Similarity and Ensemble - Classification

Sovanna Ramirez

2023-03-20

Introduction

This Classification notebook looks at the data set **Video Game Sales**. The data set was downloaded from here: https://www.kaggle.com/datasets/rush4ratio/video-game-sales-with-ratings (https://www.kaggle.com/datasets/rush4ratio/video-game-sales-with-ratings). We are provided the general structure of our data set as shown below. Name, Platform, Year, Genre, and Publisher provide details on the video game itself. And sales for each different region are represented in millions. We will explore this data set by performing logistic regression, kNN, and decision tree regression and further analyze our results at the end.

```
#The data set is read into and stored in a data frame df
df <- read.csv("~/Downloads/vgsales.csv")
df <- df[,c(2:13)]
df <- na.omit(df)
str(df)
```

```
## 'data.frame':    8137 obs. of  12 variables:
##  $ Platform       : chr  "Wii" "Wii" "Wii" "DS" ...
##  $ Year_of_Release: chr  "2006" "2008" "2009" "2006" ...
##  $ Genre          : chr  "Sports" "Racing" "Sports" "Platform" ...
##  $ Publisher      : chr  "Nintendo" "Nintendo" "Nintendo" "Nintendo" ...
##  $ NA_Sales       : num  41.4 15.7 15.6 11.3 14 ...
##  $ EU_Sales       : num  28.96 12.76 10.93 9.14 9.18 ...
##  $ JP_Sales       : num  3.77 3.79 3.28 6.5 2.93 4.7 4.13 3.6 0.24 2.53 ...
##  $ Other_Sales    : num  8.45 3.29 2.95 2.88 2.84 2.24 1.9 2.15 1.69 1.77 ...
##  $ Global_Sales   : num  82.5 35.5 32.8 29.8 28.9 ...
##  $ Critic_Score   : int  76 82 80 89 58 87 91 80 61 80 ...
##  $ Critic_Count   : int  51 73 73 65 41 80 64 63 45 33 ...
##  $ User_Score     : chr  "8" "8.3" "8" "8.5" ...
##  - attr(*, "na.action")= 'omit' Named int [1:8582] 2 5 6 10 11 13 19 21 22 23 ...
##   ..- attr(*, "names")= chr [1:8582] "2" "5" "6" "10" ...
```

Here, we will use the **as.factor()** function to convert some of the column's data type to be a factor instead. Rank and Name are unique values for each of the video games and we can leave them as is for now. We see the updated structure of our data frame below.

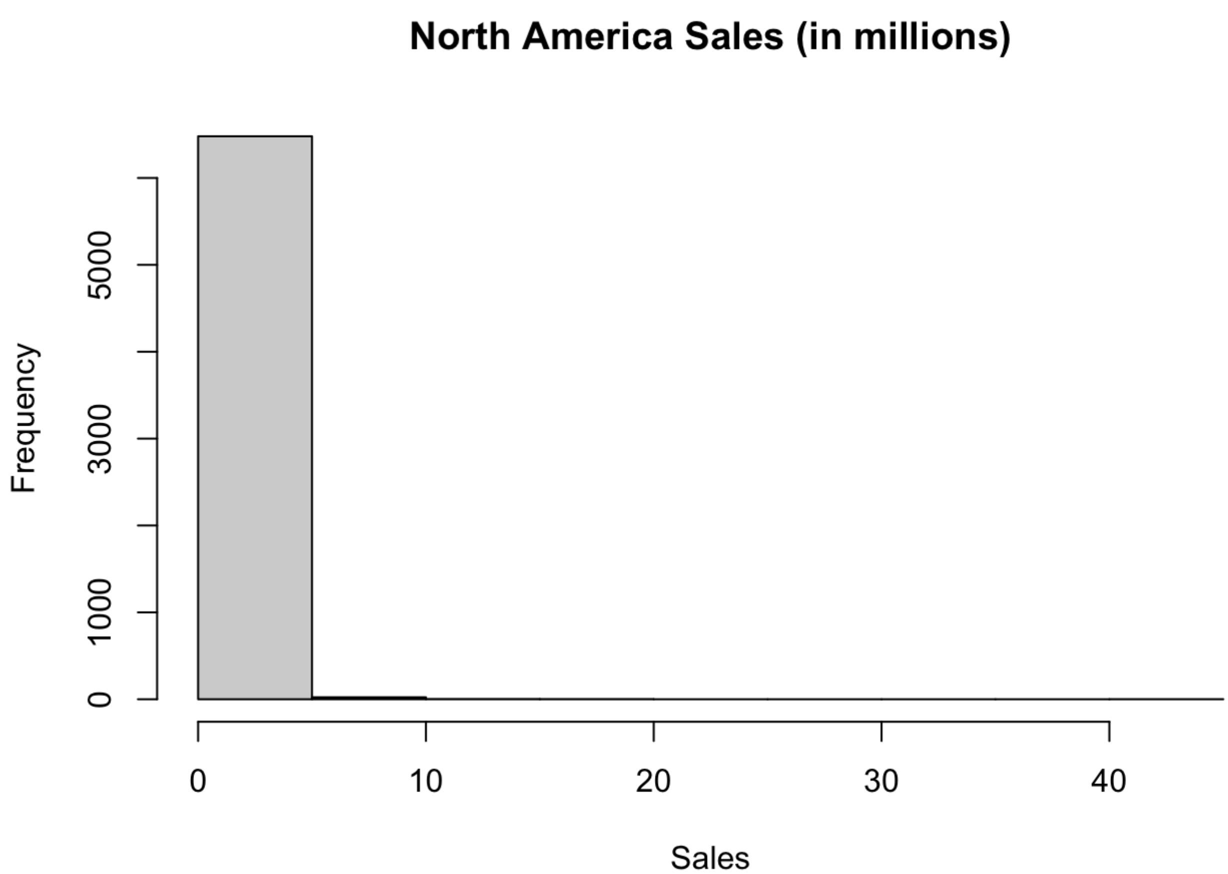
```
df$Platform <- as.factor(df$Platform)
df$Year <- as.factor(df$Year)
df$Genre <- as.factor(df$Genre)
df$Publisher <- as.factor(df$Publisher)
```

```
# Train Test Split
set.seed(1234) # set seed to ensure data remains consistent across runs
i <- sample(1:nrow(df), nrow(df)*0.80, replace=FALSE)
train <- df[i,]
test <- df[-i,]
```

Data Exploration

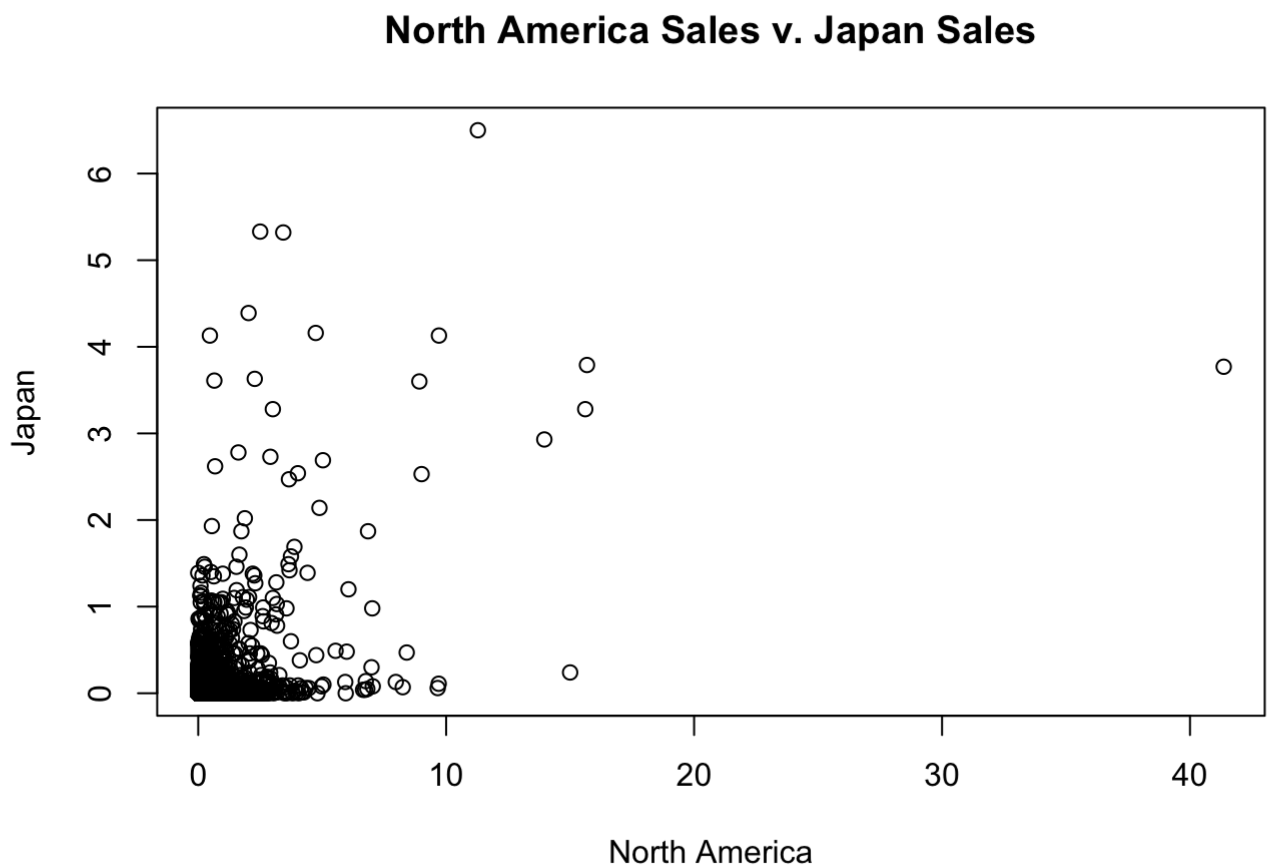
Now that the data is divided, we can explore the training data statistically and graphically. The first graph we have is a histogram of the video game sales in North America.

```
hist(train$NA_Sales, main="North America Sales (in millions)",
xlab="Sales")
```



For our second graph we use **plot()** to compare video game sales in North America and Japan.

```
plot(train$NA_Sales, train$JP_Sales, main="North America Sales v. Japan Sales", xlab =
"North America", ylab = "Japan")
```



Now, using the **summary()** function we can take a look at the summary statistics of the training data set.

```
summary(train)
```

```
##    Platform    Year_of_Release           Genre
##  PS2    :1042   Length:6509        Action      :1511
##  X360   : 716   Class :character   Sports      : 956
##  PS3    : 678   Mode  :character   Shooter     : 767
##  PC     : 591                      Racing      : 597
##  XB     : 581                      Role-Playing: 582
##  DS     : 563                      Misc        : 421
##  (Other):2338                      (Other)     :1675
##                           Publisher       NA_Sales         EU_Sales
##  Electronic Arts              : 815   Min.   : 0.0000   Min.   : 0.0000
##  Activision                   : 447   1st Qu.: 0.0500   1st Qu.: 0.0100
##  Ubisoft                      : 438   Median : 0.1300   Median : 0.0500
##  THQ                          : 334   Mean   : 0.3541   Mean   : 0.2088
##  Sony Computer Entertainment  : 291   3rd Qu.: 0.3300   3rd Qu.: 0.1800
##  Konami Digital Entertainment: 273   Max.   :41.3600   Max.   :28.9600
##  (Other)                      :3911
##     JP_Sales         Other_Sales       Global_Sales       Critic_Score
##  Min.   :0.00000   Min.   :0.0000   Min.   : 0.0100   Min.   :13.00
##  1st Qu.:0.00000   1st Qu.:0.0100   1st Qu.: 0.1000   1st Qu.:60.00
##  Median :0.00000   Median :0.0200   Median : 0.2400   Median :71.00
##  Mean   :0.05433   Mean   :0.0709   Mean   : 0.6884   Mean   :68.88
##  3rd Qu.:0.01000   3rd Qu.:0.0600   3rd Qu.: 0.6400   3rd Qu.:79.00
##  Max.   :6.50000   Max.   :8.4500   Max.   :82.5300   Max.   :98.00
##
##   Critic_Count     User_Score            Year
##  Min.   :  3.00   Length:6509        2008   : 566
##  1st Qu.: 12.00   Class :character   2007   : 549
##  Median : 22.00   Mode  :character   2009   : 531
##  Mean   : 26.31                      2005   : 515
##  3rd Qu.: 36.00                      2002   : 504
##  Max.   :113.00                      2006   : 494
##                                      (Other):3350
```

Logistic Regression With Multiple Predictors

Build a Logistic Regression Model

Here we build a generalized logistic model using the **glm()** function as opposed to lm() in linear regression. In our model we will use “Critic_Score” to predict the target “Global_Sales”. With the **summary()** we can look at null deviance - which tells us the lack of fit using only the intercept and residual deviance - lack of fit of entire model. We see that our residual deviance is lower than our null deviance which is a good sign.

```
df$NA_Sales <- as.factor(df$NA_Sales)
df$JP_Sales <- as.factor(df$JP_Sales)
df$EU_Sales <- as.factor(df$EU_Sales)
df$Other_Sales <- as.factor(df$Other_Sales)
df$Global_Sales <- as.factor(df$Global_Sales)
```

```
set.seed(1234) # set seed to ensure data remains consistent across runs
i <- sample(1:nrow(df), nrow(df)*0.80, replace=FALSE)
train <- df[i,]
test <- df[-i,]
```

```
glm1 <- glm(Global_Sales ~ Platform + Genre, data = train, family = binomial)
summary(glm1)
```

```
## 
## Call:
## glm(formula = Global_Sales ~ Platform + Genre, family = binomial, 
##     data = train)
## 
## Deviance Residuals: 
##     Min       1Q   Median       3Q      Max  
## -3.6687   0.0863   0.1192   0.1817   0.7455  
## 
## Coefficients:
##                     Estimate Std. Error z value Pr(>|z|)    
## (Intercept)          4.94302    1.01783   4.856  1.2e-06 ***
## PlatformDC          13.80634 1942.82129   0.007  0.99433    
## PlatformDS          -0.46576    1.07021  -0.435  0.66342    
## PlatformGBA         -1.73934    1.04131  -1.670  0.09485 .  
## PlatformGC          -0.83302    1.09005  -0.764  0.44475    
## PlatformPC          -2.61458    1.02235  -2.557  0.01055 *  
## PlatformPS          13.70558  517.65850   0.026  0.97888    
## PlatformPS2          0.45827    1.10445   0.415  0.67819    
## PlatformPS3          1.59628    1.42141   1.123  0.26143    
## PlatformPS4         -1.03616    1.12830  -0.918  0.35844    
## PlatformPSP          0.03386    1.16260   0.029  0.97676    
## PlatformPSV         -1.57959    1.16759  -1.353  0.17610    
## PlatformWii          0.56634    1.23174   0.460  0.64567    
## PlatformWiiU        -2.10673    1.13233  -1.861  0.06281 .  
## PlatformX360         0.25983    1.12759   0.230  0.81776    
## PlatformXB          -0.45182    1.08001  -0.418  0.67569    
## PlatformXOne        -1.87066    1.09377  -1.710  0.08721 .  
## GenreAdventure      -0.33231    0.44468  -0.747  0.45488    
## GenreFighting       -0.45991    0.50535  -0.910  0.36278    
## GenreMisc            0.44980    0.62179   0.723  0.46944    
## GenrePlatform        0.31121    0.50534   0.616  0.53800    
## GenrePuzzle         -1.19012    0.44722  -2.661  0.00779 ** 
## GenreRacing         -0.71171    0.32225  -2.209  0.02720 *  
## GenreRole-Playing    0.54532    0.43840   1.244  0.21354    
## GenreShooter         0.09306    0.33867   0.275  0.78348    
## GenreSimulation      0.15347    0.44873   0.342  0.73234    
## GenreSports          0.18920    0.38386   0.493  0.62209    
## GenreStrategy        0.31368    0.43206   0.726  0.46783    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## (Dispersion parameter for binomial family taken to be 1)
## 
##     Null deviance: 1188.2  on 6508  degrees of freedom
## Residual deviance: 1012.9  on 6481  degrees of freedom
## AIC: 1068.9
## 
## Number of Fisher Scoring iterations: 17
```

Evaluate on Test Data Set

```
probs <- predict(glm1, newdata=test, type="response")
pred1 <- ifelse(probs>0.5, 1, 2)
acc1 <- mean(pred1==as.integer(test$Global_Sales))
print(paste("glm1 accuracy = ", acc1))
```

```
## [1] "glm1 accuracy =  0.0184275184275184"
```

Given the accuracy, see that our target “Global_Sales” has almost no linear correlation with “Genre” and “Platform.” To improve this accuracy we may want to utilize the cor() function to find correlations between target and predictors.

kNN

For kNN Classification we won’t be building a model, instead we will just load into memory and make our predictions on the test data all at once. Using genre as our train and test labels

```
#library(class)
#pred2 <- knn(train = train, test = test, cl = df.trainLabels, k = 3)
#acc2 <- length(which(pred2 == df.testLabels)) / length(pred2)
#print(paste("kNN accuracy = ", acc2))
```

Decision Trees

```
library(rpart)
# Build a decision tree with global sales as the target variable
tree <- rpart(Global_Sales ~ Genre, data = train, method = "class")
summary(tree)
```

```
## Call:
## rpart(formula = Global_Sales ~ Genre, data = train, method = "class")
##   n= 6509
##
##   CP nsplit rel error xerror xstd
## 1  0      0         1      0    0
##
## Node number 1: 6509 observations
##   predicted class=0.02  expected loss=0.9623598  P(node) =1
##     class counts:   119   245   213   172   206   186   165   167   146   148   161
142   129   130   122    92   100    99   113    88    96    71    78    81    75    66
70    67    66    50    47    62    64    46    43    37    40    40    49    35    41
48    39    35    41    35    41    22    34    38    32    39    33    31    20    32
33    28    21    26    30    18    24    25    17    30    13    19    13    21    19
24    17    20    23    12    12    12    17    12    13     9    14    12    12     8
21    16    19    23    12    15    17    13    15    10    15    10    13     6    10
16    13    10     8    12     8     9    12     8     6    15    11    11     9     8
11     4    10     6    12     6    11     5     9     5    19     6    10     4     3
5     4    12     8     7     7     7     1     9     6     9     6     5     6     7
3     7     6     6     5     3     6     5     7     4     9     6     4     5     0
5     4     2     3     5     6     7     3     5     4     5     3     3     6     7
2     2     0     2     4     4     2     2     3     6     3     6     1     0     2
6     4     8     4     1     1     3     2     1     3     3     3     4     3     3
3     6     4     3     2     2     2     1     1     5     2     2     6     0     2
3     5     5     2     2     4     2     1     3     2     2     7     2     0     1
4     5     3     1     1     5     2     1     2     2     1     2     3     1     1
0     1     2     4     3     4     1     3     1     3     3     1     1     2     3
5     0     0     2     0     1     0     1     3     2     1     2     1     3     1
3     3     4     1     1     2     2     4     1     1     1     1     3     2     2
2     1     1     3     3     1     2     0     0     1     1     2     1     2     1
1     1     1     0     1     0     2     1     1     1     1     1     2     2     2
0     1     1     2     1     1     1     1     3     1     1     2     1     1     1
2     1     2     1     0     2     1     0     1     2     1     2     2     1     1
2     1     1     4     1     1     1     1     0     1     2     1     2     2     2
1     0     1     1     1     1     1     1     1     1     1     1     0     1     2
1     2     1     1     2     1     1     1     1     2     1     1     1     1     1
1     0     1     1     1     1     1     0     1     1     0     1     1     1     1
0     1     1     1     0     1     0     0     1     1     1     1     1     1     3
2     1     1     0     2     1     0     2     1     0     1     1     0     1     1
1     0     1     1     0     1     1     1     0     1     1     1     0     1     1
1     0     1     1     1     1     0     1     1     1     1     0     1     2     1
1     1     1     1     1     0     1     1     1     0     1     0     1     1     1
0     1     1     1     0     1     1     1     1     1     1     1     1     0     1
1     1     1     1     1     1     1     0     1     1     0     1     1     0     1
0     1     0     1     1     1     1     1     0     1     1     1     1     1     0
1     1     1     1     1
##    probabilities: 0.018 0.038 0.033 0.026 0.032 0.029 0.025 0.026 0.022 0.023 0.025
0.022 0.020 0.020 0.019 0.014 0.015 0.015 0.017 0.014 0.015 0.011 0.012 0.012 0.012 0.01
0 0.011 0.010 0.010 0.008 0.007 0.010 0.010 0.007 0.007 0.006 0.006 0.006 0.008 0.005 0.
006 0.007 0.006 0.005 0.006 0.005 0.006 0.003 0.005 0.006 0.005 0.006 0.005 0.005 0.003
0.005 0.005 0.004 0.003 0.004 0.005 0.003 0.004 0.004 0.003 0.005 0.002 0.003 0.002 0.00
3 0.003 0.004 0.003 0.003 0.004 0.002 0.002 0.002 0.003 0.002 0.002 0.001 0.002 0.002 0.
```

```
002 0.001 0.003 0.002 0.003 0.004 0.002 0.002 0.003 0.002 0.002 0.002 0.002 0.002 0.002
0.001 0.002 0.002 0.002 0.002 0.001 0.002 0.001 0.001 0.002 0.001 0.001 0.002 0.002 0.00
2 0.001 0.001 0.002 0.001 0.002 0.001 0.002 0.001 0.002 0.001 0.001 0.001 0.003 0.001 0.
002 0.001 0.000 0.001 0.001 0.002 0.001 0.001 0.001 0.001 0.000 0.001 0.001 0.001 0.001
0.001 0.001 0.001 0.000 0.001 0.001 0.001 0.001 0.000 0.001 0.001 0.001 0.001 0.001 0.00
1 0.001 0.001 0.000 0.001 0.001 0.000 0.000 0.001 0.001 0.001 0.000 0.001 0.001 0.001 0.
000 0.000 0.001 0.001 0.000 0.000 0.000 0.000 0.001 0.001 0.000 0.000 0.000 0.001 0.000
0.001 0.000 0.000 0.000 0.001 0.001 0.001 0.001 0.000 0.000 0.000 0.000 0.000 0.000 0.00
0 0.000 0.001 0.000 0.000 0.000 0.001 0.001 0.000 0.000 0.000 0.000 0.000 0.000 0.001 0.
000 0.000 0.001 0.000 0.000 0.000 0.001 0.001 0.000 0.000 0.001 0.000 0.000 0.000 0.000
0.000 0.001 0.000 0.000 0.000 0.001 0.001 0.000 0.000 0.000 0.001 0.000 0.000 0.000 0.00
0 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.001 0.000 0.001 0.000 0.000 0.000 0.
000 0.000 0.000 0.000 0.000 0.000 0.001 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.001 0.000 0.000 0.000 0.000 0.001 0.00
0 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.
000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.00
0 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.
000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.001 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.00
0 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.
000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.00
0 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.
000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.00
0 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.
000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.00
0 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.
000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000
```

```
pred3 <- predict(tree, newdata=test, type="class")
acc3 <- mean(pred3==test$Global_Sales)
print(paste("tree accuracy = ", acc3))
```

```
## [1] "tree accuracy =  0.0405405405405405"
```

Analysis & Results

After performing logistic regression, kNN, and decision tree regression we see that they all came up with similar accuracies. Out of the three algorithms, logistic regression was the easiest to interpret the fit of the model given the null deviance and residual deviance. With kNN we did not have any model for interpretation. Moreover, kNN required more training in finding distances between the train and test samples. And our decision tree could be intrepreted by taking a look at the root node down to the leaf nodes. All three algorithms had the capability to model non-linear relationships. To improve the accuracy of my logistic regression model, it would have been wise to start by finding the correlation between predictors and my target using the cor() function. Overall, we see that the decision tree had the best accuracy (although still very low).