Capstone Project 2: Choose your own Project

Majid Sedighi

10/16/2020

1- Introduction

The goal of the study and the dataset choosen is described in this section.

The dataset is taken from the ML-friendly Public Datasets list on Kaggle, as suggested by the assingment. it is called "Video_Games_Sales_as_at_22_Dec_2016". This dataset contains a list of video games with sales greater than 100,000 copies. It was generated by a scrape of vgchartz.com. The main focus of the dataset is the sales of each video game, and it is counted in North America, Japan, Europe, and the rest of the world separately.

The task and the goal of this project is as follows:

Supposing the game is already released in North America and Japan, the goal is to develop an algorithm that recommends the publisher if it is worth it to release the game in Europe or not.

The metric for a successful release in Europe is defined as acheiving a ratio of sales compared to North America and Japan which is higher than median across all games. This means that the sales in Europe divided by the sales in the rest of the world is the metric, and the criteria for success is this value being higher than median across all games.

The Fields included in this datset are:

- Rank - Ranking of overall sales
- Name - The games name
- Platform - Platform of the games release (i.e. PC,PS4, etc.)
- Year - Year of the game's release
- Genre - Genre of the game
- Publisher - Publisher of the game
- NA_Sales - Sales in North America (in millions)
- EU_Sales - Sales in Europe (in millions)
- JP_Sales - Sales in Japan (in millions)
- Other_Sales - Sales in the rest of the world (in millions)
- Global_Sales - Total worldwide sales.
- Critic_score - Aggregate score compiled by Metacritic staff
- Criticcount - The number of critics used in coming up with the Criticscore

- User_score - Score by Metacritic's subscribers
- Usercount - Number of users who gave the userscore
- Developer - Party responsible for creating the game
- Rating - The ESRB ratings

The two methods chosen for this analysis is a classification decision tree and random forests. In the following sections the algorithms are described in detail and the results of these two models will be compared.

2- Analysis

In this section the dataset is studied and explored and then the algorithm is described in detail.

The dataset while originally from Kaggle, is uploaded on the Github repository, and is accessible via this code:

```
url <- "https://raw.githubusercontent.com/M-Sedighi/Harvardx-DataScience-Capstone-2/main/Video_Games_Sa
Video_Games_Sales_as_at_22_Dec_2016 <- read_csv(url)
sales <- Video_Games_Sales_as_at_22_Dec_2016
```

The dataset has the following structure

```
## [1] "dimensions of dataset"
```

```
## [1] 16719    16
```

```
## # A tibble: 6 x 16
##   Name  Platform Year_of_Release Genre Publisher NA_Sales EU_Sales JP_Sales
##   <chr> <chr>    <chr>           <chr> <chr>        <dbl>    <dbl>    <dbl>
## 1 Wii ~ Wii      2006            Spor~ Nintendo      41.4    29.0      3.77
## 2 Supe~ NES      1985            Plat~ Nintendo      29.1     3.58     6.81
## 3 Mari~ Wii      2008            Raci~ Nintendo      15.7    12.8      3.79
## 4 Wii ~ Wii      2009            Spor~ Nintendo      15.6    10.9      3.28
## 5 Poke~ GB       1996            Role~ Nintendo      11.3     8.89    10.2
## 6 Tetr~ GB       1989            Puzz~ Nintendo      23.2     2.26     4.22
## # ... with 8 more variables: Other_Sales <dbl>, Global_Sales <dbl>,
## #   Critic_Score <dbl>, Critic_Count <dbl>, User_Score <chr>, User_Count <dbl>,
## #   Developer <chr>, Rating <chr>
```

There are some steps necessary to prepare the data for our anlysis, which are described in the following.

First of all, by looking at the Ratings column, we realise that the values here are the age rating for the games based on the ESRB determinations, and these abbreviations have meanings as follows:

```
##   Abbreviation          rating
## 1           RP  Rating Pending
## 2           EC Early Childhood
## 3            E        Everyone
## 4         E10+    Everyone 10+
## 5            T            Teen
## 6            M          Mature
## 7           AO     Adults Only
```

As can be seen, these values cover age ranges, and a case could be made for them being ordinal, since for example adults only covers higher values for age than teen. For this purpose, we factorize these values and consider them ordinal. Also in order to handle NAs, we consider them the same as the Rating Pending category. This makes sense since a game that has not been rated yet and a game that we do not know its rating are practically the same for us. Another point to consider is the label "K-A". This rating was known as Kids to Adults (K-A) until 1998, when it was renamed "Everyone". Therefore we change this value to "E" for our dataset. These steps are carried out in the code below:

```
ESRB <- sales$Rating
ESRB[ESRB == "K-A"] <- "E"
ESRB[is.na(ESRB)] <- "RP"
lvls = c("RP", "EC", "E", "E10+", "T", "M", "AO")
lbls = c("Rating Pending", "Early Childhood", "Everyone",
         "Everyone 10+", "Teen", "Mature", "Adults Only")

ESRB <- factor(ESRB, levels = lvls, labels = lbls,
       ordered = TRUE)
```

Another feature in this dataset is the Genre. It is definitely is a useful predictor for sales, and since it has a value "misc" in our dataset, we resort to setting the NAs to the same as "misc", since they could be of any genre.

```
sales$Genre[is.na(sales$Genre)] <- "Misc"
```

Platform, Year of release, and publisher are also useful predictors for sales, since they are play a role in how well a game sells. However, developer is a categorical variable with many different values, since there are many different studios developing games, but way fewer publishers. Also this feature has a lot of NA values that we do not know how to handle, therefore it will be ignored. Same goes for the critic and user scores. They could be possibly useful, but since they have large portions of NAs, they won't be useful in our analysis. Decision trees are not robust to NAs; on the other hand, they are a suitable technique when dealing with combinations of categorical and numerical predictors.

Based on the considerations described above, we modify our sales dataset as below:

```
sales <- sales %>% mutate(Publisher = factor(Publisher), Platform = factor(Platform),
                          Genre = factor(Genre), Rating = ESRB)
```

As for our metric for a successful Euro release, we define a boolean that shows true when the release is predicted to be successful and False when not. The median ratio of Euro sales and the sales in the rest of the world is calculated, and any games that sells higher than that is considered successful. The ratio is calculated here:

```
sales <- sales %>% mutate(EU_factor = EU_Sales/(NA_Sales+JP_Sales+Other_Sales))
```

We might run into an issue, which is if a game is only marketed in Europe and has no sales in the rest of the world, our ration will be deviding to zero. this will be problematic for the rest of our calculation, so for calculating the median, we will not consider them. Once we have the median, we can calculate our EU release logical variable by comparing the ratios for each element to the median:

```
sum(!is.finite(sales$EU_factor))
```

```
## [1] 601
```

```
EU_scf <- median(sales$EU_factor[is.finite(sales$EU_factor)])
sales <- sales %>% mutate(EU_release = EU_factor > EU_scf)
```

The median value is 0.2222222, so any game with Europe to rest of the world ratio higher than this value is advisable for a Europe release. We also face some NA values here, which is the result of elements that have zero as all of their sale data. Since they did not sell any copies, it is logical to advice against their release in Europe, therefore they get a FALSE for their release value.

```
which(is.na(sales$EU_release))
```

```
## [1] 16681 16714
```

```
sales$EU_release[is.na(sales$EU_release)] <- FALSE
table(sales$EU_release)
```

```
##
## FALSE  TRUE
##  8116  8603
```

```
sales <- sales %>% mutate(EU_release= factor(EU_release))
```

Now that we have the dataset preprocessed and ready for our analysis, it is time to split the data into training and test set. We do it randomly and allocate 10% of the data to the test set and rest to the training set.

```
set.seed(2, sample.kind="Rounding")
test_index <- createDataPartition(y = sales$EU_release, times = 1, p = 0.1, list = FALSE)
train_set <- sales[-test_index,]
test_set <- sales[test_index,]
```

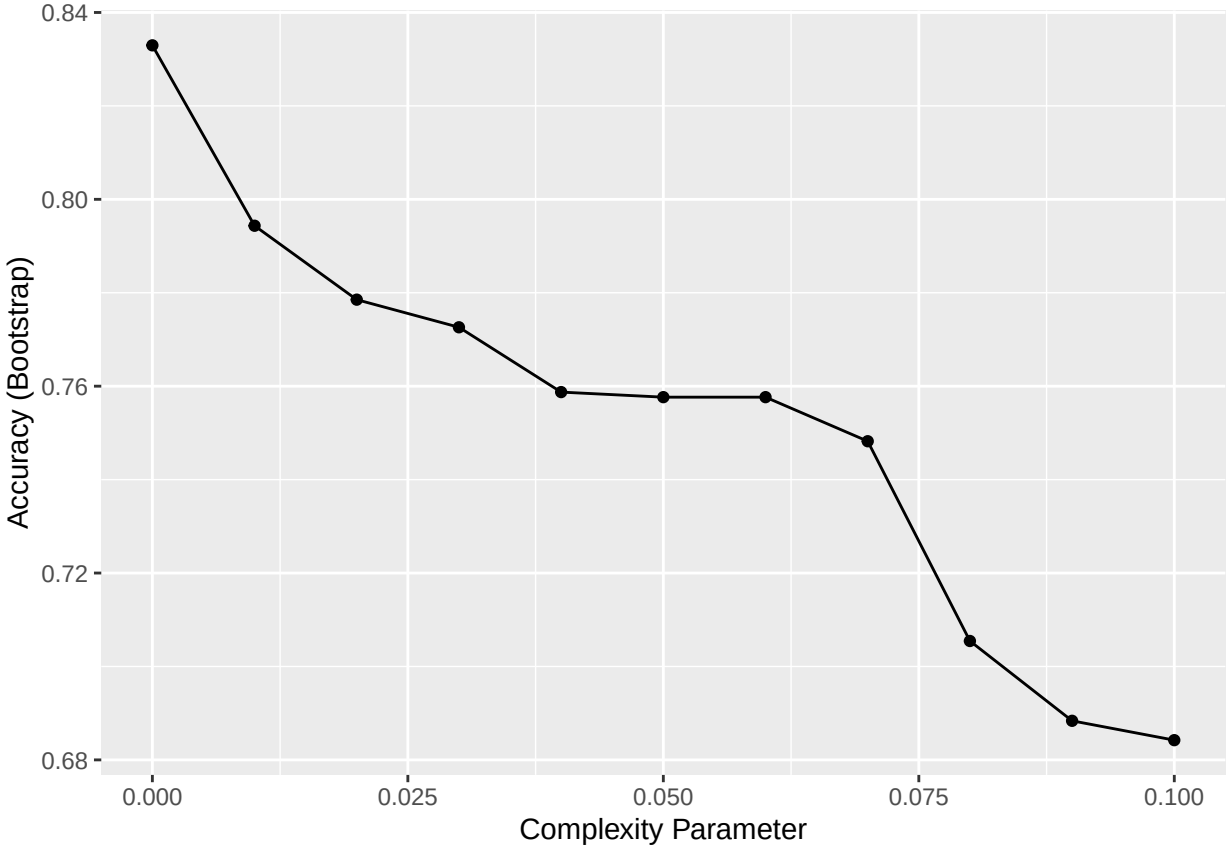
In this analysis, we will use two methods to solve our binary classification problem. First we use a single decision tree, and find the best model by tuning the *cp* value. We will create the confusion matrix and find our accuracy values. The we will try to improve this accuracy using the random forests method, from the Rborist package. Random forests genrally involves many more calculations and therefore it is time consuming, so the goal here is to confirm if it is worth using over a single decision tree for this particular dataset.

3- Results

We will start solving the decision tree problem while tuning the parameter *cp* in the code below. At this stage we pick a wide range for *cp* to see which values result in higher training accuracy.

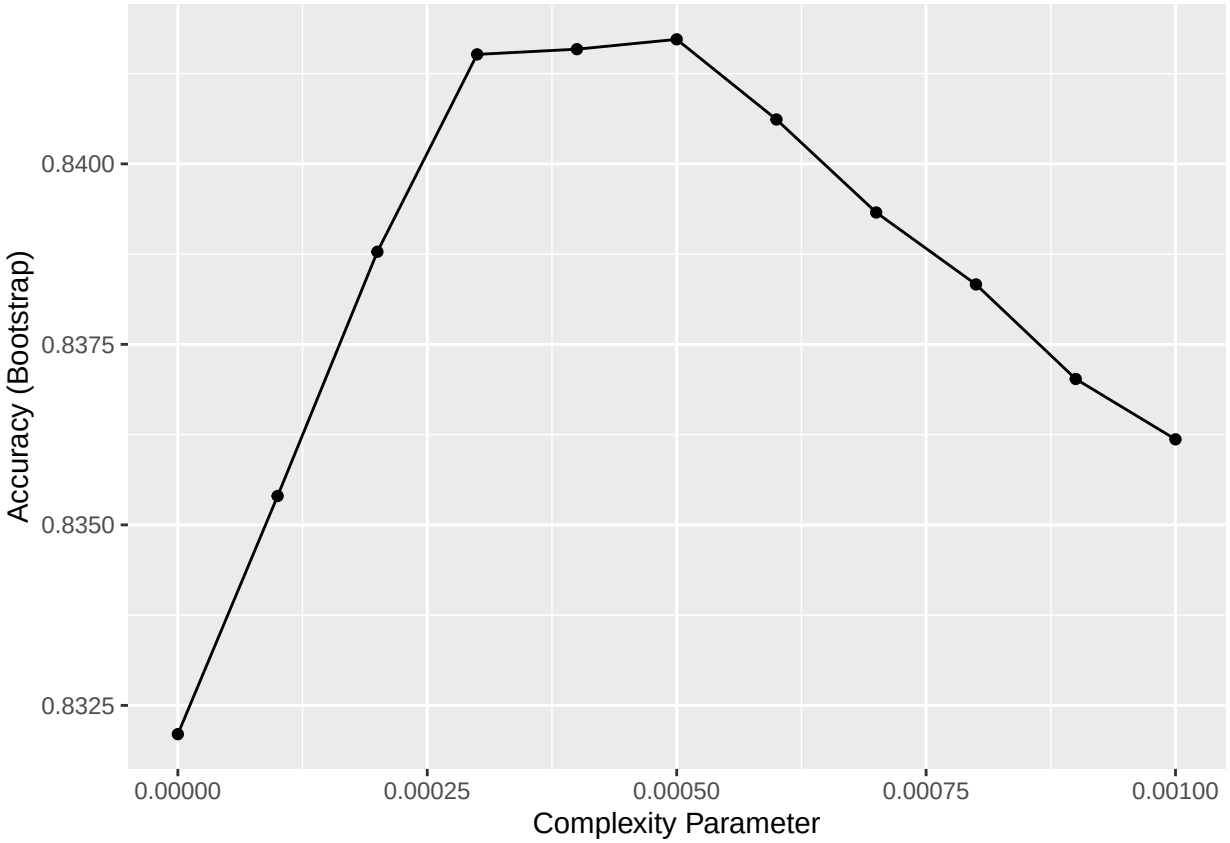
```
train_rpartc <- train(EU_release ~ Publisher+Genre+Platform+Year_of_Release+Rating+NA_Sales+JP_Sales,
                      tuneGrid = data.frame(cp = seq(0, 0.1, len = 11)), method = "rpart", data = train_

ggplot(train_rpartc)
```



as can be seen in the result of the training, the training accuracy is highest at the smallest values, therefore we repeat the training with the smaller range of *cp* closer to 0.

```
train_rpart <- train(EU_release ~ Publisher+Genre+Platform+Year_of_Release+Rating+NA_Sales+JP_Sales,
                     tuneGrid = data.frame(cp = seq(0, 0.001, len = 11)), method = "rpart", data = trai
ggplot(train_rpart)
```



This range for *cp* shows generally higher values for accuracy on the training set. We will keep this model for later comparison to the Random forests method results. For random forests, we choose the Rborist pacakge, and run our training with the default values. (this training takes around 20 minutes)

```
train_rf <- train(EU_release ~ Publisher+Genre+Platform+Year_of_Release+Rating+NA_Sales+JP_Sales,
                  method = 'Rborist',
                  data = train_set)
```

based on these two models, we can now predict the outputs for our test set and creat confusion matrix to compare the accuracy of the two models.

```
y_hat <- predict(train_rpart, test_set)
y <- data.frame(Y=test_set$EU_release,Y_hat=y_hat)
confusionMatrix(y$Y,y$Y_hat)$overall["Accuracy"]
```

```
##  Accuracy
## 0.8607292
```

```
confusionMatrix(y$Y,y$Y_hat)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
```

```
##      FALSE   680  132
##      TRUE    101  760
##
##                Accuracy : 0.8607
##                  95% CI : (0.8432, 0.877)
##     No Information Rate : 0.5332
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.7209
##
##  Mcnemar's Test P-Value : 0.04937
##
##             Sensitivity : 0.8707
##             Specificity : 0.8520
##          Pos Pred Value : 0.8374
##          Neg Pred Value : 0.8827
##              Prevalence : 0.4668
##          Detection Rate : 0.4065
##    Detection Prevalence : 0.4854
##       Balanced Accuracy : 0.8613
##
##        'Positive' Class : FALSE
##
```

```
y_hat_2 <- predict(train_rf, test_set)
y_2 <- data.frame(Y=test_set$EU_release,Y_hat=y_hat_2)
confusionMatrix(y_2$Y,y_2$Y_hat)$overall["Accuracy"]
```

```
##  Accuracy
## 0.8469815
```

```
confusionMatrix(y_2$Y,y_2$Y_hat)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE   612  200
##      TRUE     56  805
##
##                Accuracy : 0.847
##                  95% CI : (0.8288, 0.8639)
##     No Information Rate : 0.6007
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6921
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9162
##             Specificity : 0.8010
##          Pos Pred Value : 0.7537
##          Neg Pred Value : 0.9350
```

```
##              Prevalence : 0.3993
##          Detection Rate : 0.3658
##    Detection Prevalence : 0.4854
##       Balanced Accuracy : 0.8586
##
##        'Positive' Class : FALSE
##
```

To our surprise, we can see here that a single decision tree produces a higher accuracy than the Rborist random forest with the default values. Considering the amount of training time that is saved by choosing a single decision tree in comparison to random forest, the use of random forests for this particular problem seems to be redundant. The accuracy we achieve with a single decision tree for this test set 0.8607292 is high enough to justify the use of this method.

4- Conclusion

The goal of the project was to develop a method that can predict the success of a video game release in europe based on the characteistics of the game and its sales in the rest of the world. This goal was achieved with acceptable accuracy using the decision tree algorithm. It should be noted that in real life, the criteria of a release success is widely different from what we used here, and additionally, the decision for a release in a continenet often has to be made before any release and there would be no sale data available for the game.

On choosing the classification method, it is worth noting that a major advantage of decision trees is their ability to combine numerical and categorical predictors. This is a feature that makes this method more efficient with such datasets compared to other classification methods. It is however possible to modify the data in order to use other methods such as knn. This will be achieved through one-hot encoding the catororical variables into sparse matrices.

Further examination can be made on why the single decision tree performed better than the Rborist model. Perhaps by tuning the hyperparameters of the random forest method a higher accuracy can be achieved. However since the accuracy of a single decision tree is sufficient for the purposes of this study, further investigation on the effect of hyperparameters for random forest is not included here.