Classification

Micah Katz

2/17/2023

How Linear Models For Classification Work

Linear Models for Classification allow for prediction of the classifications of items through linear means. We find where the boundary is between two classifications of a set of data and use a linear system to describe it.

Dataset

https://www.kaggle.com/datasets/thedevastator/global-video-game-sales

Import Dataset

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
vgsales <- read.csv("vgsales.csv")
vgsales$Platform <- as.factor(vgsales$Platform)
```

Split into 80/20 training data

We split the data using the `sample()` function and make sure to split the data 80/20 between training and testing. We find the index to split the data on and assign it to `i` and then separate the data between training and testing.

```
i <- sample(1:nrow(vgsales), 0.80*nrow(vgsales), replace=FALSE)
training <- vgsales[i, ]
testing <- vgsales[-i, ]
```

Functions for data exploration

Here we are using different functions to explore the given data. `str()` will give us the structure of the data. `summary()` will give us a summary of the data. `hist()` will give us a histogram and we are plotting a histogram of the North America Sales. The `cor()` function will tell us the correlation of NA_Sales to EU_Sales. The `head()` function will show the first 6 rows of the training data.

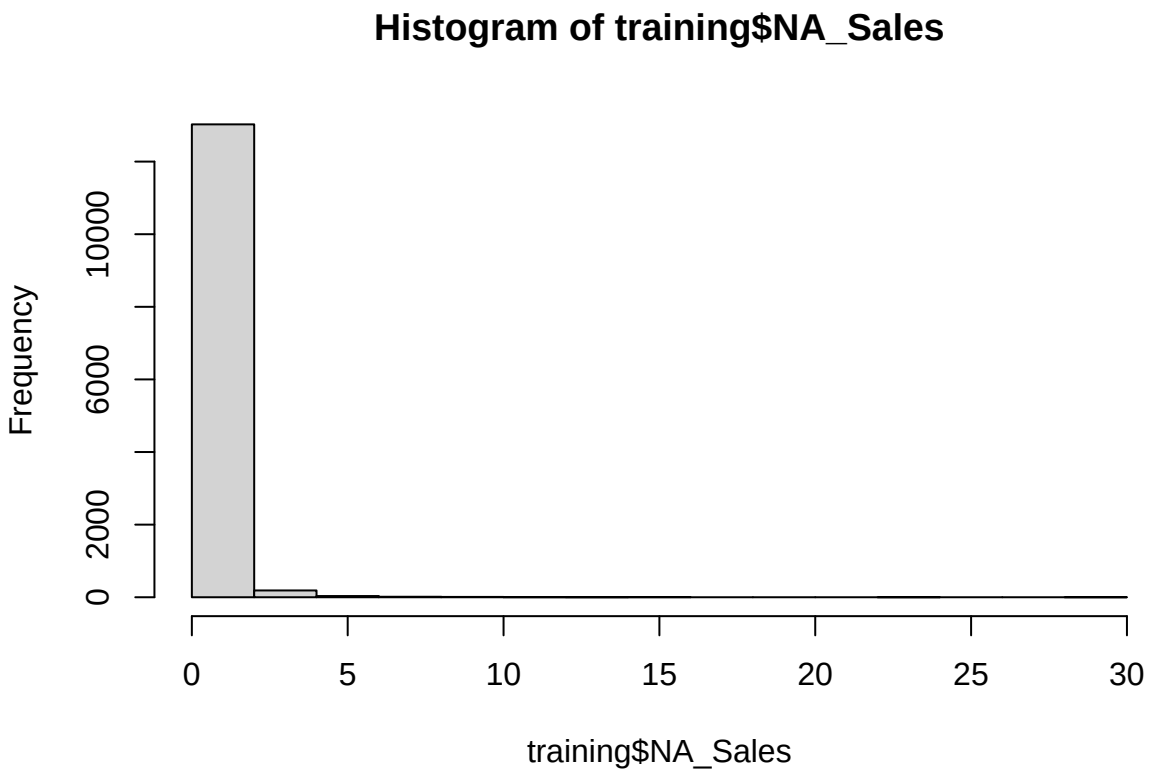
```
str(training)
```

```
## 'data.frame':    13278 obs. of  11 variables:
##  $ Rank        : int  14461 1993 15547 5846 14825 446 6702 3114 15901 15868 ...
##  $ Name        : chr  "Pet Alien: An Intergalactic Puzzlepalooza" "Dragon Ball Z: Budokai Tenkaichi 3
##  $ Platform    : Factor w/ 31 levels "2600","3DO","3DS",..: 5 26 5 5 26 18 16 29 17 5 ...
##  $ Year        : chr  "2007" "2007" "2006" "2010" ...
##  $ Genre       : chr  "Action" "Fighting" "Misc" "Action" ...
##  $ Publisher   : chr  "Game Factory" "Atari" "Atlus" "Warner Bros. Interactive Entertainment" ...
##  $ NA_Sales    : num  0.03 0.33 0 0.24 0.02 1.69 0.14 0.39 0 0.01 ...
##  $ EU_Sales    : num  0 0.37 0 0.04 0 0.87 0.09 0.2 0 0 ...
##  $ JP_Sales    : num  0 0.26 0.02 0 0 0.14 0 0 0.02 0 ...
##  $ Other_Sales : num  0 0.09 0 0.02 0 0.42 0.02 0.06 0 0 ...
##  $ Global_Sales: num  0.03 1.04 0.02 0.3 0.03 3.12 0.25 0.65 0.02 0.02 ...
```

```
summary(training)
```

```
##       Rank           Name              Platform        Year
##  Min.   :    2   Length:13278       DS     :1763   Length:13278
##  1st Qu.: 4140   Class :character   PS2    :1743   Class :character
##  Median : 8286   Mode  :character   PS3    :1048   Mode  :character
##  Mean   : 8298                      Wii    :1035
##  3rd Qu.:12474                      X360   :1028
##  Max.   :16600                      PSP    : 959
##                                     (Other):5702
##     Genre            Publisher           NA_Sales          EU_Sales
##  Length:13278       Length:13278       Min.   : 0.0000   Min.   : 0.0000
##  Class :character   Class :character   1st Qu.: 0.0000   1st Qu.: 0.0000
##  Mode  :character   Mode  :character   Median : 0.0800   Median : 0.0200
##                                        Mean   : 0.2619   Mean   : 0.1435
##                                        3rd Qu.: 0.2400   3rd Qu.: 0.1100
##                                        Max.   :29.0800   Max.   :11.0100
##
##     JP_Sales         Other_Sales        Global_Sales
##  Min.   : 0.00000   Min.   : 0.00000   Min.   : 0.0100
##  1st Qu.: 0.00000   1st Qu.: 0.00000   1st Qu.: 0.0600
##  Median : 0.00000   Median : 0.01000   Median : 0.1700
##  Mean   : 0.07801   Mean   : 0.04702   Mean   : 0.5307
##  3rd Qu.: 0.04000   3rd Qu.: 0.04000   3rd Qu.: 0.4800
##  Max.   :10.22000   Max.   :10.57000   Max.   :40.2400
##
```

```
hist(training$NA_Sales)
```




```
cor(training[,c("NA_Sales", "EU_Sales")])
```

```
##           NA_Sales  EU_Sales
## NA_Sales 1.0000000 0.7342018
## EU_Sales 0.7342018 1.0000000
```

```
head(training)
```

```
##        Rank                                          Name Platform Year
## 14459 14461    Pet Alien: An Intergalactic Puzzlepalooza       DS 2007
## 1992   1993           Dragon Ball Z: Budokai Tenkaichi 3      Wii 2007
## 15545 15547                                Jinsei Game DS       DS 2006
## 5845   5846 Batman: The Brave and the Bold the Videogame       DS 2010
## 14823 14825                        Backyard NFL Football      Wii 2007
## 446     446               The Elder Scrolls IV: Oblivion      PS3 2007
##              Genre                                Publisher NA_Sales EU_Sales
## 14459       Action                              Game Factory     0.03     0.00
## 1992      Fighting                                     Atari     0.33     0.37
## 15545         Misc                                     Atlus     0.00     0.00
## 5845        Action Warner Bros. Interactive Entertainment     0.24     0.04
## 14823       Sports                                     Atari     0.02     0.00
## 446   Role-Playing                                   Ubisoft     1.69     0.87
##       JP_Sales Other_Sales Global_Sales
## 14459     0.00        0.00         0.03
```

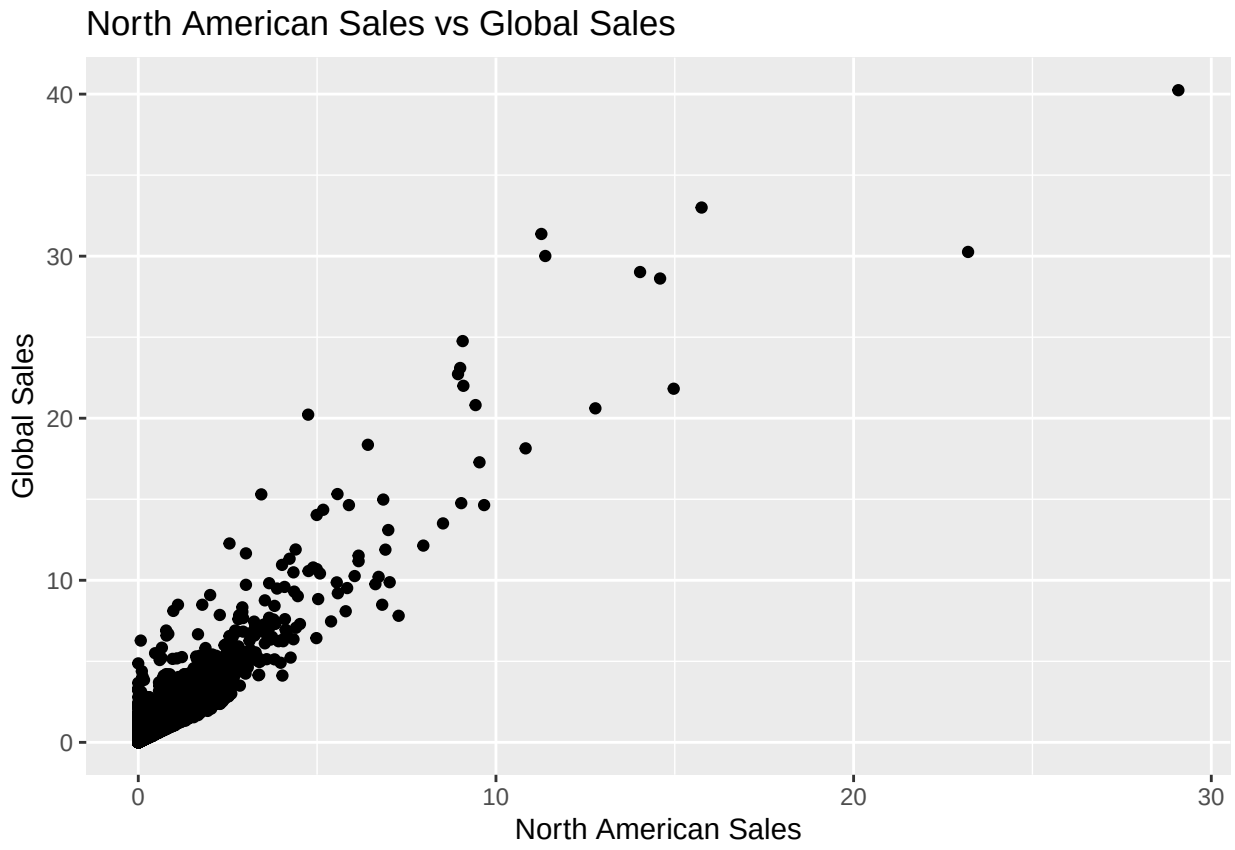
```
## 1992      0.26        0.09        1.04
## 15545     0.02        0.00        0.02
## 5845      0.00        0.02        0.30
## 14823     0.00        0.00        0.03
## 446       0.14        0.42        3.12
```

Graphs for Data Exploration

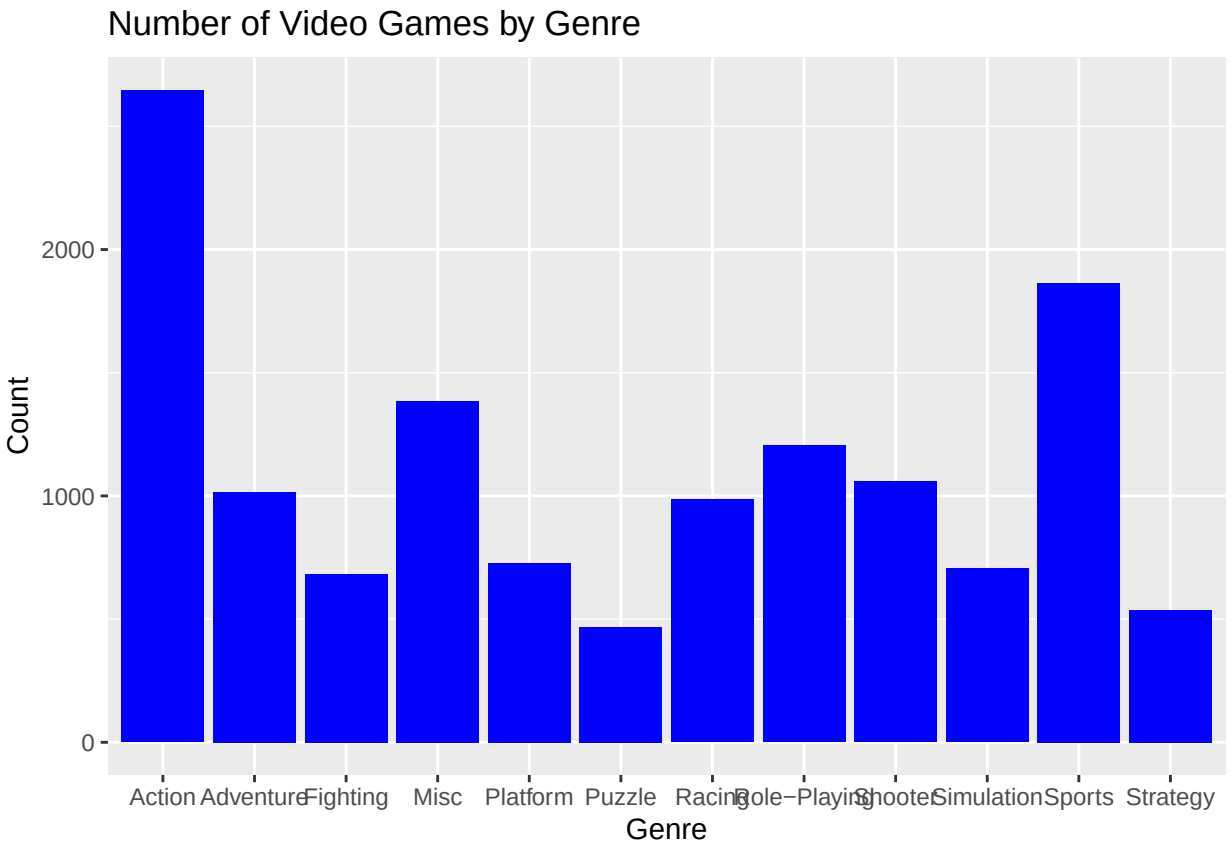
We use `ggplot2` to graph a scatter plot of the training data of the North American Sales vs the Global Sales. We then plot a bar graph of Number of Video Games by Genre

```
library(ggplot2)

ggplot(training, aes(x = NA_Sales, y = Global_Sales)) +
  geom_point() +
  labs(title = "North American Sales vs Global Sales",
       x = "North American Sales",
       y = "Global Sales")
```




```
ggplot(training, aes(x = Genre)) +
  geom_bar(fill = "blue") +
  labs(title = "Number of Video Games by Genre",
       x = "Genre",
       y = "Count")
```



Logistic Regression Model for Classification

Here we are building a logistic regression model that will predict the Platform for a game based on the genre of the game. We get in the summary the deviance residuals that show how well the model fits the data. We have Min which is the smallest residual, the first quartile is 1Q, then we have the median, the third quartile, and the maximum. We then see the coefficients that correspond to each genre. These tell us the estimates of the coefficients, the standard error, the z value, and the p values. We then get the null deviance and residual deviance which tells us the deviance if there were no predictors, and the deviance of the model that was fitted, respectively.

```
logistic_model <- glm(Platform ~ Genre, data = training, family = binomial())
summary(logistic_model)
```

```
##
## Call:
## glm(formula = Platform ~ Genre, family = binomial(), data = training)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.6220   0.0628   0.1037   0.1800   0.1953
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)         3.9497     0.1428  27.663  < 2e-16 ***
```

```
## GenreAdventure      2.2778     0.7221   3.155 0.001607 **
## GenreFighting       2.5753     1.0109   2.548 0.010845 *
## GenreMisc           1.6707     0.4702   3.553 0.000381 ***
## GenrePlatform       1.0229     0.4709   2.172 0.029850 *
## GenrePuzzle         0.3919     0.4350   0.901 0.367584
## GenreRacing         1.3284     0.4705   2.823 0.004755 **
## GenreRole-Playing  16.6164   510.9808   0.033 0.974059
## GenreShooter        0.1650     0.2831   0.583 0.560017
## GenreSimulation     2.6085     1.0108   2.581 0.009865 **
## GenreSports         1.2728     0.3477   3.660 0.000252 ***
## GenreStrategy      16.6164   764.4105   0.022 0.982657
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1196.5  on 13277  degrees of freedom
## Residual deviance: 1106.7  on 13266  degrees of freedom
## AIC: 1130.7
##
## Number of Fisher Scoring iterations: 19
```

Naive Bayes Model

Here we are using the `e1071` library to call the `naiveBayes()` function. We are predicting which `Platform` a game will be based on its `Genre` . This method will assume that `Genre` and `Platform` are independent which is the "naive" approach.

```
library(e1071)

nb_model <- naiveBayes(Platform ~ Genre, data = training)

print(nb_model)
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##         2600          3DO          3DS           DC           DS           GB
## 7.681880e-03 1.506251e-04 3.042627e-02 3.313752e-03 1.327760e-01 6.702817e-03
##          GBA           GC          GEN           GG          N64          NES
## 4.993222e-02 3.313752e-02 1.581563e-03 7.531255e-05 1.890345e-02 5.949691e-03
##           NG           PC         PCFX           PS          PS2          PS3
## 6.778129e-04 5.716222e-02 7.531255e-05 7.184817e-02 1.312698e-01 7.892755e-02
##          PS4          PSP          PSV          SAT          SCD         SNES
## 1.973189e-02 7.222473e-02 2.372345e-02 1.054376e-02 3.765627e-04 1.446001e-02
##         TG16          Wii         WiiU           WS         X360           XB
## 1.506251e-04 7.794849e-02 9.112818e-03 2.259376e-04 7.742130e-02 5.098659e-02
##         XOne
```

```
## 1.250188e-02
##
## Conditional probabilities:
##       Genre
## Y           Action   Adventure    Fighting        Misc    Platform      Puzzle
##   2600 0.490196078 0.019607843 0.009803922 0.049019608 0.049019608 0.058823529
##   3DO  0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.500000000
##   3DS  0.356435644 0.071782178 0.029702970 0.091584158 0.051980198 0.037128713
##   DC   0.022727273 0.181818182 0.250000000 0.000000000 0.045454545 0.000000000
##   DS   0.162790698 0.109472490 0.017016449 0.181508792 0.045377198 0.111741350
##   GB   0.067415730 0.056179775 0.000000000 0.089887640 0.213483146 0.123595506
##   GBA  0.200603318 0.049773756 0.030165913 0.129713424 0.173453997 0.048265460
##   GC   0.184090909 0.029545455 0.081818182 0.065909091 0.136363636 0.027272727
##   GEN  0.142857143 0.095238095 0.238095238 0.047619048 0.190476190 0.000000000
##   GG   0.000000000 0.000000000 0.000000000 0.000000000 1.000000000 0.000000000
##   N64  0.123505976 0.015936255 0.091633466 0.055776892 0.095617530 0.031872510
##   NES  0.164556962 0.012658228 0.037974684 0.012658228 0.291139241 0.139240506
##   NG   0.000000000 0.000000000 0.888888889 0.000000000 0.000000000 0.000000000
##   PC   0.173913043 0.069828722 0.006587615 0.028985507 0.010540184 0.027667984
##   PCFX 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
##   PS   0.129979036 0.053459119 0.090146751 0.064989518 0.053459119 0.027253669
##   PS2  0.160642570 0.088927137 0.070567986 0.103270224 0.048766495 0.007458405
##   PS3  0.275763359 0.058206107 0.055343511 0.087786260 0.030534351 0.002862595
##   PS4  0.358778626 0.057251908 0.041984733 0.049618321 0.038167939 0.003816794
##   PSP  0.176225235 0.163712200 0.059436913 0.083420229 0.032325339 0.037539103
##   PSV  0.349206349 0.209523810 0.034920635 0.057142857 0.022222222 0.006349206
##   SAT  0.021428571 0.150000000 0.185714286 0.078571429 0.028571429 0.028571429
##   SCD  0.000000000 0.000000000 0.000000000 0.400000000 0.200000000 0.000000000
##   SNES 0.057291667 0.015625000 0.098958333 0.078125000 0.114583333 0.046875000
##   TG16 0.000000000 0.500000000 0.000000000 0.000000000 0.000000000 0.000000000
##   Wii  0.186473430 0.060869565 0.034782609 0.212560386 0.041545894 0.041545894
##   WiiU 0.429752066 0.024793388 0.033057851 0.132231405 0.123966942 0.024793388
##   WS   0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
##   X360 0.255836576 0.039883268 0.047665370 0.098249027 0.019455253 0.006809339
##   XB   0.183161004 0.033973412 0.062038405 0.062038405 0.059084195 0.008862629
##   XOne 0.319277108 0.072289157 0.042168675 0.060240964 0.024096386 0.000000000
##       Genre
## Y           Racing Role-Playing     Shooter  Simulation      Sports    Strategy
##   2600 0.049019608  0.000000000 0.166666667 0.009803922 0.098039216 0.000000000
##   3DO  0.000000000  0.000000000 0.000000000 0.500000000 0.000000000 0.000000000
##   3DS  0.019801980  0.175742574 0.017326733 0.061881188 0.054455446 0.032178218
##   DC   0.113636364  0.090909091 0.068181818 0.022727273 0.204545455 0.000000000
##   DS   0.028927964  0.090187181 0.018718094 0.130459444 0.065229722 0.038570618
##   GB   0.022471910  0.202247191 0.011235955 0.044943820 0.089887640 0.078651685
##   GBA  0.076923077  0.090497738 0.045248869 0.025641026 0.111613876 0.018099548
##   GC   0.104545455  0.043181818 0.088636364 0.022727273 0.200000000 0.015909091
##   GEN  0.000000000  0.142857143 0.047619048 0.000000000 0.047619048 0.047619048
##   GG   0.000000000  0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
##   N64  0.183266932  0.027888446 0.071713147 0.031872510 0.239043825 0.031872510
##   NES  0.050632911  0.075949367 0.063291139 0.000000000 0.151898734 0.000000000
##   NG   0.000000000  0.000000000 0.000000000 0.000000000 0.111111111 0.000000000
##   PC   0.060606061  0.110671937 0.154150198 0.127799736 0.046113307 0.183135705
##   PCFX 0.000000000  1.000000000 0.000000000 0.000000000 0.000000000 0.000000000
##   PS   0.121593291  0.084905660 0.072327044 0.053459119 0.190775681 0.057651992
```

```
##   PS2  0.095811819  0.087779690 0.077452668 0.039586919 0.186460126 0.033275961
##   PS3  0.063931298  0.091603053 0.123091603 0.025763359 0.166984733 0.018129771
##   PS4  0.049618321  0.133587786 0.099236641 0.019083969 0.137404580 0.011450382
##   PSP  0.056308655  0.172054223 0.031282586 0.021897810 0.112617310 0.053180396
##   PSV  0.031746032  0.193650794 0.015873016 0.006349206 0.050793651 0.022222222
##   SAT  0.050000000  0.100000000 0.142857143 0.021428571 0.092857143 0.100000000
##   SCD  0.200000000  0.200000000 0.000000000 0.000000000 0.000000000 0.000000000
##   SNES 0.046875000  0.223958333 0.046875000 0.020833333 0.203125000 0.046875000
##   TG16 0.000000000  0.000000000 0.500000000 0.000000000 0.000000000 0.000000000
##   Wii  0.075362319  0.026086957 0.052173913 0.063768116 0.185507246 0.019323671
##   WiiU 0.008264463  0.049586777 0.082644628 0.008264463 0.057851240 0.024793388
##   WS   0.000000000  0.666666667 0.000000000 0.000000000 0.000000000 0.333333333
##   X360 0.084630350  0.059338521 0.156614786 0.035992218 0.174124514 0.021400778
##   XB   0.143279173  0.025110783 0.162481536 0.033973412 0.196454948 0.029542097
##   XOne 0.084337349  0.060240964 0.168674699 0.018072289 0.144578313 0.006024096
```

Comparing Models

Here we are comparing the two models using the predictions. We use the MSE and Correlation to compare the models and output the results.

```
# Predict on test data using logistic regression model
logistic_preds <- predict(logistic_model, newdata = testing, type = 'response')
logistic_preds <- factor(ifelse(logistic_preds > 0.5, "Wii", "NES"), levels = levels(testing$Platform))



nb_preds <- predict(nb_model, newdata = testing)
nb_preds <- factor(nb_preds, levels = levels(testing$Platform))

logistic_cm <- table(logistic_preds, testing$Platform)
nb_cm <- table(nb_preds, testing$Platform)

logistic_recall <- confusionMatrix(logistic_cm)$byClass['Recall']
logistic_accuracy <- confusionMatrix(logistic_cm)$overall['Accuracy']
logistic_precision <- confusionMatrix(logistic_cm)$byClass['Precision']

nb_recall <- confusionMatrix(nb_cm)$byClass['Recall']
nb_accuracy <- confusionMatrix(nb_cm)$overall['Accuracy']
nb_precision <- confusionMatrix(nb_cm)$byClass['Precision']

cat("Logistic Regression:\n")
```

```
## Logistic Regression:
```

```
cat("Accuracy: ", logistic_accuracy, "\n")
```

```
## Accuracy:  0.0873494
```

```
cat("Precision: ", logistic_precision, "\n")
```

```
## Precision:  NA
```

```
cat("Recall: ", logistic_recall, "\n")
```

```
## Recall:  NA
```

```
cat("Naive Bayes:\n")
```

```
## Naive Bayes:
```

```
cat("Accuracy: ", nb_accuracy, "\n")
```

```
## Accuracy:  0.1816265
```

```
cat("Precision: ", nb_precision, "\n")
```

```
## Precision:  NA
```

```
cat("Recall: ", nb_recall, "\n")
```

```
## Recall:  NA
```

Strengths and weaknesses of Naïve Bayes and Logistic Regression

It seems that the Naive Bayes is more accurate than the logistic regression in this case.

Benefits, drawbacks of each of the classification metrics

There are benefits to the Naive Bayes method. If your factors are truly independent then it is a great way to determine that. As for the logistic regression, it is better for determining values that fall into separate categories and are able to be separated by a line.