# Final Project

Ricardo Morales

2024-06-12

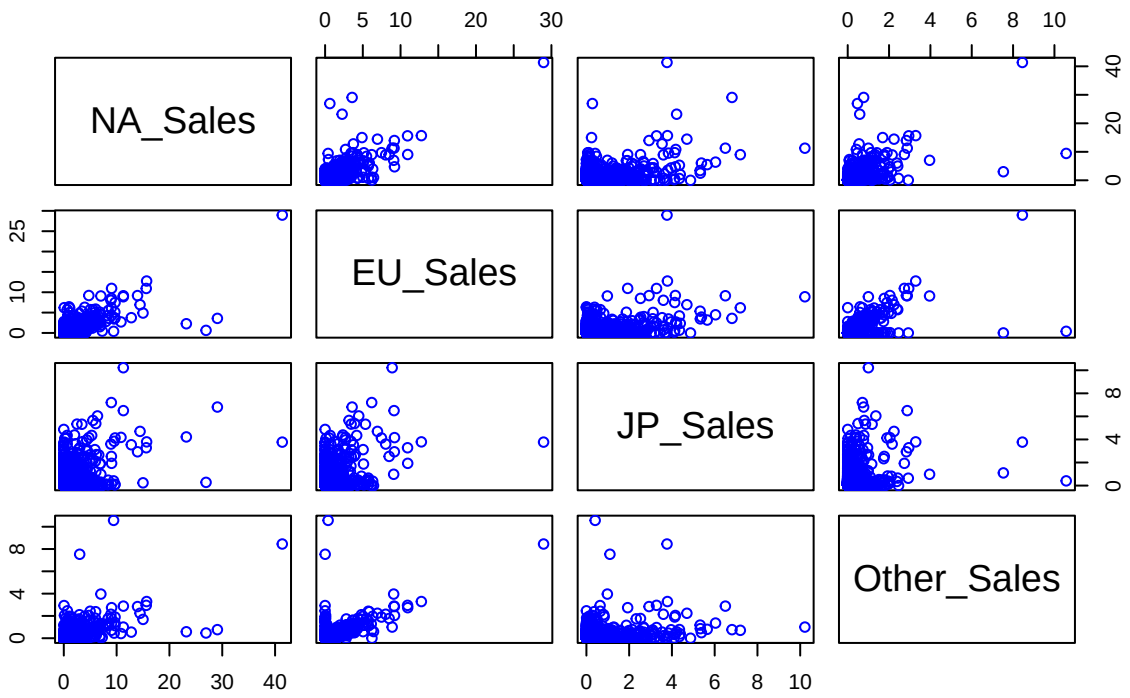
## 1. Introduction

We will use the **Video Game Sales with Ratings** data set, which is available in the ***Kaggle*** website. As its name indicates, this data set shows video game sales (in millions) for different markets (such as USA, Europe, Japan and others) and their respective scores (given by both critics and users), among other information associated to those video games. Specifically, there are **16** variables and **16,719** rows in the original data set.
Our goal is to design algorithms that can help us predict the scores given by the *Metacritic* (an specialized video games site) staff, which are stored in the variable **Critic_Score** based on the information included in the data set. We expect to develop algorithms with high accuracy, considering that scores assigned by critics can be related to video game sales and also by ratings provided by users (these are stored in the variable **User_Score**). We could think, for example, that a video game with high sales must be of good quality and, therefore, must receive a relatively good score. Besides, we can assume that, in general, users and critics tend to assess video games in a similar way.
We will use four techniques in order to predict the scores by the critics based on the predictor selected: *k-nearest neighbors*, *classification tree*, *random forest*, and *ensembles*. We will estimate the models, according to what we learnt throughout the course, to then compare their corresponding accuracy measures.

## 2. Analysis

We download the data set from the ***Kaggle*** website and stored it under the **vg** object. Then, we start by making some data cleaning. In that regard, we detect the presence of some missing values (**NAs**) in variables such as *Critic_Score*, *Critic_Count*, *User_Score* and *User_Count*. We also notice that, unlike *Critic_Score*, the variable type of *User_score* is *character*. Besides, we can see some extreme values for the *sales* variables (very high video game sales amounts in some markets).

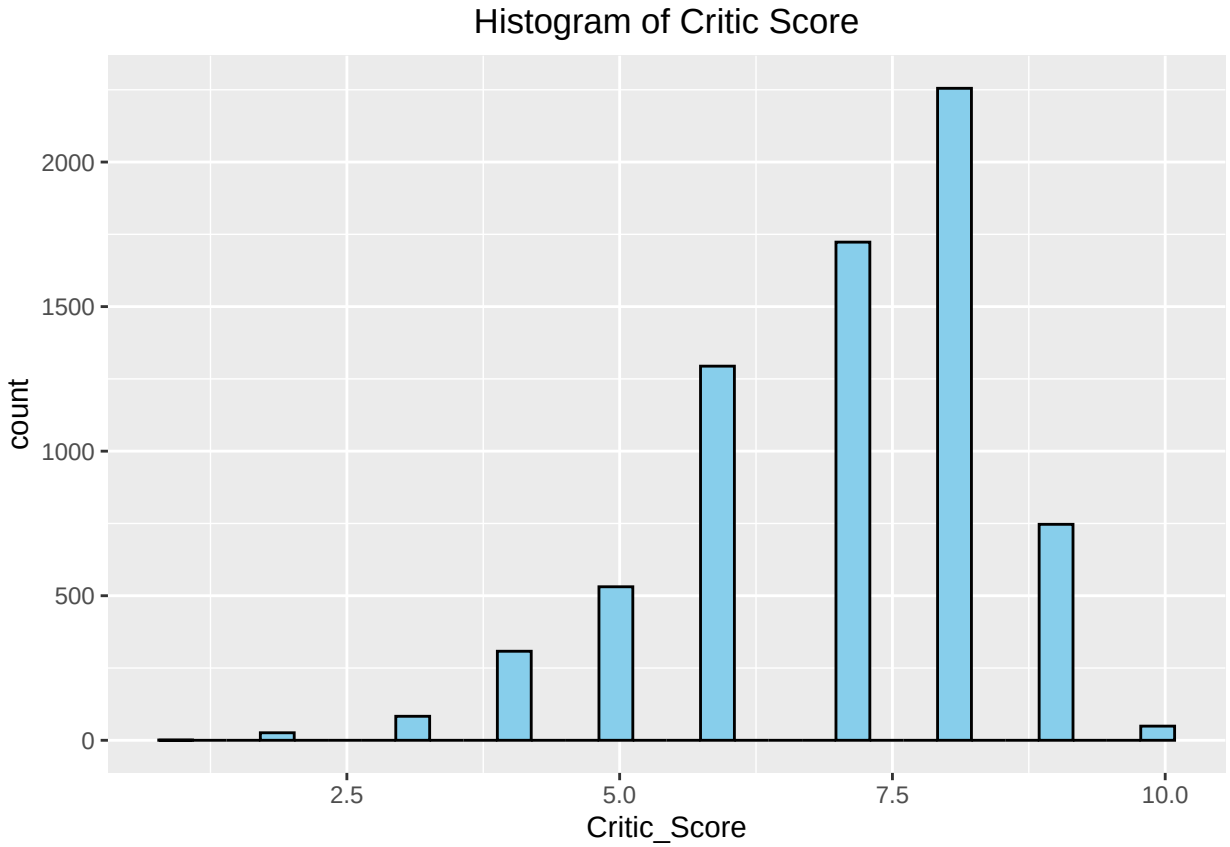
We observe that the sales amounts in USA and Europe are considerably higher than those registered in other markets. Therefore, we can consider these two variables (*NA_Sales* and *EU_Sales*) as representative of video game sales.




Furthermore, we standardized the scales of both critics and user scores, such that they are expressed in a scale from 0 to 10, in integers. We correct all these issues using the following piece of code. Notice we end up selecting four variables: *NA_Sales*, *EU_Sales*, *User_Score* and *Critic_Score*, where the first four will be employed as predictors of the latter.

```
vg <- vg %>% filter(!is.na(Critic_Score), !is.na(Critic_Count),
                    !is.na(User_Count)) %>%
  mutate(Critic_Score = round(Critic_Score/10),
         User_Score = round(as.numeric(User_Score))) %>%
  select(NA_Sales, EU_Sales, User_Score, Critic_Score)
```

The histogram for the variable *Critic_Scores* is shown next. We can see it is relatively skewed to the left. As we mentioned, both are expressed on a scale from 0 to 10. Since it would be difficult to expect for our algorithms to predict accurately a variable with 10 classes (considering that the *vg* data set is not that large in relative terms), we turn them into *binary* variables (with values of 0 or 1).




```
vg<- vg %>% mutate(Critic_Score = ifelse(Critic_Score<=5,0,1),
                   User_Score = ifelse(User_Score<=5,0,1))
```

Then, we proceed to split the *vg* data set into a *train_set* and a *test_set* (where the latter will be of a size equal to 10% of the former). These data sets will include four variables each. We do this in the following manner:

```
set.seed(1, sample.kind="Rounding")
index <- sample(nrow(vg), 0.1*nrow(vg))
train_set <- vg[-index,]
test_set <- vg[index,]
train_set$Critic_Score <- factor(train_set$Critic_Score)
train_set$User_Score <- factor(train_set$User_Score)
test_set$Critic_Score <- factor(test_set$Critic_Score,
                                levels = levels(train_set$Critic_Score))
test_set$User_Score <- factor(test_set$User_Score,
                              levels = levels(train_set$User_Score))
```

Next, we start estimating our models to predict *Critic_Score* using the tools provided by the *caret* library. We begin with the **k-Nearest Neighbors** algorithm (method = *knn* in the *train* function). In order to select a proper value for the parameter $k$, we employ *cross validation*, using the set of specifications recommended in the course (number = 10 and p = 0.9). We try different values for $k$, from 3 to 50. The value for $k$ that allows us to get the best *tune* is *29*. Notice that, in doing all this, we have used the *train_set* only.

```
set.seed(1, sample.kind="Rounding")
train_knn <- train(Critic_Score ~ .,
                   method = "knn",
                   data = train_set,
                   tuneGrid = data.frame(k = seq(3,50,2)),
                   trControl = trainControl(method = "cv",
                                            number = 10, p = 0.9))
train_knn$bestTune
```

```
##     k
## 14 29
```

Then, based on the specifications just determined, we run the *knn* model but this time we apply it on the *test_set*. Finally, we calculated the model overall accuracy by comparing the scores predicted by the algorithm to those that are part of the *test_set*, obtaining an accuracy measure of **0.8973**.

```
knn_Critic_Score <- predict(train_knn, test_set)
mean(knn_Critic_Score == test_set$Critic_Score)
```

```
## [1] 0.8972896
```

Then, we estimate a **classification tree** model (method = *rpart* in the *train* function) to predict the scores of interest, always based on the *train_set* (at least in this first stage). We try different values for the complex parameter *cp* (from 0 to 0.05), obtaining that the best fit happens when *cp* equals 0.008.

```
set.seed(10, sample.kind="Rounding")
train_rpart <- train(Critic_Score ~ .,
                     data = train_set,
                     method = "rpart",
                     tuneGrid = data.frame(cp = seq(0, 0.05, 0.002)))
train_rpart$bestTune
```

```
##      cp
## 5 0.008
```

Hence, based on the best *classification tree* model, we predict the scores of interest, which we then compare to the *Critic_Score* values in *test_set* in order to calculate the prediction accuracy. Thus, we obtain an accuracy measure of **0.9016**.

```
rpart_Critic_Score <- predict(train_rpart, test_set)
mean(rpart_Critic_Score == test_set$Critic_Score)
```

```
## [1] 0.9015692
```

Next, we estimate a **random forest** model (method = *rf* in the *train* function) to predict the critic scores, based initially on the *train_set*. Again, we take as reference the specifications tried when we covered this topic in the course (ntree = 100 and values for *mtry* from 1 to 10).

```
set.seed(11, sample.kind="Rounding")
train_rf <- train(Critic_Score ~ .,
                  data = train_set,
                  method = "rf",
                  ntree = 100,
                  tuneGrid = data.frame(mtry = seq(1:10)))
train_rf$bestTune
```

```
##   mtry
## 1    1
```

We obtain a value of 1 for *mtry*. Then, based on the *random forest* model just estimated, we predict the critic scores to compare them to the *Critic_Score* values of test_set in order to calculate the overall accuracy. Thus, the accuracy measure is **0.8973**.

```
rf_Critic_Score <- predict(train_rf, test_set)
mean(rf_Critic_Score == test_set$Critic_Score)
```

```
## [1] 0.8972896
```

Finally, we construct a simple algorithm known as **ensemble**, which relies on the results obtained for the previous three models: *knn*, *classification trees* and *random forest*. We create a data frame with the predictions generated from each of these three algorithms and determine a new vector of predictions based on simply counting which observation is the most common (either 0 or 1). We do this by running the next piece of code in R:

```
ensemble <- data.frame(knn = knn_Critic_Score,
           rpart = rpart_Critic_Score,
           rf = rf_Critic_Score)
ensemble_Critic_Score <- ifelse(rowSums(ensemble == 1) > 1, 1, 0)
mean(ensemble_Critic_Score == test_set$Critic_Score)
```

```
## [1] 0.9001427
```

As usual, we then compare the predictions estimated to the actual values for *Critic_Score* in *test_set*. Thus, we obtain an overall accuracy of **0.9001**.

## 3. Results

We have employed four algorithms in order to reach our goal (predict the video game scores given by critics): *knn*, *classification trees*, *random forest* and *ensemble*. We have found that their respective accuracy measures are of roughly 0.90 and that the algorithm that provides a slightly higher overall accuracy is the classification tree model, followed by the ensemble approach, as can be seen in the following table.

```
##       algorithm accuracy
## 1           knn   0.8973
## 2 classif. tree   0.9016
## 3 random forest   0.8973
## 4      ensemble   0.9001
```

This results tell us that the techniques we used have a reasonably good (and similar) predictive power.

## 4. Conclusion

We have tried four algorithms to predict the critic scores contained in the **Video Game Sales with Ratings** data set, which was the goal we set for this final project. In the construction of such algorithms, we took as our starting point the different specifications covered in the course. We found that all of the models we estimated have a similar overall accuracy, of around 0.90, which can be considered relatively good. However, there is certainly room for improvement. We could try more sophisticated techniques based, for example, in matrix factorization and principal component analysis, which would probably enhanced the predictive power of our models.

## 5. References


- Data Science: Machine Learning (edX)
- Introduction to Data Science: Data Analysis and Prediction Algorithms with R, Rafael A. Irizarry.
- www.kaggle.com