Movie Recommendation System

Ridhiwan Mseya

9/24/2021

INTRODUCTION

In different networks and services users have the ability to rate and share their comments on movies, series or shows that they have watched. The feedback given by the users helps the service provider to know which kinds of movies perform better than others in a given demographic. This allows the service provider to improve their offerings. In this project we will make a movie recommendation system that displays certain kinds of movies to a user depending on how they rated the movies they previously watched or depending on what other similar users have previously watched and rated. This will allow the service provider to order different movies for different users more accurately for the goal of creating satisfied customers.

To achieve this we will use the "MovieLens data set" containing 10 million observations that is available here:https://grouplens.org/datasets/movielens/10m/. The data set was initially split into "edx" and "validation". We will firstly check the structure of the edx data set to see if it has what we need for the modelling of our system. edx will be used for the creation of our model and the validation set will be used for evaluation of our final model.

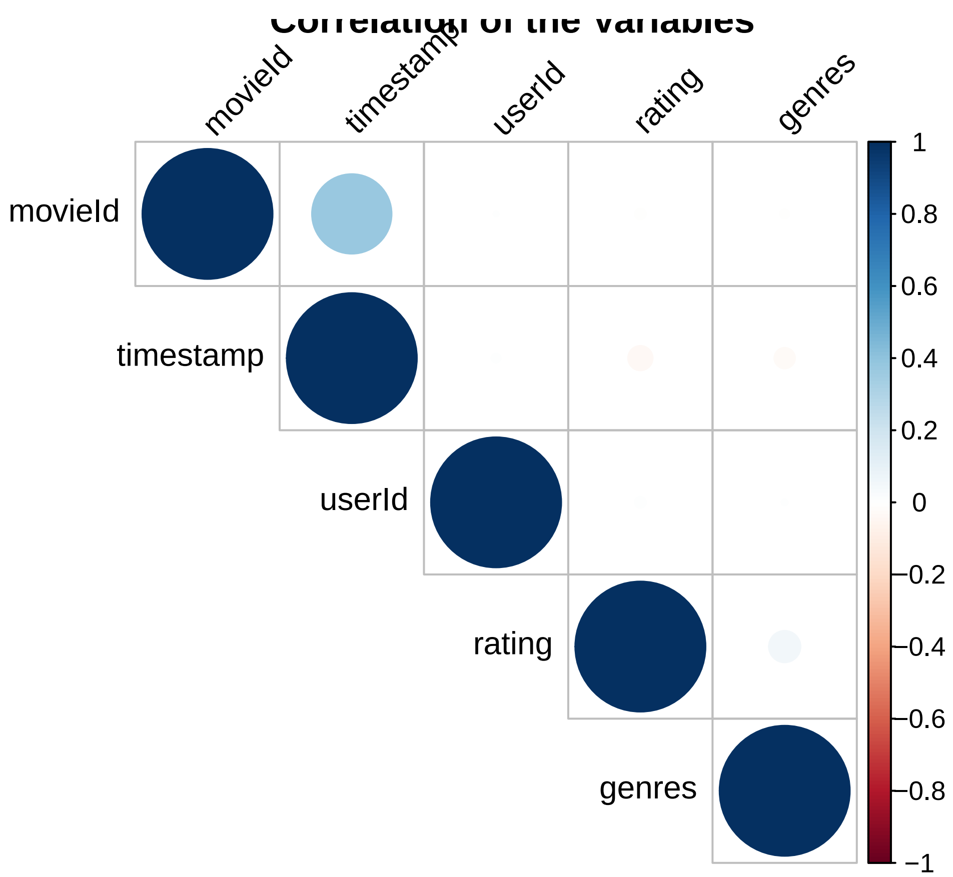
```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 8
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
##  - attr(*, ".internal.selfref")=<externalptr>
```

Our edx data set contains a bit more than 9 million observables the remaining are in the validation set. It contains six variables to which out of the bat, the most important seem to be the "userId", "movieId", "rating" and "genres". The userId is the Id number given to the person watching and rating the movie, movieId is the Id number of the movie being watched and rated, rating is the rating given to the movie ranging from 1 to 5 and genres is the genre of the movie i.e Comedy, Thriller, Action or a combination of them(Action|Thriller).

The goal is to use these variables to train and test the best machine learning model that will give the most accurate predictions. The accuracy of these predictions will be tested using the root mean squared error(RMSE). The model with the lowest rmse is the best model. We will split the edx data set into training and test sets and each time we will create a model and test whether it is performing better than the previous. The models will be different from each other by countering for common problems in machine learning like variability and regularization. Finally we could even combine different models to check if the results are any better.

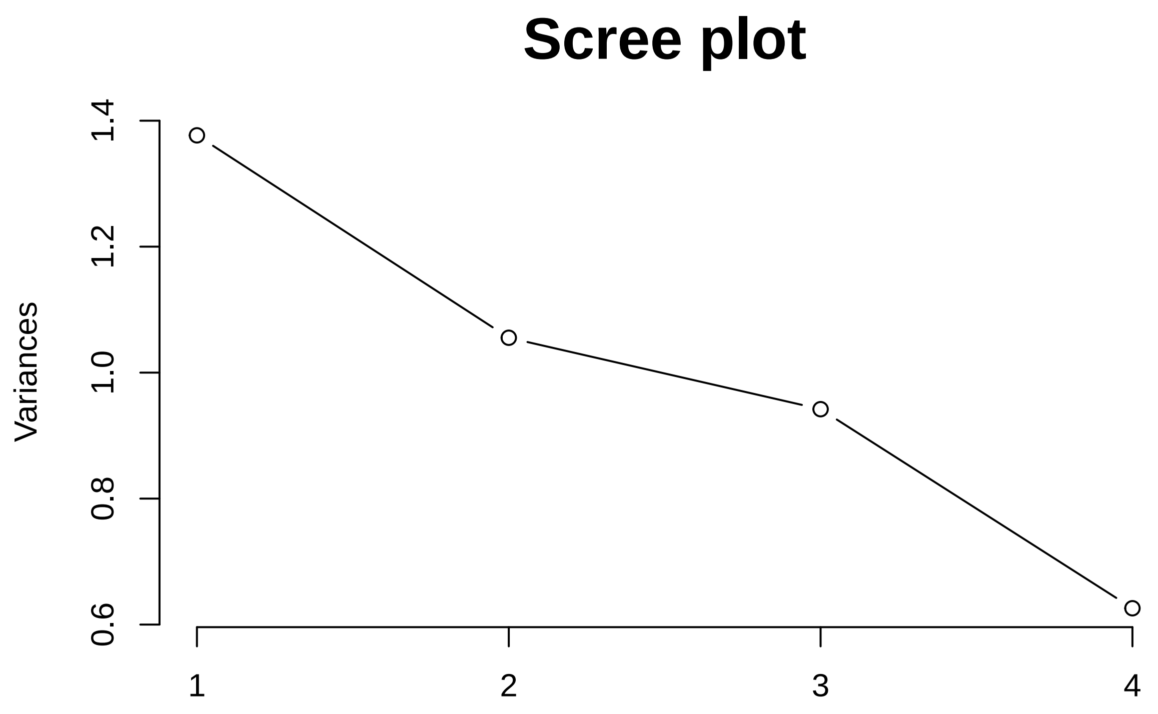
ANALYSIS

We will start with a correlation study on the variables to check if they are any that are too strongly correlated to affect correct predictions from our models. Before we perform the correlation analysis we will first convert some columns with strings to factors and remove columns which are not significant i.e title. We will also use a sample of the data rather than the whole edx data set for most of the analysis that involves exploring the data set.



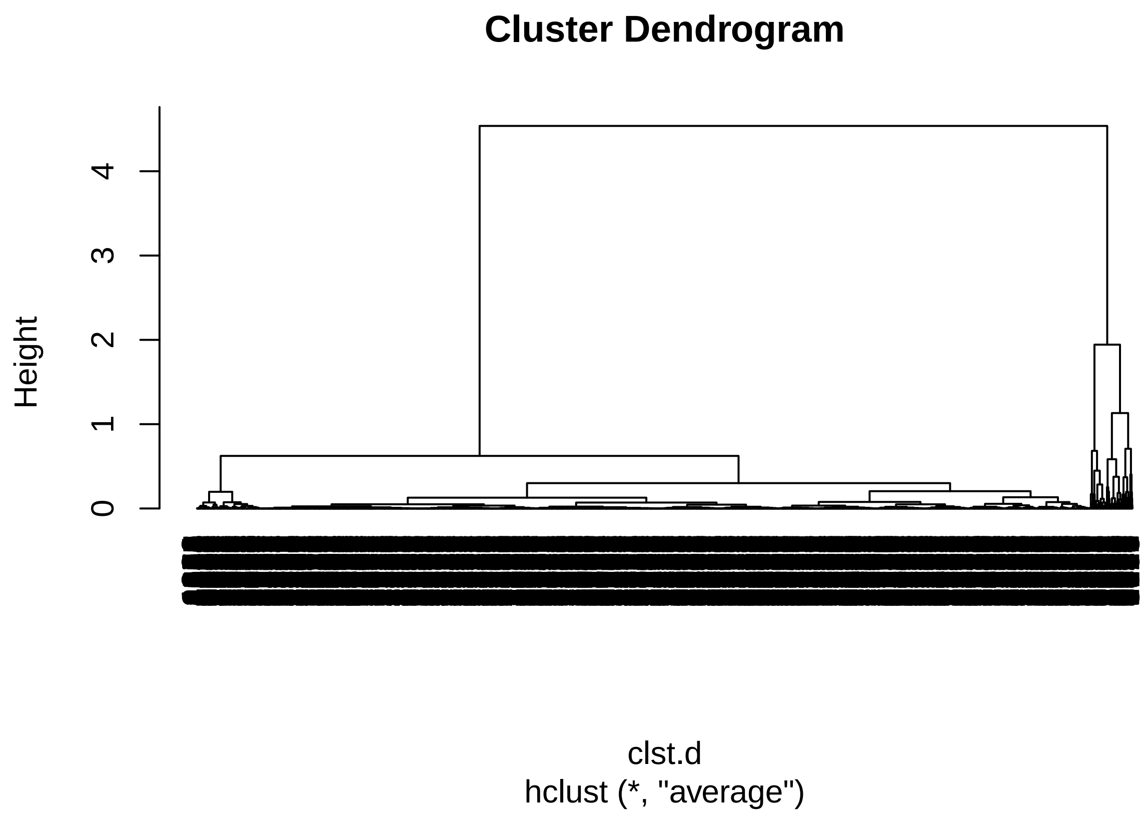
There is no significant correlation between the variables except for timestamp-movieId and rating-genres which have a positive correlation but with low significance.

Next we will check if we can reduce the dimensions of our data and understand it with few predictors. We can check for this using principal component analysis. We will exclude the userId in the analysis and use it to represent the users.



In the scree plot the highest variance is contained in the first three principal components and there is only a total of 4 principal components showing that the data can be reduced to fewer dimensions and based on the userId there could be some clear clustering of our data.

From the above principal component analysis it seems that from the sample of our data set it is possible to cluster our users into different groups.We will try to cluster them using different methods to try and estimate the possible number of different user groups.



The cluster dendrogram shows that clusters exist in our data and we will choose 8 clusters. We will use the silhouette function to observe how good our clustering is when we use 8 clusters.

```
## Silhouette of 5000 units in 8 clusters from silhouette.default(x = (cutree(clst.h, 8)), dist = clst.
##  Cluster sizes and average silhouette widths:
##      4432       341        70        59        21        19        45        13
## 0.6285828 0.8157660 0.5490142 0.4935473 0.7132225 0.8674116 0.5355631 0.6585198
## Individual silhouette widths:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.6754  0.6572  0.7625  0.6391  0.7881  0.8906
```

From the summary the mean silhouette width is not as high as it could be, this signals that some of the clusters may not be very well separated because the number of partitions I have chosen is too high. We will reduce the number to only 2 clusters and rerun our analysis.

```
## Silhouette of 5000 units in 2 clusters from silhouette.default(x = (cutree(clst.h, 2)), dist = clst.
##  Cluster sizes and average silhouette widths:
##      4773       227
## 0.9439258 0.7012582
## Individual silhouette widths:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.2891  0.9447  0.9537  0.9329  0.9599  0.9616
```

With only two clusters we observe a very good separation of the users in our data. The number of clusters could increase with the number of users added to our analysis.

To this point the data set that we are using to create our recommendation system is too big to be analyzed fully given our resources. But a sample from the data set has shown that there is very little to no correlation among the variables in the data set and that it can be clustered. The models that we will use in our recommendation system should involve many differentiating parameters in order to get the best performance.

METHOD

MACHINE LEARNING MODELS

We will start with the simplest model of prediction of ratings of a given movie by given users and move on to better models to try and get the best prediction with our resources. By predicting the rating the user would give to the movie we directly predict the movies that could be highly rated by that user or group of users and can therefore use that prediction to recommend a movie that they will most likely enjoy.

```r
library(caret)
set.seed(33)
test_ind <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
# make a data partition for a test set and train set

train <- edx[-test_ind,] # training set
test <- edx[test_ind,] # testing set

test <- test %>%
     semi_join(train, by = "movieId") %>%
     semi_join(train, by = "userId")
     # get rid of movies or users that are in the test set but not the training set.

rmse <- function(predicted_ratings,ratings){
     sqrt(mean((ratings - predicted_ratings)^2))
}
# a function that calculates the error that we have to minimize to get better models
```

The first model of prediction will involve only one parameter which is the average rating for all the movies combined. We will assume this as the prediction of the rating for any movie and calculate how far from the truth our prediction is, using the “rmse” function.

```r
mu <- mean(train$rating) # average rating for all movies
first_rmse <- rmse(test$rating,mu) # the first prediction's error
first_rmse # print rmse
```

```
## [1] 1.061258
```

The first model has an error that is 1.061258 which is quite high. we will try to reduce this by introducing a second parameter which will take into account the average rating of each individual movie instead of the average of all movies.

```r
avgs <- train %>%
  group_by(movieId) %>%
  summarise(a = mean(rating - mu)) # the average rating for each movie

predicted_ratings <- mu + test %>%
     left_join(avgs, by='movieId') %>%
     .$a # our prediction of the rating of movies in the test set

second_rmse <- rmse(predicted_ratings,test$rating) # the second prediction's error
second_rmse
```

```
## [1] 0.9442879
```

There is a little improvement with our second model with an error of 0.9442879 but we will add another parameter which is specific to users and observe if we can improve our model even further.

```
avgr <- train %>%
  left_join(avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarise(b = mean(rating - mu - a)) # the average rating given by each user

predicted_ratings <- test %>%
  left_join(avgs, by='movieId') %>%
  left_join(avgr, by='userId') %>%
  mutate(pred = mu + a + b) %>%
  .$pred # our prediction of the rating of movies in the test set

third_rmse <- rmse(predicted_ratings,test$rating) # the third prediction's error
third_rmse
```

```
## [1] 0.8669478
```

With the inclusion of the users rating the error is reduced again to 0.8669478 but it is not very good. We will now try to add another parameter to check if it will improve our predictions of the ratings. We will add a parameter for the average rating of a genre.

```
avgt <- train %>%
  left_join(avgs, by = "movieId") %>%
  left_join(avgr, by = "userId") %>%
  group_by(genres) %>%
  summarise(c = mean(rating - mu - a - b)) # the average rating for each genre

predicted_ratings <- test %>%
  left_join(avgs, by='movieId') %>%
  left_join(avgr, by='userId') %>%
  left_join(avgt, by = "genres") %>%
  mutate(pred = mu + a + b + c) %>%
  .$pred # our prediction of the rating of movies in the test set

fourth_rmse <- rmse(predicted_ratings,test$rating) # the fourth prediction's error
fourth_rmse
```

```
## [1] 0.866629
```

There is small improvement from third to fourth model with an error of 0.866629 for the latter. We will add another parameter of the average rating by the time at which the rating was done. To do this we will have to add another column to our train and test sets.

```
library(lubridate)
train <- train %>%
  mutate(timestamp_t=format(as_datetime(train$timestamp), "%Y-%m-%d"))
# add a new column to the training set with new date format
test <- test %>%
  mutate(timestamp_t=format(as_datetime(test$timestamp), "%Y-%m-%d"))
# add a new column to the test set with new date format
```

After the modification we can now work on our fifth model using these sets of data.

```
avgu <- train %>%
  left_join(avgs, by = "movieId") %>%
  left_join(avgr, by = "userId") %>%
```

```r
  left_join(avgt, by = "genres") %>%
  group_by(timestamp_t) %>%
  summarise(d = mean(rating - mu - a - b - c))
  # the average rating for each timestamp

predicted_ratings <- test %>%
  left_join(avgs, by='movieId') %>%
  left_join(avgr, by='userId') %>%
  left_join(avgt, by = "genres") %>%
  left_join(avgu, by = "timestamp_t") %>%
  mutate(pred = mu + a + b + c + d) %>%
  .$pred # our prediction of the rating of movies in the test set

fifth_rmse <- rmse(predicted_ratings,test$rating) # the fifth prediction's error
fifth_rmse
```

```
## [1] 0.8661281
```

The improvement is much better after the fifth model having an error of 0.8661281 but we have to take a different approach to make the results a bit better. We will try regularization on all the four parameters that we added to the simplest model in order to improve its performance.

```r
pens <- seq(0, 10, 0.25) # the range of penalty constants to choose from
RMSES <- sapply(pens, function(p){
     a_r <- train %>%
          group_by(movieId) %>%
          summarise(a_r = sum(rating - mu)/(n()+p))
     # regularization of the parameter
     b_r <- train %>%
          left_join(a_r, by="movieId") %>%
          group_by(userId) %>%
          summarise(b_r = sum(rating - a_r - mu)/(n()+p))
     # regularization of the parameter
     c_r <- train %>%
          left_join(a_r, by="movieId") %>%
          left_join(b_r, by="userId") %>%
          group_by(genres) %>%
          summarise(c_r = sum(rating - a_r - b_r - mu)/(n()+p))
     # regularization of the parameter
     d_r <- train %>%
          left_join(a_r, by="movieId") %>%
          left_join(b_r, by="userId") %>%
          left_join(c_r, by="genres") %>%
          group_by(timestamp_t) %>%
          summarise(d_r = sum(rating - a_r - b_r - c_r - mu)/(n()+p))
     # regularization of the parameter
     predicted_ratings <-
          test %>%
          left_join(a_r, by = "movieId") %>%
          left_join(b_r, by = "userId") %>%
          left_join(c_r, by = "genres") %>%
          left_join(d_r, by = "timestamp_t") %>%
          mutate(pred = mu + a_r + b_r + c_r + d_r) %>%
          .$pred # making the prediction of ratings
     return(rmse(predicted_ratings, test$rating))
}) # A function that returns rmses at different penalties

pen <- pens[which.min(RMSES)] # look for the penalty constant which gives the minimum error
pen # print the penalty constant
```

The penalty constant that gives the lowest error is 5.25, now we will check to see what is the value of the lowest error.

```
sixth_model <- function(p){
     a_r <- train %>%
          group_by(movieId) %>%
          summarise(a_r = sum(rating - mu)/(n()+p))
     b_r <- train %>%
          left_join(a_r, by="movieId") %>%
          group_by(userId) %>%
          summarise(b_r = sum(rating - a_r - mu)/(n()+p))
     c_r <- train %>%
          left_join(a_r, by="movieId") %>%
          left_join(b_r, by="userId") %>%
          group_by(genres) %>%
          summarise(c_r = sum(rating - a_r - b_r - mu)/(n()+p))
     d_r <- train %>%
          left_join(a_r, by="movieId") %>%
          left_join(b_r, by="userId") %>%
          left_join(c_r, by="genres") %>%
          group_by(timestamp_t) %>%
          summarise(d_r = sum(rating - a_r - b_r - c_r - mu)/(n()+p))
     predicted_ratings <-
          test %>%
          left_join(a_r, by = "movieId") %>%
          left_join(b_r, by = "userId") %>%
          left_join(c_r, by = "genres") %>%
          left_join(d_r, by = "timestamp_t") %>%
          mutate(pred = mu + a_r + b_r + c_r + d_r) %>%
          .$pred
     return(rmse(predicted_ratings, test$rating))
} # A function that returns the lowest error
sixth_model(5.25) # calling the function to calculate the error from the sixth model
```

Now we have a much better improvement to the error in our sixth model, reducing it from 1.061258 in the first model to 0.8654189 in the sixth model.

RESULTS

We will prepare our validation set to check the actual error of our model.

```
library(tidyverse)
library(lubridate)
load("validation")
validation_edit <- validation %>%
     semi_join(train, by = "movieId") %>%
  semi_join(train, by = "userId")
# get rid of movies or users that are in the validation set but not the training set.
validation_edit <- validation_edit %>% mutate(timestamp_t=format(as_datetime(validation_edit$timestamp)
# add a new column to the validation set with new date format
```

```r
sixth_model <- function(p){
     a_r <- train %>%
          group_by(movieId) %>%
          summarise(a_r = sum(rating - mu)/(n()+p))
     b_r <- train %>%
          left_join(a_r, by="movieId") %>%
          group_by(userId) %>%
          summarise(b_r = sum(rating - a_r - mu)/(n()+p))
     c_r <- train %>%
          left_join(a_r, by="movieId") %>%
          left_join(b_r, by="userId") %>%
          group_by(genres) %>%
          summarise(c_r = sum(rating - a_r - b_r - mu)/(n()+p))
     d_r <- train %>%
          left_join(a_r, by="movieId") %>%
          left_join(b_r, by="userId") %>%
          left_join(c_r, by="genres") %>%
          group_by(timestamp_t) %>%
          summarise(d_r = sum(rating - a_r - b_r - c_r - mu)/(n()+p))
     predicted_ratings <-
          validation_edit %>%
          left_join(a_r, by = "movieId") %>%
          left_join(b_r, by = "userId") %>%
          left_join(c_r, by = "genres") %>%
          left_join(d_r, by = "timestamp_t") %>%
          mutate(pred = mu + a_r + b_r + c_r + d_r) %>%
          .$pred
     return(rmse(predicted_ratings, validation_edit$rating))
} # A function that returns the lowest error
sixth_model(5.25) # calling the function to calculate the error from the sixth model
```

```
## [1] 0.8647672
```

From the validation set we obtain an error of 0.8647672 from our sixth and final model. This result is much better and shows that our model is not the best but it is good enough to be usable in predictions of ratings and in movie recommendation systems. The first model was clearly never going to work because it just predicted an average rating for all movies in the data but it was a good starting point to guide us into the following models. The second model added a single parameter by checking the variation in the average rating of each movie in the dataset this increased our accuracy a little bit. The following parameter in the third model was the variation in the ratings given by users that also increased our accuracy further. The fourth model added the variation in the ratings of each genre but the increase in accuracy was not as we expected. The fifth model looked into the variation in the time that the rating was made which usually coincides with its release. Movies released in the summer and holiday seasons tend to have different ratings from those rated in other months of the year. All of these models in accumulation increased our accuracy of prediction significantly but they were all overlooking one key factor in the structure of the data. There is an effect of size in the variability of our data thus with regularization we can penalize large estimates arising from these effects. So lastly we use regularization to estimate all our five parameters to make the sixth and final model for our prediction.

After the final model calculations we get an error of 0.8647672 which is big improvement from the 1.061258 error which we obtained initially.

CONCLUSION

The machine learning models that we used proved to be useful in the automation of rating prediction of the movies that they were provided from the validation set. They are several short comings to our prediction model; we are overfitting our data as our validation set is not an original data set but a fraction of the data set that we used to train and test our models, our model for machine learning is only based on average variations and our machine learning model could be improved by using more advanced models available in the literature. For future applications a computer with higher specifications that can handle much larger workloads should be used to avoid long wait times for calculations to finish processing. Better machine learning models and even deep learning models might work wonders in prediction of ratings.

To conclude the bottle necks faced in the project are not substantial and can be easily alleviated with a few changes in the computational resources used and the learning models.