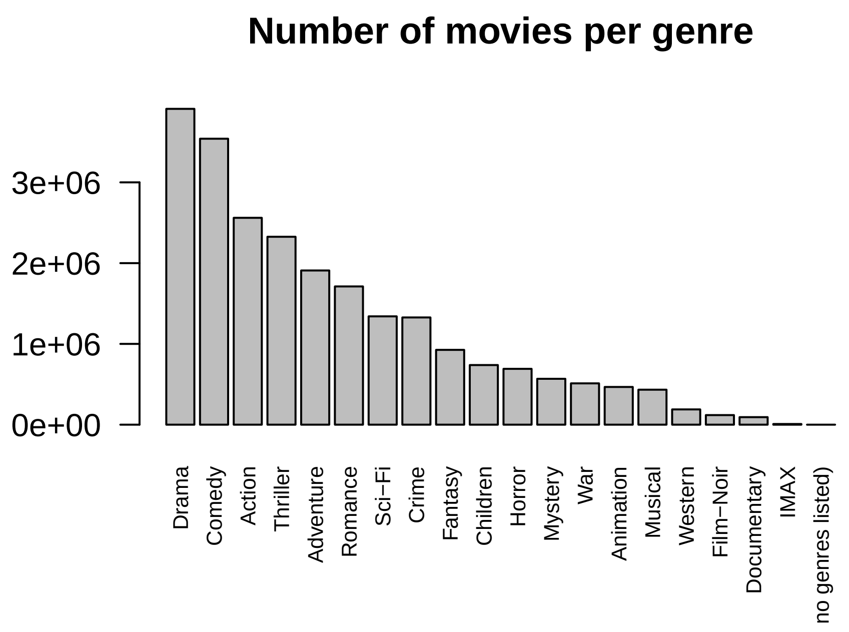
Here are the most rated movies in the `edx` dataset:

title	number
Pulp Fiction (1994)	31289
Forrest Gump (1994)	31055
Silence of the Lambs, The (1991)	30274
Jurassic Park (1993)	29324
Shawshank Redemption, The (1994)	27975
Braveheart (1995)	26272

Notice that they appear to be "classic" movies, most of them being from the 1990s.

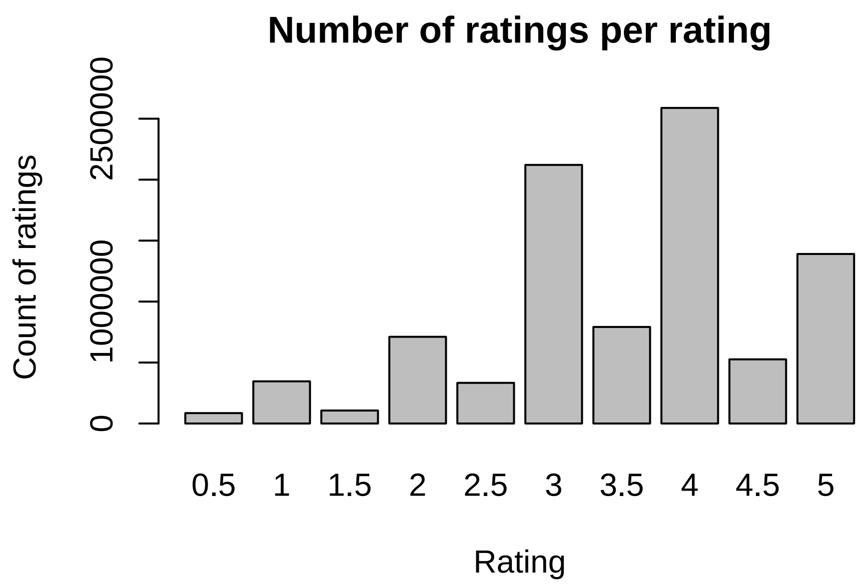
On average each movie has been rated 843 times with a standard deviation of 2238.3041539. This shows that there is a great disparity between movies: some are much more popular than others and have therefore been rated many times

We also notice that some genres get far more ratings than others:



Here we see that drama, comedy and action genres are the most rated whereas film-noir, documentary and IMAX are the least.

Furthermore, by plotting the number of ratings we realize that whole ratings are more popular than half ratings:



Objective

Our goal is to correctly predict the rating a user is going to give to a movie. This could be useful for a movie recommendation algorithm, similar to the ones used by Netflix or Hulu. We will be testing our method on the `validation` dataset and report the RMSE (root-mean-square deviation) for each of the methods used.

$$RMSE = \sqrt{\frac{1}{N}\sum(y - \hat{y})^2}$$

With y being the predicted value, $\hat{y}$ being the corresponding expected value in the `validation` dataset, and N represents the number of items in total.

We will try to minimize the `RMSE`.

Methods

In order to accurately predict the user rating of a movie we will be conducting our analysis using different approaches. We will start with a basic prediction using only the overall mean, from there we will take into account the movie and user biases. Finally we will introduce the concept of regularization which will allow us to penalize movies with only a handful of ratings.

First Approach : overall mean

For this part we will suppose that every movie gets rated the same. We will compute the mean rating in the `edx` dataset and use this value as the predicted rating in the `validation` dataset using this code:

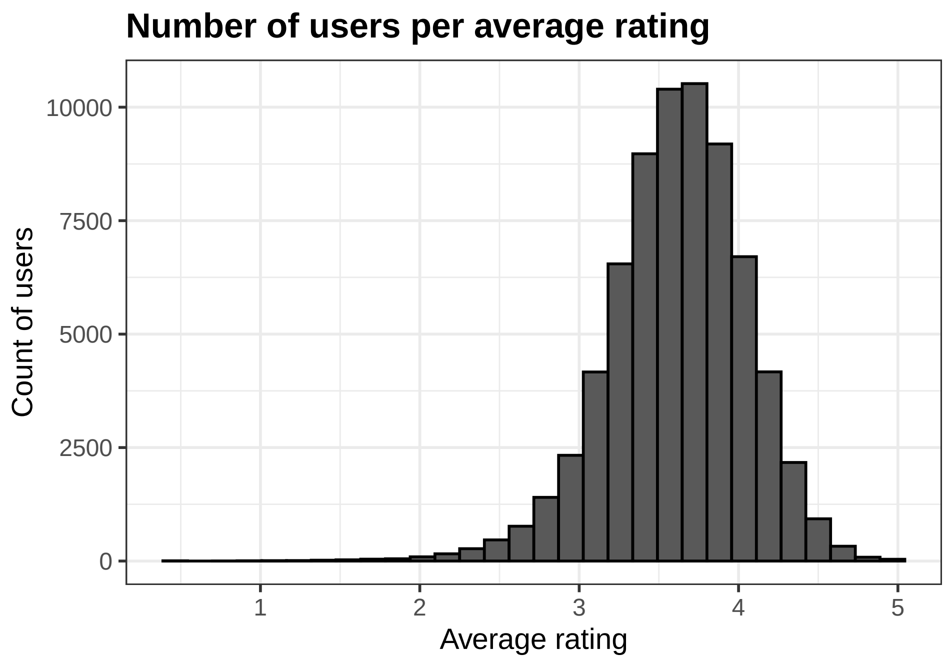
```
mu_hat <- mean(edx$rating)
```

This very simple approach doesn't take into account the fact that some movies are better rated than other or that some users tend to be more generous than others with their ratings. We will need to introduce the concept of bias.

Second Approach : user bias and movie bias

In this part we will study the impact of the user bias and of the movie bias on the rating of a movie.

We know from experience that some movies generally get better ratings and as seen below, some users tend to give higher ratings compared to others:



We need to compute the difference between each individual movie average and the average movie rating which we calculate earlier as being 3.5124581.

Using this code we compute the difference to average for each movie, we will call this the movie bias:

```
movie_averages <- edx %>%
  group_by(movieId) %>%
  summarise(b_m = mean(rating - mu_hat))
```

After that we can compute the predicted ratings for the movies in the `validation` dataset:

```
predicted_ratings <- mu_hat + validation %>%
  left_join(movie_averages, by='movieId') %>%
  pull(b_m)
```

We now have a vector containing 999997 items called `predicted_ratings` which are the predicted ratings for the respected movies in the `validation` dataset. This prediction includes the movie bias.

We can run the same code with a few tweaks to include the user bias and calculated the predicted ratings:

```
user_averages <- edx %>%
  left_join(movie_averages, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu_hat - b_m))


predicted_ratings_um_bias <- validation %>%
  left_join(movie_averages, by='movieId') %>%
  left_join(user_averages, by='userId') %>%
  mutate(pred = mu_hat + b_m + b_u) %>%
  pull(pred)
```

The result is a vector containing the predicted rating for each movie in the `validation` dataset taking into account the movie as well as the user bias.

Third Approach: regularization

For this final approach we will introduce the concept of regularization which will allow us to penalize movies with only a few ratings as these are more difficult to predict accurately. We will be trying to better understand the variability of the movie bias.

Lets inspect the best and worst movies in the `edx` dataset:

best_movies	worst_movies
Satan's Tango (Sátántangó) (1994)	Besotted (2001)
Shadows of Forgotten Ancestors (1964)	Hi-Line, The (1999)
Fighting Elegy (Kenka erejii) (1966)	Altered (2006)
Sun Alley (Sonnenallee) (1999)	Accused (Anklaget) (2005)
Blue Light, The (Das Blaue Licht) (1932)	War of the Worlds 2: The Next Wave (2008)
Human Condition II, The (Ningen no joken II) (1...	SuperBabies: Baby Geniuses 2 (2004)
Human Condition III, The (Ningen no joken III) ...	Hip Hop Witch, Da (2000)
Constantine's Sword (2007)	Disaster Movie (2008)

It appears that the best and worst movies have very few ratings so predicting their rating accurately would be impossible.

Our goal is to penalize the movie effect when few users have rated the movie. The regularization formula for calculating the movie bias using our penalty term `p` is:

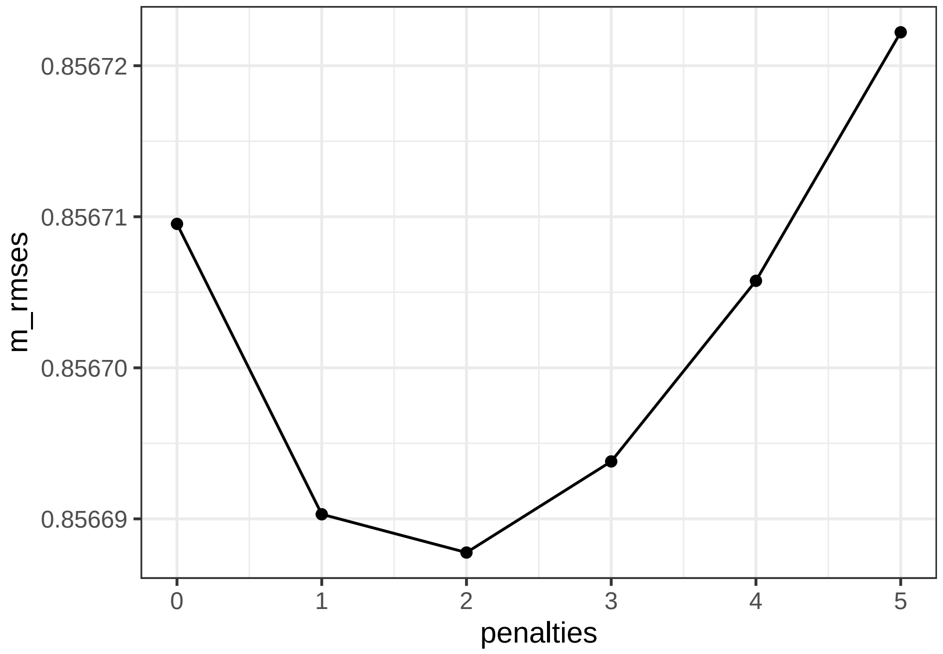
$$b_i(p) = \frac{1}{p+n} \sum (Y_i - \hat{\mu})$$

We need to choose our penalty term wisely as it must minimize our `RMSE`. We will calculate the `RMSE` of our predicted ratings on the `edx` dataset while testing several values for our penalty. To do so we will use this following code:

```r
# Create a list of penalties we want to calculate the RMSE with
penalties <- seq(0, 5, 1)

# Function to calculate the RMSE using the penalties list
m_rmses <- sapply(penalties, function(p){

  reg_movie_avgs <- edx %>%
    group_by(movieId) %>%
    summarize(regmoviebias = sum(rating - mu_hat)/(n()+p))

  predicted_ratings <-
    edx %>%
    left_join(reg_movie_avgs, by = "movieId") %>%
    left_join(user_averages, by = "userId") %>%
    mutate(regmoviebias = mu_hat + b_u + regmoviebias) %>%
    .$regmoviebias

  return(RMSE(predicted_ratings, edx$rating))

})

# Determine for which value of p the RMSE is the lowest
moviepenalty_optimal <- penalties[which.min(m_rmses)]
```

```
# Compute the movie averages using the calculated optimal penalty
reg_movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(regmoviebias = sum(rating - mu_hat)/(n()+moviepenalty_optimal))

# Compute the predicted ratings for the movies in the validation dataset
predicted_ratings <-
  validation %>%
  left_join(reg_movie_avgs, by = "movieId") %>%
  left_join(user_averages, by = "userId") %>%
  mutate(regmoviebias = mu_hat + b_u + regmoviebias) %>%
  .$regmoviebias
```

We then plot the `RMSE` against the values of the penalties:

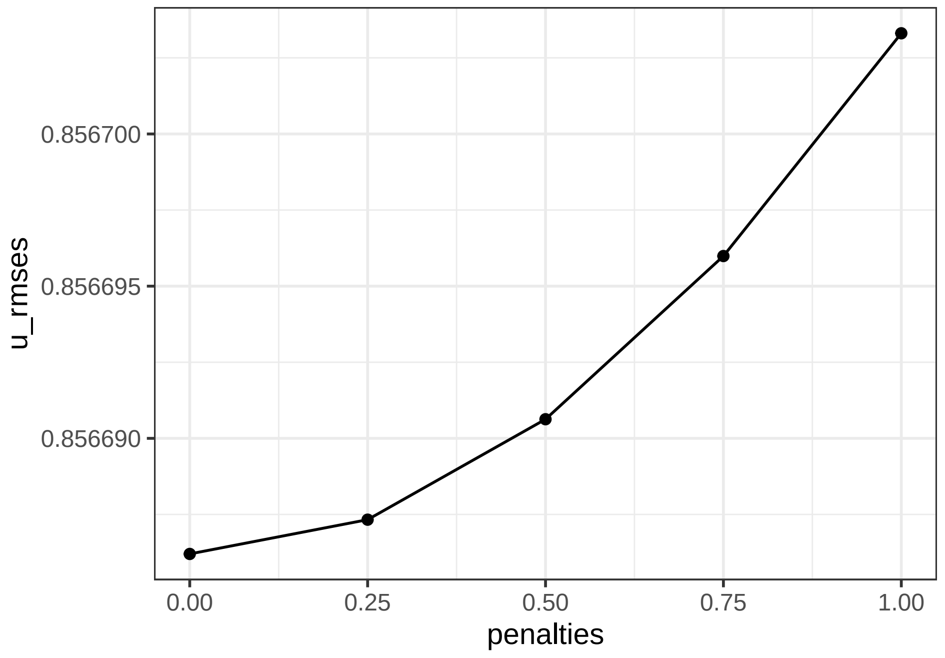


The minimum is obtained when `p` equals 2

We use the same method to determine the best value for `p` with both the movie effect and the user effect. We use the following code:

```
# Create a list of penalties we want to calculate the RMSE with
penalties <- seq(0, 1, 0.25)

# Function to calculate the RMSE using the penalties list
u_rmses <- sapply(penalties, function(p){

  reg_user_avgs <- edx %>%
    left_join(reg_movie_avgs, by="movieId") %>%
    group_by(userId) %>%
    summarize(reguserbias = sum(rating - regmoviebias - mu_hat)/(n()+p))

  predicted_ratings <-
    edx %>%
    left_join(reg_movie_avgs, by = "movieId") %>%
    left_join(reg_user_avgs, by = "userId") %>%
    mutate(regusermoviebias = mu_hat + regmoviebias + reguserbias) %>%
    .$regusermoviebias

  return(RMSE(predicted_ratings, edx$rating))

})

# Determine for which value of p the RMSE is the lowest
userpenalty_optimal <- penalties[which.min(u_rmses)]  #determine which is lowest

# Compute the user averages using the calculated optimal penalty
reg_user_avgs <- edx %>%
  left_join(reg_movie_avgs, by="movieId") %>%
  group_by(userId) %>%
  summarize(reguserbias = sum(rating - regmoviebias - mu_hat)/(n()+userpenalty_optimal))

# Compute the predicted ratings for the movies in the validation dataset
reg_predicted_ratings <-
  validation %>%
  left_join(reg_movie_avgs, by = "movieId") %>%
  left_join(reg_user_avgs, by = "userId") %>%
  mutate(regusermovie = mu_hat + regmoviebias + reguserbias) %>%
  .$regusermovie
```

We can then plot the same graph as before with the new penalty:



The minimum is obtained when p equals 0

We have successfully applied regularization to improve our prediction. However, can further improve our guess.

Final improvement

By looking at the highest and lowest movie prediction we stumble upon a problem: some of our predictions are either under 0.5 or higher than 5.

lowest_predictions	highest_predictions
-0.381522640969768	5.79406987762581
-0.293988932528662	5.79841938646495
-0.230655673089949	5.84429710676931
-0.178953187353531	5.87575539864135
-0.0703892511872484	5.87706129664826
-0.0222436502735726	5.88759058792462
0.0656271785603113	5.89832500206862

We can fix a lower and an upper limit to avoid this issue. We use the following code:

```
reg_predicted_ratings_limit <- pmax(pmin(predicted_ratings, 5), 0.5)
```

We now have limited the lowest ratings at 0.5 and the highest at 5.

lowest_predictions	highest_predictions
0.5	5
0.5	5
0.5	5
0.5	5
0.5	5
0.5	5
0.5	5

ML Appraoch

Using the h2o library and by rearranging our dataset using one-hot encoding we can further improve our prediction. We encourage the reader to look at the R code for the details on one-hot encoding and we use this code for the machine learning approach:

```
h2o.init(nthreads = -1, max_mem_size = "16G")
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         3 hours 14 minutes
##     H2O cluster timezone:       Europe/Paris
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.30.1.2
##     H2O cluster version age:    14 days, 4 hours and 15 minutes
##     H2O cluster name:           H2O_started_from_R_oganesson_ttf720
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   15.46 GB
##     H2O cluster total cores:    12
##     H2O cluster allowed cores:  12
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         Amazon S3, XGBoost, Algos, AutoML, Core V3, TargetEncoder, Core V4
##     R Version:                  R version 4.0.2 (2020-06-22)
```

```
##################
# Define the model in h2o

# turn the matrices into h2o objects
edx_h2o <- as.h2o(edx_gen_norm)
```

```
## Warning in use.package("data.table"): data.table cannot be used without R
## package bit64 version 0.9.7 or higher. Please upgrade to take advangage of
## data.table speedups.
```

```
##   |                                                                              |
```

```
val_h2o <- as.h2o(val_gen_norm)
```

```
## Warning in use.package("data.table"): data.table cannot be used without R
## package bit64 version 0.9.7 or higher. Please upgrade to take advangage of
## data.table speedups.
```

```
##   |                                                                              |
```

```
# Specify labels and predictors
y <- "rating"
x <- setdiff(names(edx_h2o), y)

# Turn the labels into categorical data.
edx_h2o[,y] <- as.factor(edx_h2o[,y])
val_h2o[,y] <- as.factor(val_h2o[,y])

# Train a deep learning model and validate on test set
```

```r
DL_model <- h2o.deeplearning(
  x = x,
  y = y,
  training_frame = edx_h2o,
  validation_frame = val_h2o,
  distribution = "AUTO",
  activation = "RectifierWithDropout",
  hidden = c(256, 256, 256, 256),
  input_dropout_ratio = 0.2,
  sparse = TRUE,
  epochs = 15,
  stopping_rounds = 3,
  stopping_tolerance = 0.01, #stops if it doesn't improve at least 0.1%
  stopping_metric = "AUTO",
  nfolds = 10,
  variable_importances = TRUE,
  shuffle_training_data = TRUE,
  mini_batch_size = 2000
)
```

```
## Warning in .h2o.processResponseWarnings(res): Dropping bad and constant columns: [Imax_u].

##   |                                                                              |
```

Results

We have presented the different methods we are going to evaluate. Each method will be used to predict the ratings in the validation dataset and we will compute the RMSE using this code:

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

RMSE First Approach

This is the most basic estimate so we are not expecting to be very accurate. We get the following RMSE:

```
RMSE_overallmean <- RMSE(validation$rating, mu_hat)
RMSE_overallmean
```

```
## [1] 1.060714
```

$$RMSE_{overallmean} = 1.060714$$

RMSE Second Approach

By computing the movie and user biases we should be able to significantly improve our RMSE. We get the following:

```
RMSE_biases <- RMSE(validation$rating, predicted_ratings_um_bias)
RMSE_biases
```

```
## [1] 0.8651915
```

$$RMSE_{biases} = 0.8651915$$

RMSE Third Approach

Now by using regularization we are going to penalize movies and users with very few ratings by introducing a penalty term in our equations.

```
RMSE_reg <- RMSE(validation$rating, reg_predicted_ratings)
RMSE_reg
```

```
## [1] 0.8651017
```

$$RMSE_{regularization} = 0.8651017$$

RMSE Final Improvment

By cutting of the extreme values we can further improve our estimate. We get the following result:

```
RMSE_limits <- RMSE(validation$rating, reg_predicted_ratings_limit)
RMSE_limits
```

```
## [1] 0.8649177
```

$$RMSE_{limits} = 0.8649177$$

RMSE ML h2o library

Using the Ml algorithm and the h2o library we get the following result :

```
DL_RMSE_validation <- h2o.rmse(DL_model, valid = TRUE)
DL_RMSE_validation
```

```
## [1] 0.8329757
```

$$RMSE_{h2o} = 0.8270073$$

Conclusion

Summary

To conclude, here's a brief recap of our results:

Method	Results
Overall Mean	1.0607143
User and Movie bias	0.8651915
Regularization	0.8651017
Regularization with limits	0.8649177
ML Using h2o library	0.8329757

Our final best RMSE is **0.8329757**, which means that on average the predicted value differs from the real value by 0.8329757.

Further Work

Even though our RMSE score is sufficiently low according to the objective we could pursue different approaches. The Random Forest algorithm could be applied here but due to the size of the dataset its complexity it would require too much computational power.

Another approach would be to use the Tensorflow library but due to the nature of the data this would require modifying our dataset using one-hot encoding. Furthermore, this method is also computationally intensive.