ML Portfolio 2: Logistic Regression & Naive Bayes

Bushra Rahman

How Logistic Regression Works

Logistic regression, contrary to its name, is a qualitative classification algorithm that creates linear decision boundaries between data points of different classes. Like linear regression, logistic regression models the equation $y = wx + b$, but the purpose of this line is for classification, not regression. The variable x represents the predictor value(s), while y represents a qualitative target value. Classification can determine whether the target is a member of one class or multiple classes. Like linear regression, logistic regression is a high-bias low-variance algorithm.

Logistic Regression on VGSales

The CSV file **"vgsales"** is a dataset of video games with sales greater than 100,000 copies, scraped from the VGChartz Network and uploaded to Kaggle (URL https://www.kaggle.com/datasets/gregorut/videogamesales (https://www.kaggle.com/datasets/gregorut/videogamesales)). This dataset contains 16,598 rows of video game titles and 11 columns of variables. Of those 11 variables, 3 are qualitative factors, all with multiple levels: Platform, Publisher, and Genre. One-versus-all classification can be done on one of these factors, but it would be easier to create a new binary factor representing two parts of of one factor. **Genre** in particular seems interesting: Perhaps there is a trend where a certain genre might outperform others in sales. Logistic regression can be used to determine if such a trend exists for one of the genres: **Action**.

Train and Test Sets

After importing the CSV file into a data frame, its factors need to be designated as such using the **as.factor()** function.

Then, using the information in summary(), str(), and levels() below, it is determined that Action is the most frequent genre in the data. Do action games have higher sales than other games? A new factor **isAction** is created for vgsales to represent whether a game is Action or not.

Finally, the data can be divided into a train set and a test set. By setting a seed and randomly sampling the row indices into a vector, a train set can be created that contains 80% of the rows. The remaining 20% are then put into the test set.

```
# import data
setwd("C:/Users/Bushra/CS4375/Portfolio2")
vgsales <- read.csv("vgsales.csv")
# set factors
vgsales$Genre <- as.factor(vgsales$Genre)
vgsales$Platform <- as.factor(vgsales$Platform)
vgsales$Publisher <- as.factor(vgsales$Publisher)
# create new factor isAction that is TRUE for Action games only
vgsales$isAction <- FALSE
vgsales$isAction[vgsales$Genre=="Action"] <- TRUE
vgsales$isAction <- factor(vgsales$isAction)
# set train and test sets
set.seed(1234)
i <- sample(1:nrow(vgsales), nrow(vgsales)*0.8, replace=FALSE)
train <- vgsales[i,]
test <- vgsales[-i,]
```

Data Exploration

Now that the training data has been procured, it can be explored using R's built-in data exploration functions. The 6 functions used below are dim(), str(), levels(), summary(), head(), and tail().

The **dim()** function returns the row and column dimensions of the data frame, which are 13278 and 12 respectively (with an extra column now representing isAction). This is because 13278 is 80% of 16598, the size of the entire vgsales dataset.

```
dim(train)
```

```
## [1] 13278    12
```

The **str()** function displays information on the data frame's structure, like the column names, their data type, and the first few entries in each column. This output shows that Rank is an integer vector, the 3 original factors along with Name and Year are character string vectors, and the rest of the columns are numerical vectors. Because Genre, Platform, and Publisher were converted into factors when vgsales was read in, str() now shows how many levels each factor has: Platform has 31 levels, Genre has 12 levels, and Publisher has 579 levels. Finally, the new binary column isAction is a factor with 2 levels, TRUE and FALSE.

```
str(train)
```

```
## 'data.frame':    13278 obs. of  12 variables:
##  $ Rank        : int  7453 8017 7163 8087 9197 623 15243 10886 935 12689 ...
##  $ Name        : chr  "Mortal Kombat: Special Forces" "Reel Fishing II" "Dark Souls II" "Batt
le Commander: Hachibushu Shura no Heihou" ...
##  $ Platform    : Factor w/ 31 levels "2600","3DO","3DS",..: 16 16 31 24 8 16 17 26 26 30 ...
##  $ Year        : chr  "2000" "2000" "2015" "1991" ...
##  $ Genre       : Factor w/ 12 levels "Action","Adventure",..: 3 11 8 12 11 1 2 11 1 1 ...
##  $ Publisher   : Factor w/ 579 levels "10TACLE Studios",..: 330 549 352 62 330 138 276 227 17
139 ...
##  $ NA_Sales    : num  0.12 0.1 0.13 0 0.11 1.15 0 0.09 0.85 0.04 ...
##  $ EU_Sales    : num  0.08 0.07 0.07 0 0.03 1.14 0 0 0.71 0.01 ...
##  $ JP_Sales    : num  0 0 0 0.18 0 0.06 0.02 0 0.13 0 ...
##  $ Other_Sales : num  0.01 0.01 0.02 0 0 0.13 0 0.01 0.16 0 ...
##  $ Global_Sales: num  0.21 0.18 0.22 0.18 0.14 2.48 0.02 0.09 1.86 0.06 ...
##  $ isAction    : Factor w/ 2 levels "FALSE","TRUE": 1 1 1 1 1 2 1 1 2 2 ...
```

A function that can be used specifically on factors is the **levels()** function, which outputs the ordering of all levels in the factor. This is crucial information for understanding how the factor is organized. The output of levels() on Genre shows that the factors, in order, are “Action”, “Adventure”, “Fighting”, “Misc”, “Platform”, “Puzzle”, “Racing”, “Role-Playing”, “Shooter”, “Simulation”, “Sports”, and “Strategy”. Of these genres, Action was picked as the one to investigate in relation to sales. Running levels() on isAction shows that FALSE is designated by 1 and TRUE is designated by 2.

```
levels(train$Genre)
```

```
##  [1] "Action"       "Adventure"    "Fighting"     "Misc"         "Platform"
##  [6] "Puzzle"       "Racing"       "Role-Playing" "Shooter"      "Simulation"
## [11] "Sports"       "Strategy"
```

```
levels(train$isAction)
```

```
## [1] "FALSE" "TRUE"
```

The **summary()** function returns statistical information on each column. Because the factors are now codified as such, there is unique statistical information for them: the most frequent levels are listed in descending order, with all other levels listed as (Other). Under Genre, it appears that Action is the most common, followed by Sports, Misc, Role-Playing, Shooter, and Adventure. For isAction, FALSE is given as 10639 and TRUE as 2639, indicating that although Action is the most frequent single genre compared to all other genres, the total of all other genres outweighs Action. This imbalance may explain potential future inaccuracies in the logistic model that will be produced.

```
summary(train)
```

```
##       Rank           Name              Platform        Year
##  Min.   :    1   Length:13278       DS     :1730   Length:13278
##  1st Qu.: 4156   Class :character   PS2    :1723   Class :character
##  Median : 8324   Mode  :character   PS3    :1068   Mode  :character
##  Mean   : 8314                      Wii    :1031
##  3rd Qu.:12444                      X360   :1019
##  Max.   :16600                      PS     : 967
##                                     (Other):5740
##          Genre                             Publisher       NA_Sales
##  Action      :2639   Electronic Arts             :1076   Min.   : 0.0000
##  Sports      :1850   Activision                  : 767   1st Qu.: 0.0000
##  Misc        :1406   Namco Bandai Games          : 744   Median : 0.0800
##  Role-Playing:1219   Ubisoft                     : 721   Mean   : 0.2627
##  Shooter     :1048   Konami Digital Entertainment: 673   3rd Qu.: 0.2400
##  Adventure   :1039   THQ                         : 588   Max.   :41.4900
##  (Other)     :4077   (Other)                     :8709
##     EU_Sales          JP_Sales         Other_Sales       Global_Sales
##  Min.   : 0.0000   Min.   : 0.00000   Min.   : 0.0000   Min.   : 0.0100
##  1st Qu.: 0.0000   1st Qu.: 0.00000   1st Qu.: 0.0000   1st Qu.: 0.0600
##  Median : 0.0200   Median : 0.00000   Median : 0.0100   Median : 0.1700
##  Mean   : 0.1456   Mean   : 0.07825   Mean   : 0.0483   Mean   : 0.5352
##  3rd Qu.: 0.1100   3rd Qu.: 0.04000   3rd Qu.: 0.0400   3rd Qu.: 0.4700
##  Max.   :29.0200   Max.   :10.22000   Max.   :10.5700   Max.   :82.7400
##
##   isAction
##  FALSE:10639
##  TRUE : 2639
##
##
##
##
##
```

Finally, the **head()** and **tail()** functions display the first and last few rows of the training data. This output provides a glimpse at how the actual data is organized in table format.

```
head(train)
```

	R…	Name	Platform	Y…	Genre
	<int>	<chr>	<fct>	<chr>	<fct>
7452	7453	Mortal Kombat: Special Forces	PS	2000	Fighting
8016	8017	Reel Fishing II	PS	2000	Sports
7162	7163	Dark Souls II	XOne	2015	Role-Playing
8086	8087	Battle Commander: Hachibushu Shura no Heihou	SNES	1991	Strategy
9196	9197	NFL Blitz 20-03	GC	2002	Sports
623	623	Tomb Raider: The Last Revelation	PS	1998	Action

6 rows | 1-6 of 13 columns

```
tail(train)
```

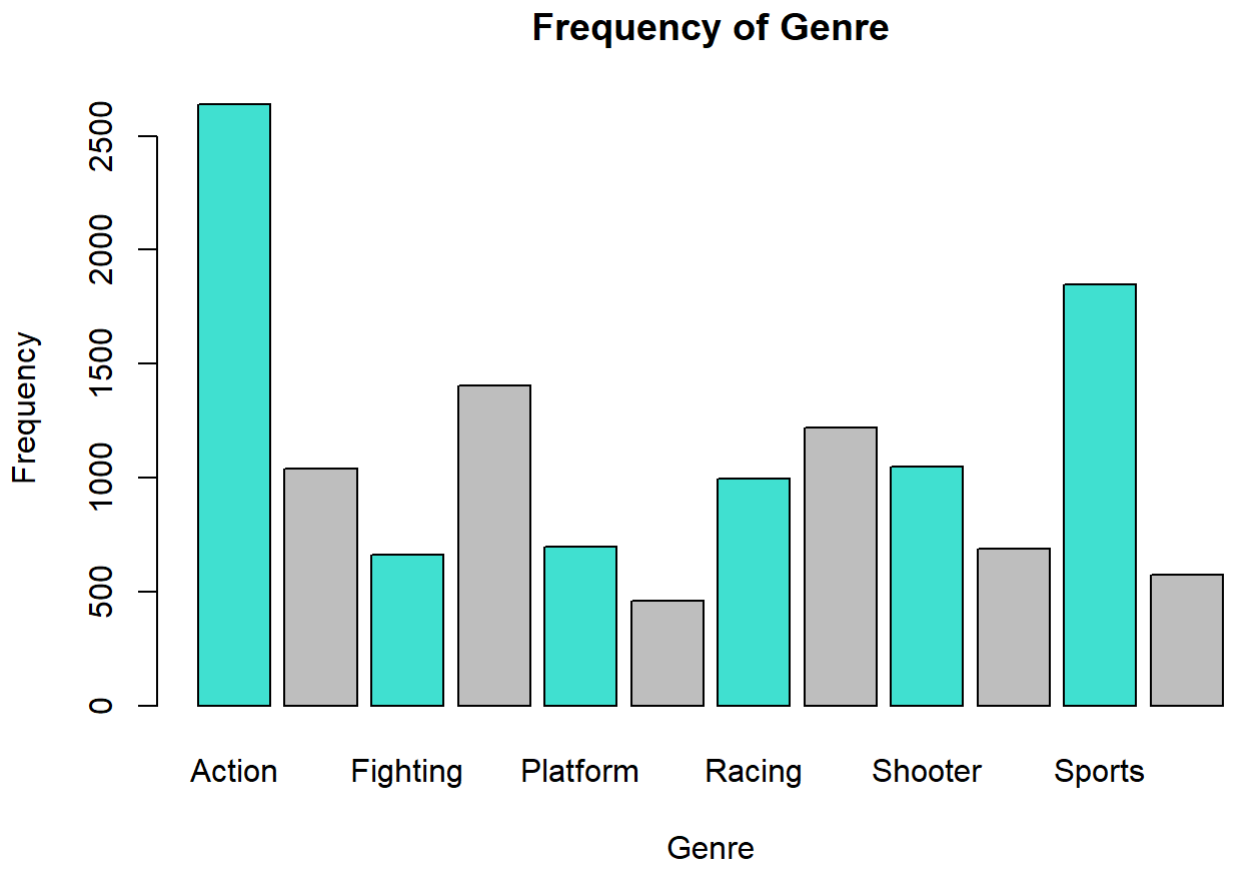
	Rank	Name	Platform	Y…	Genre
	<int>	<chr>	<fct>	<chr>	<fct>
6242	6243	Tak and the Guardians of Gross	PS2	2008	Action
15562	15564	Lovely x Cation 1 & 2	PSV	2015	Action
15244	15246	GunParade Orchestra: Midori no Shou	PS2	2006	Adventure
4609	4610	Vanquish	X360	2010	Shooter
1744	1745	Final Fantasy Tactics: The War of the Lions	PSP	2007	Role-Playing
15463	15465	Crysis: Warhead	PC	2008	Shooter

6 rows | 1-6 of 13 columns

Informative Graphs

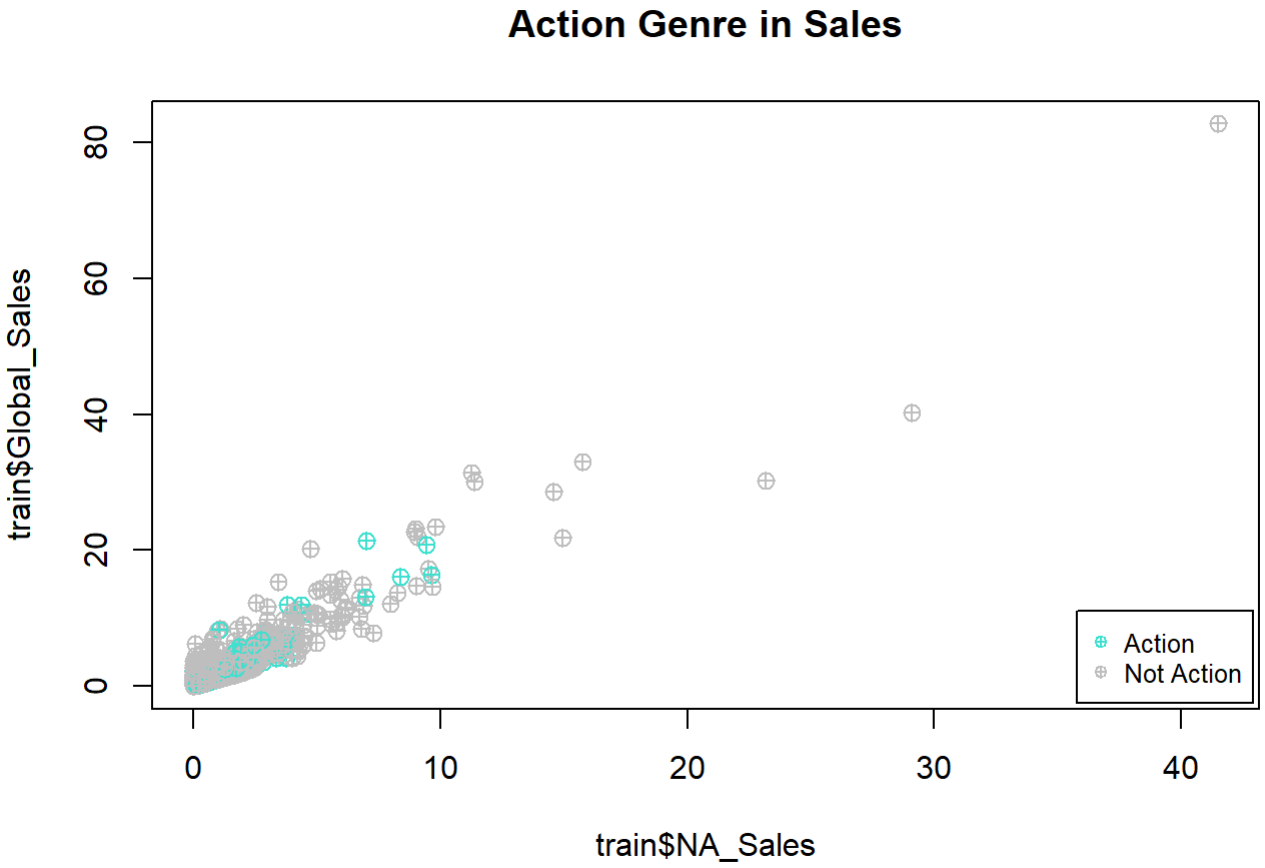
Graphs can be used to visually represent the data in different columns. Below is a **barplot** for Genre. It shows that Action is by far the most frequent genre, with Sports coming in second and every other genre being in more or less a similar range, matching the information from the summary.

```
counts <- table(train$Genre)
barplot(counts, xlab="Genre", ylab="Frequency",main="Frequency of Genre",
col=c("turquoise","gray"))
```



The second graph is a **scatter plot** of isAction on NA_Sales against Global Sales. It shows how the action genre is largely intermixed with other genres in terms of their sales in North America versus globally. These 2 sales alone may not be enough predictors for the Action genre.

```
plot(train$NA_Sales, train$Global_Sales, pch=10, cex=1.2,main="Action Genre in Sales", col=c("gr
ay","turquoise")[unclass(train$isAction)])
      #x=32,y=18
legend(x=35.8,y=11.1,c("Action", "Not Action"),cex=.8,col=c("turquoise","gray"),pch=10)
```



Logistic Regression Model Summary

Now that a binary factor has been established and the data has been explored and visualized, a logistic regression model can be created. This model uses all regional sales columns as predictors on isAction, the target. The overloaded summary() function can then be used to output a **summary** of the model's metrics.

```
glm1 <- glm(isAction ~ Global_Sales + NA_Sales + EU_Sales + JP_Sales, data = train, family = "bi
nomial")
summary(glm1)
```

```
##
## Call:
## glm(formula = isAction ~ Global_Sales + NA_Sales + EU_Sales +
##     JP_Sales, family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.5384  -0.6739  -0.6712  -0.6096   2.7971
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.37165    0.02348 -58.419  < 2e-16 ***
## Global_Sales  0.34190    0.15000   2.279   0.0226 *
## NA_Sales     -0.38908    0.17174  -2.266   0.0235 *
## EU_Sales     -0.20142    0.19219  -1.048   0.2946
## JP_Sales     -1.09059    0.20285  -5.376  7.6e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 13243  on 13277  degrees of freedom
## Residual deviance: 13188  on 13273  degrees of freedom
## AIC: 13198
##
## Number of Fisher Scoring iterations: 5
```

The first part of the summary is the **Deviance Residuals**, which quantifies the deviance contributed by each observation. Lower deviance residual values are better, because they indicate less error in the model's predicted probability versus the actual probability. The deviance residuals in the summary range from -1.5384 to 2.7971.

The second part of the summary is the **Coefficients**. The coefficients represent change in the log odds of y for every 1 unit change in the predictor(s). The intercept and p-values function similarly to their role in linear regression, wherein the intercept is just a fitting parameter and the p-value needs to be less than 0.05 to be within acceptable range. The p-value for JP_Sales appears to be the lowest and the best, with Global_Sales and NA_Sales having the next best p-values, and EU_Sales being too high.

Finally, the third part of the summary is the statistical information on the **Null Deviance** and **Residual Deviance**. The null deviance is the measure of deviance based only on the intercept, while the residual deviance is the measure of deviance for the entire model. Ideally, a decrease would be noticed between the null deviance and residual deviance. There is a small but distinct decrease in the summary, from a null deviance of 13243 to a residual deviance of 13188. This is alright, but not very good. The AIC score is very high, indicating that this model is undesirably complex.

Naive Bayes Model Summary

Naive Bayes is another classification algorithm that calculates the probability of the positive class for each data point, which is TRUE for Action in this case. This algorithm makes the "naive" assumption that each predictor is independent, so that their joint probabilities don't have to be calculated.

```
library(e1071)
nb1 <- naiveBayes(isAction~Global_Sales + NA_Sales + EU_Sales + JP_Sales, data=train)
nb1
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##     FALSE      TRUE
## 0.8012502 0.1987498
##
## Conditional probabilities:
##        Global_Sales
## Y            [,1]     [,2]
##   FALSE 0.5356857 1.640520
##   TRUE  0.5331868 1.208927
##
##        NA_Sales
## Y            [,1]      [,2]
##   FALSE 0.2619579 0.8511531
##   TRUE  0.2659151 0.5832939
##
##        EU_Sales
## Y            [,1]      [,2]
##   FALSE 0.1426882 0.5311668
##   TRUE  0.1575142 0.4212644
##
##        JP_Sales
## Y             [,1]      [,2]
##   FALSE 0.08476549 0.3340830
##   TRUE  0.05196665 0.1777265
```

Printing the model shows all the probabilities that the NB algorithm learned from the data. The prior probabilities of FALSE (not Action) is 0.801 and TRUE (is Action) is 0.1987. The conditional probabilities for each predictor is output next. Since all the predictors are quantitative and represent continuous data, their means and standard deviations are given in the conditional probability tables. For example, the mean global sales of games that weren't action was 0.5356 million dollars, and so forth. Overall, every region's mean difference between sales of non-action and action games is not very high, indicating that there isn't much of a trend in one direction or the other.

Predicting on the Test Data

Predictions can now be made on the test data using the 2 classification models. For logistic regression, the predict() function uses the "response" parameter to output a vector of probabilities between 0 and 1 on the test data. An if-else statement is used to classify probabilities greater than 0.5 as 2 (TRUE for isAction) and less than 0.5 as 1 (FALSE for isAction). The accuracy is calculated as the mean of correct predictions matching the actual values for isAction in the test data. Finally, a table showing the frequencies of predicted values is output.

```
probs1 <- predict(glm1, newdata=test, type="response")
pred1 <- ifelse(probs1>0.5,2,1)
acc1 <- mean(pred1==as.integer(test$isAction))
err1 <- 1-acc1
print(paste("The accuracy of glm1 is",acc1))
```

```
## [1] "The accuracy of glm1 is 0.796084337349398"
```

```
print(paste("The error rate of glm1 is",err1))
```

```
## [1] "The error rate of glm1 is 0.203915662650602"
```

```
table(pred1,as.integer(test$isAction))
```

```
##
## pred1    1    2
##     1 2643  677
```

According to these metrics, the logistic regression model did fairly well. Its accuracy is given as 0.796, about 80%, which is not bad. The table shows that 2643 values were predicted as class 1 (not Action) which really were 1, while 677 values were predicted as class 1 but were really class 2 (is Action). These are fairly good numbers for such a large dataset. However, the table output does not match a confusion matrix because e the model predicted all test cases to be in one class. This means there is no false negative or true negative in the table.

For Naive Bayes, the predict() function uses the “class” parameter to predict on the test data. The resultant probabilities are factor values with 2 levels, TRUE and FALSE. These values are used to output the confusion matrix, which shows 627 true negatives, 543 true positives, 2016 false positives, and 134 false negatives. Using these values, the specificity and sensitivity of nb1 can be calculated. The sensitity, the true positive rate, is 0.21, and the specificity, the true negative rate, is 0.82.

```
probs2 <- predict(nb1, newdata=test, type="class")
acc2 <- mean(probs2==test$isAction)
print(paste("The mean accuracy of nb1 is",acc2))
```

```
## [1] "The mean accuracy of nb1 is 0.352409638554217"
```

```
t <- table(probs2, test$isAction)
tp <- t[4]
fp <- t[3]
fn <- t[2]
tn <- t[1]
sens <- tp/(tp+fn)
spec <- tn/(tn+fp)
print(paste("The sensitivity of nb1 is",sens,"and the specificity of nb1 is",spec))
```

```
## [1] "The sensitivity of nb1 is 0.212192262602579 and the specificity of nb1 is 0.823915900131
406"
```

The results indicate that the logistic regression model was more effective at predicting on the test data. This may be attributed to the fact that logistic regression does its parameter estimations and probability calculations of *y* given *x* directly, while Naive Bayes calculates how the data was generated. Perhaps the predictors were more dependent on each other such that Naive Bayes' naive assumption couldn't apply. Furthermore, logistic regression tends to perform better on larger datasets than Naive Bayes, which may be the case here since this was a large dataset.

Strengths and Weaknesses of NB vs Logistic Regression

The main difference between Naive Bayes and logistic regression is that logistic regression is a discriminative classifier, while Naive Bayes is a generative classifier. This means that Naive Bayes estimates parameters for likelihood P(X|Y) and prior P(Y), while logistic regression directly estimates the parameters for posterior P(Y|X). Naive Bayes and logistic regression are both high-bias low-variance algorithms, but Naive Bayes is more so. As stated earlier, Naive Bayes tends to perform better on datasets of smaller sizes. Naive Bayes performs fairly well despite its naive assumption, but for larger datasets, that assumption needs to hold in order for it to perform as well as logistic regression would. Logistic regression also entails an iterative optimization process, while Naive Bayes only does one pass over the data. This may work well for multiple factor predictors that each have multiple levels, since Naive Bayes can quickly extract the probabilities while logistic regression has to make multiple passes over the data.

Classification Metrics

A number of metrics can be used to evaluate classification models. Accuracy is calculated as the number of correct predictions divided by the number of observations, or the mean of correct predictions. This metric can also be calculated from the confusion matrix, where (true positives+true negatives) / (all) is also equal to the accuracy. The error rate, in turn, is equal to (1 - the accuracy). Two additional metrics gotten from the confusion matrix are sensivity and specificity, which measure the true positive rate and true negative rate respectively. These 2 metrics quantify how much data belonging to a certain class was misclassified. Another metric is the kappa value, which adjusts accuracy on the possibility that data could have been classified correctly by chance, taking the distribution of classes into consideration. Kappa values closer to 1 are ideal. Finally, the ROC curve depicts the tradeoff between predicting true positives and avoiding false positives, with the x-axis representing the false positive rate and the y-axis representing the true positive rate. Ideally this rate would go up quickly and then stagnate, instead of remaining equal throughout. The ROC curve produces a related metric called the AUC, or area under the curve.