Applied Statistical Learning

Final Project Report

Jeremy Prasad

Himani Patel

Prisha Bhamre

Rutgers University

May 5, 2023

ABSTRACT

This report is being submitted as our final project for Statistics 486 - Applied Statistical Learning. In it, we examine a large dataset of movie reviews, and apply things we've learned in and out of class such as data cleaning, support vector machines, and naïve bayes. We also explore several techniques such as predictive modeling and natural language processing to better understand text data and how it relates to statistical learning.

Keywords: Statistical Learning, NLP, SVM, Naive Bayes

BACKGROUND

I. About the Data

The data we decided to use came from Kaggle, a popular data-centric website. The data we were analyzing contained information about movie ratings and reviews that was scraped from RottenTomatoes, a website where users and critics can rate movies. Our data set is actually comprised of two files: (1) a CSV file containing the reviews and other data related to the reviews, and (2) another CSV file containing information about each movie. The file containing information about the reviews had 1,444,773 rows and 11 columns, including `movie_id`, `reviewText`, `originalScore`, `isTopCritic`, and `scoreSentiment`. The other file, which contained information about each unique movie, had 143,258 rows and 16 columns, including `movie_id`, `title`, `audienceScore`, and `genre`.

II. Cleaning the Data

To work with our dataset, we first had to clean the data. Cleaning data is an important part of data analysis because data is not collected perfectly. Data can be incomplete, inaccurate, or incorrectly formatted. It is difficult to apply functions and correctly analyze data that has not been cleaned. In our dataset, the main columns we worked with were `reviewText`, `originalScore`, `isTopCritic`, and `scoreSentiment`, as well as `genre`. To clean the dataset, we first removed all null or missing values from the dataset.

We understand that this is not the standard practice for all datasets, but because the data set is so large, we still had hundreds of thousands of rows to work with that was still representative. For the genre column, we extracted only the first genre that appeared in the list of genres since some movies had multiple genres. For the review text column, we decided to analyze the sentiment of each review. To do this, we calculated the polarity of each review where more negative values meant negative sentiment and more positive values meant positive sentiment. For the original score column, we realized that the score of a movie was saved as a string, and there were multiple different ways scores were communicated. Some scores were saved on a number scale, such as "4/10", "3/5", or "60/100", while others were on a letter scale such as "A", "B", or "F". We wanted to consolidate all of these scores into one numeric scale so we could properly compare them. Using the `fix_score` function we developed, we put all of these scores on a scale of 0 to 1. Any score that was negative or greater than 1 was considered invalid.

III. Splitting the Data

After cleaning the data, we merged the two files together using an inner-join on the `movie_id` variable. This allowed us to have one large data frame on which we could work. Since working with over a million records was not ideal for our computing capabilities (and, in fact, killed our computers), we randomly selected 10,000 records with no null values to work with. For the study of polarity and score, we used a 90-10 training-test split (9,000 training and 1,000 testing). For the rest of the project, 8,000 of records were used as training data, and the remaining 2,000 were used for testing and validation.

METHODOLOGY

I. Sentiment Analysis

Companies are now using sentiment analysis more than ever to learn more about customer experiences and sentiment towards brands. Sentiment analysis is the process of analyzing text to determine the emotional tone behind it. Learning about the emotion behind the text helps analysts categorize the text in different ways, whether that be positive or negative sentiment, a numerical polarity value, the most prevalent emotion, etc.

The most basic way of finding the sentiment of a sentence or a group of text is by determining the sentiment of each particular word. To do this, first, punctuation must be removed from the text in order to properly analyze it. Next, each word is compared to an existing sentiment dictionary and given a polarity. However, not every word has a polarity. For example, the word ‘magnet’ has no sentiment. Therefore, its polarity would be 0. A positive word like ‘good’ would have a positive polarity score, while a negative word like ‘bad’ would have a negative polarity score. However, not every tense or form of a word occurs in these dictionaries. Therefore, we must first lemmatize each word which means we must look at the root word of every word in the sentence to find its polarity through a dictionary. For example, the word ‘crying’ would probably not be present in a sentiment dictionary, but ‘cry’ would. After lemmatizing the words, they can be found in the dictionary and given a polarity. From there, we can calculate the polarity of the text body by finding the average of the polarities of all of the words. However, this way of calculating the polarities is relatively inaccurate. This is because words don't exist separate from other words. For example, if your sentence was ‘This is not good’, this way of calculating sentiments would declare the sentence to be of positive sentiment because good has a positive sentiment and all of the other words alone are neutral. However, if we combine the words and analyze them as phrases, we would declare this sentence to be of negative sentiment.

To mitigate this issue, instead of using single words, we can use *n*-grams. In this approach, we use 1-grams (“mono-grams”) which are single words and 2-grams (“bi-grams”/two words) to determine the polarities of each text body. By using 2-grams, we can consider the effect of valence shifters which alter or intensify the meaning or other words. This leads to overall polarities that are more accurate. Using our previous example, with this approach, ‘This is not good’ would correctly be identified as negative sentiment.

II. The Theory Behind Support Vector Machines

One common type of problem in machine learning is binary classification. Consider the case of a dataset characterized by $\{(\vec{x}_i,\ y_i)\}_{i=1,\ldots,N}$ where $\vec{x}_i$ is a (row) vector denoting the predictor variables of the i^{th} data point/row. And suppose that $y_i \in \{0,\ 1\}$. It is often of interest to find some sort of separator such that depending on the values in $\vec{x}_i$, we can predict, or *classify*, the value of y_i. One such example is classifying reviews as positive or negative, as is the case with our dataset.

The idea behind Support Vector Machines (SVM) aims to provide a solution to this problem. In essence, the SVM finds a separator such that everything on one side of the divider is classified as one class, and everything on the other side will be classified as another class. There are some variations on the classic SVM solution, though, including the ability to limit it to either a linear divider or a non-linear divider, as well as the ability to allow misclassifications if no perfect divider exists. In the case of a linear divider, the separator would be a hyperplane, and for non-linear separators, it would be a manifold. Additionally, allowing misclassifications is known as a “soft margin,” whereas perfect classification would be labeled “hard margin.” In soft margin SVM, the tuning parameter is C, which determines the tradeoff between the margin and the training error.

SVMs do not just find a separator, however. One could argue that there are many such dividers depending on the data. But no matter which version of SVM is being used, all try to find the *best* possible separator. Intuitively, we can understand this as the separator that is farthest away from both classes, i.e., the separator with the greatest margin. While the premise of SVM is similar to logistic regression, it differs in the role each data point plays. In logistic regression, each data point will somehow influence the fit of the sigmoid function (granted, some may have a minuscule effect, but an effect nonetheless). However, in SVMs, only a handful of data points actually influence the fit of the divider. This is because the margin is essentially confined to the data points of different classes that are closest to each other. The data points that bound the margin are known as support vectors – and these are the only data points that directly affect the fit of the SVM.

III. Naïve Bayes

Another model commonly used to solve classification problems is Naïve Bayes. It follows a probabilistic approach compared to the SVM's geometric approach. The model assumes that all predictor variables are independent of each other, which is the reason it is called "naïve" because in real world problems, as the assumption isn't always true.

The math utilized by the model to classify data through predictor variables is an application of the Bayes' theorem can be summed up as:

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})} \quad \text{or} \quad Posterior = \frac{Likelihood \times Proposition\ prior\ probability}{Evidence\ prior\ probability}$$

Where:

- C_k is the proposition and $\vec{x}$ is the evidence
- $P(C_k)$ is the prior probability of the proposition
- $P(\vec{x})$ is the prior probability of evidence
- $P(C_k|\vec{x})$ is the posterior
- $P(\vec{x}|C_k)$ is likelihood

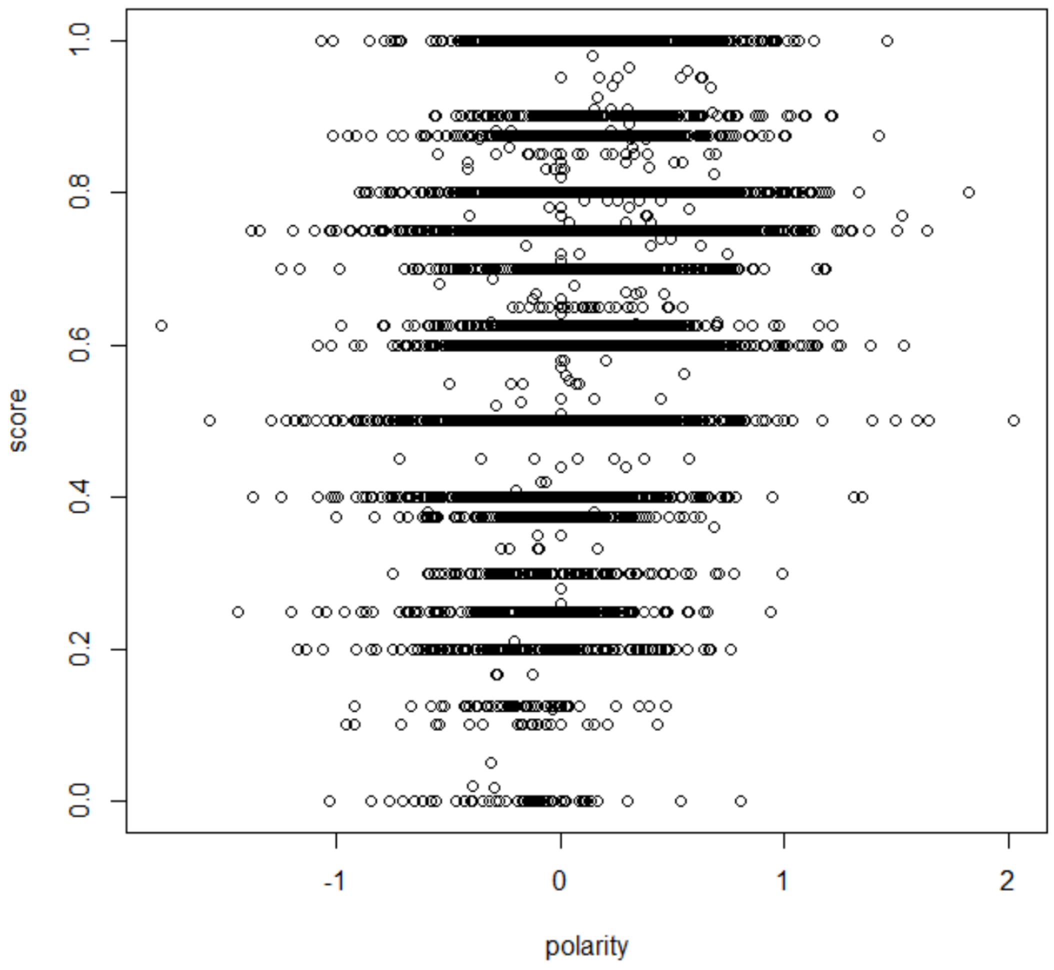
For a problem instance, the model calculates the probabilities for each of the class C_k (i.e., positive or negative) given a set of predictor variables, represented by a vector $\vec{x} = (x_1, \ldots, x_n)$. Then, it selects the class with the highest probability as the predicted outcome. One of the main advantages of Naïve Bayes is its simplicity and efficiency. It is also relatively robust to irrelevant predictor variables.

APPLICATION

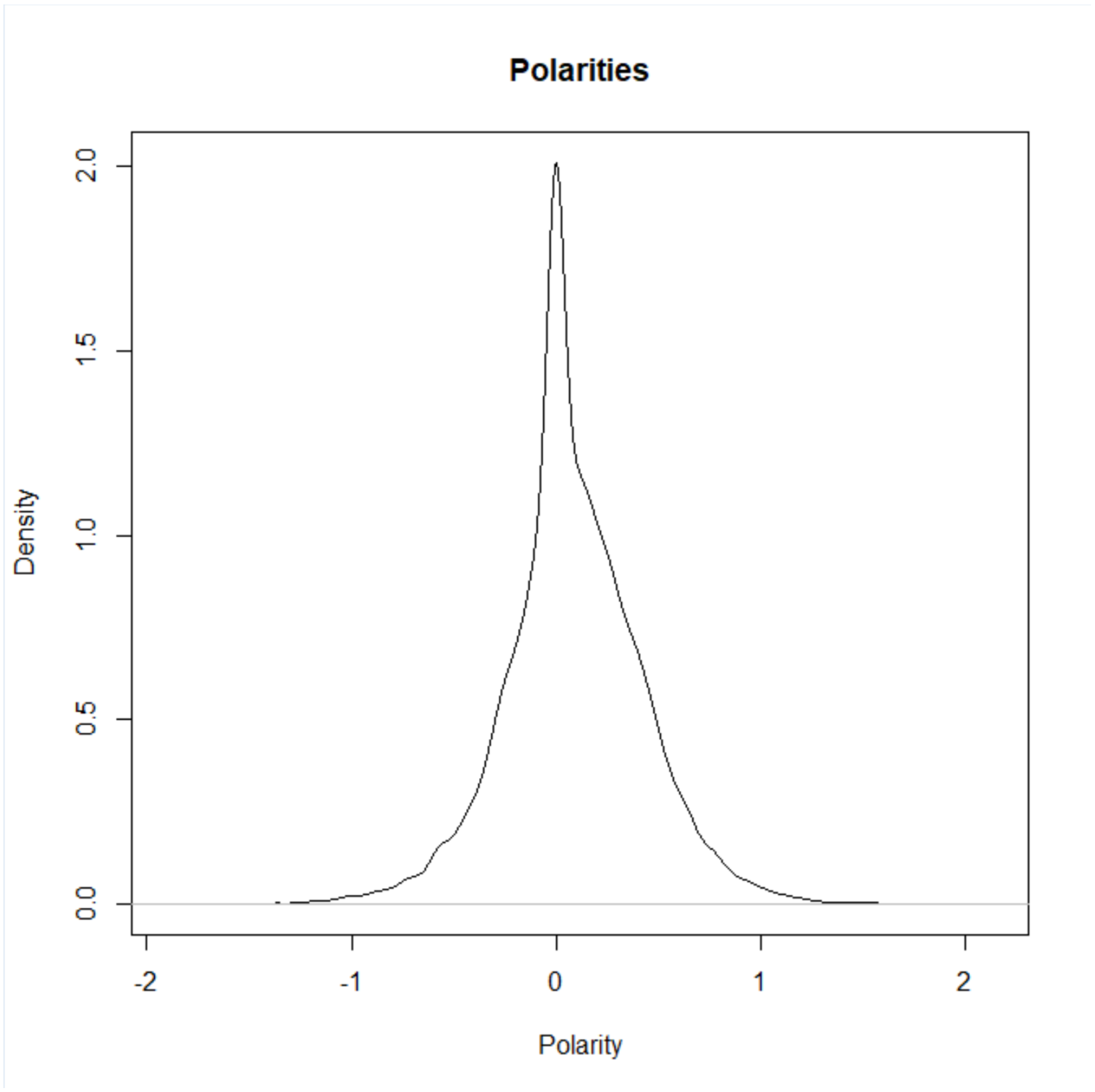
I. How We Used Sentiment Analysis

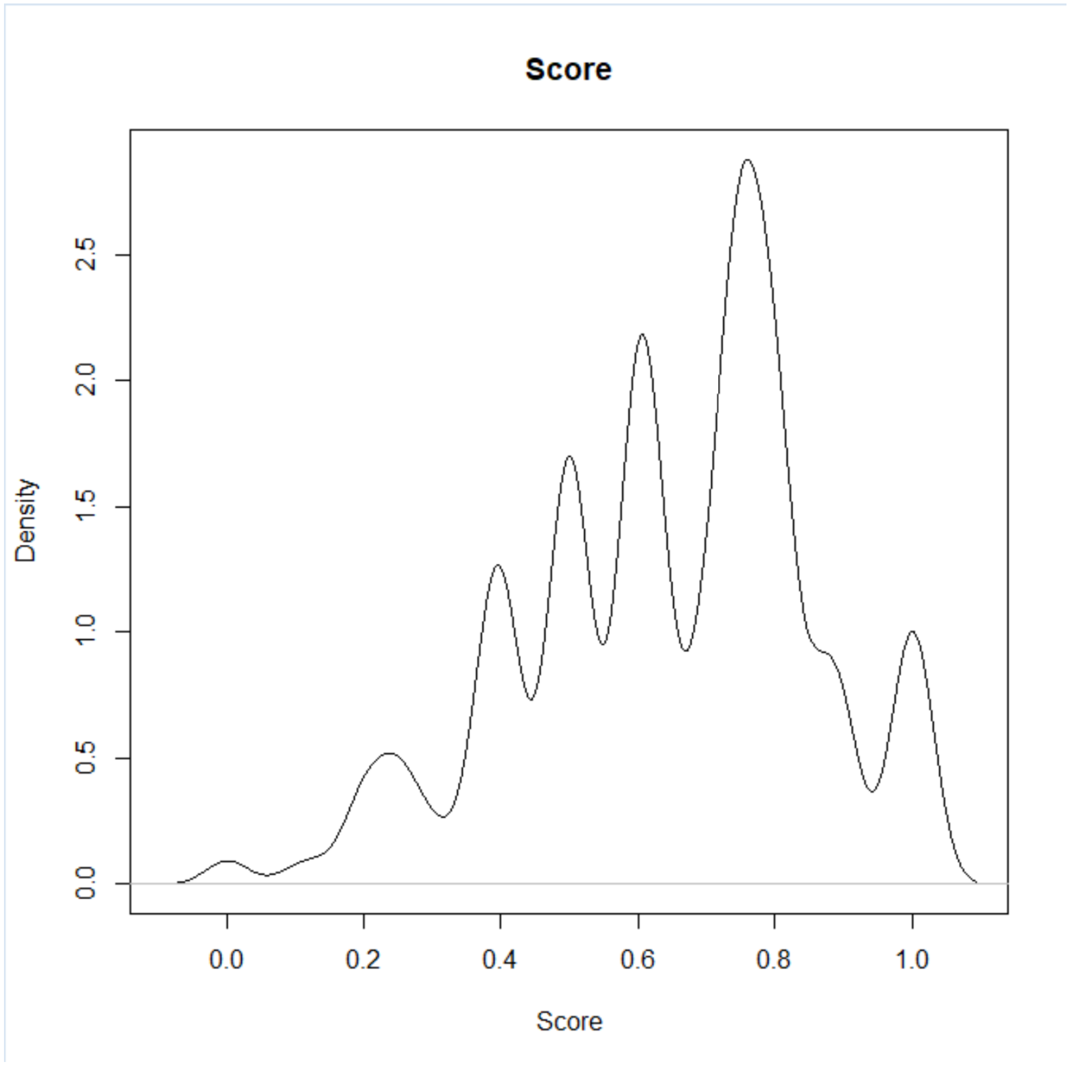
In our study, our dataset already had a column which determined whether a review was of positive or negative sentiment. However, we decided to assign a numerical polarity value to each review as well so we could see how intensely positive or negative a review was. To do this, we used library `sentimentr` to find the polarities of each review. Then, we used the polarity found to determine a numerical relationship between the polarity of a review and the score given by the reviewer. We hypothesized that as the polarity of a review increased, so would the given score.

First, using the `sentimentr` library, we found the sentiment of each review. To do this, we used the `reviewText` column of the dataset. We split each review into its sentences, found the average sentiment of each sentence, and then found the average sentiment of the entire review using the *n*-grams approach. Doing this allowed us to assign a numerical polarity value to each review. We then plotted the polarity we found against the score given by the reviewer.

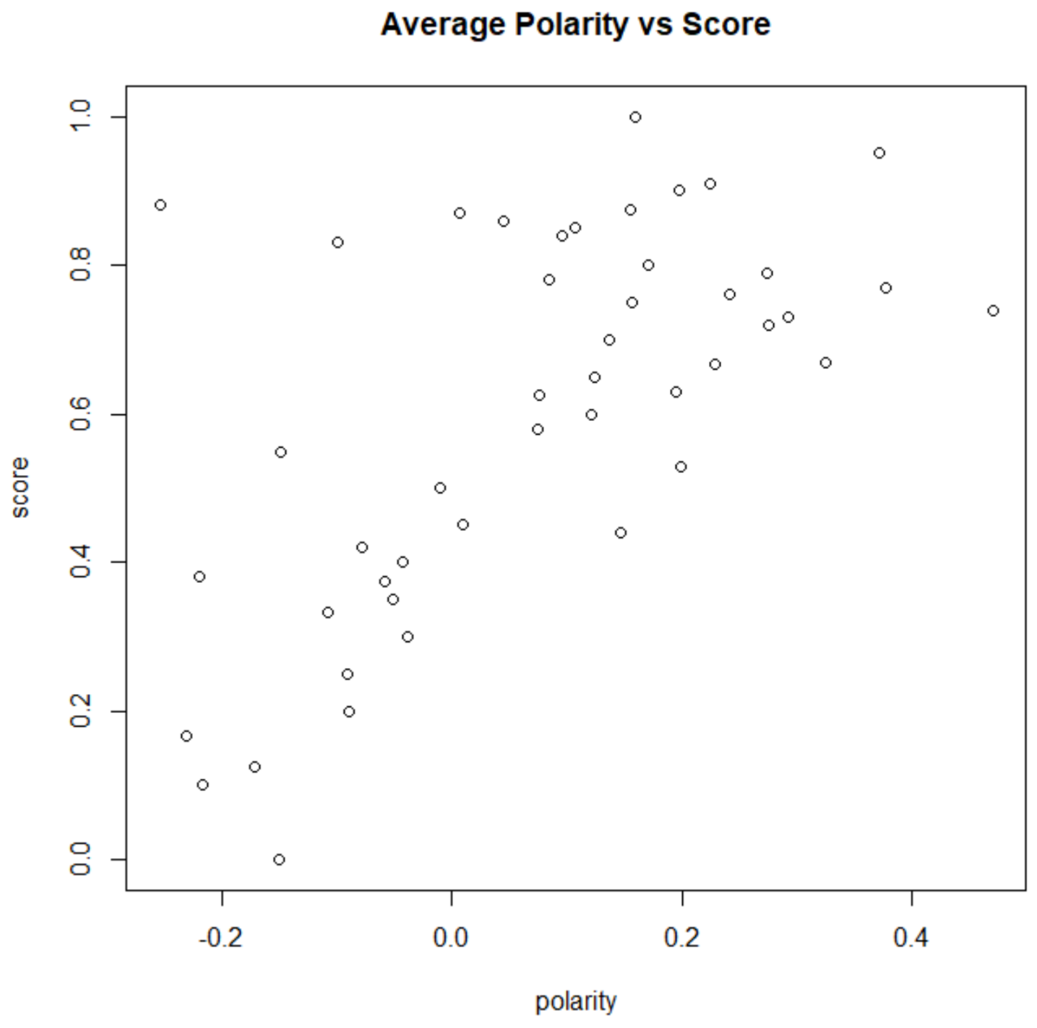


From this graph, we noticed that there was an upward trend between polarity and score, but not at the rate we thought it would be. This is because human variation had a very large impact on our hypothesis. As you can see in the density plots below, most polarities occur right around 0 because it is the most neutral and most scores occur after 0.5. This is because normally, when people score something on a scale, they tend to stay towards the middle or the more positive side because they don’t want to be too harsh or make people feel bad with their opinions.

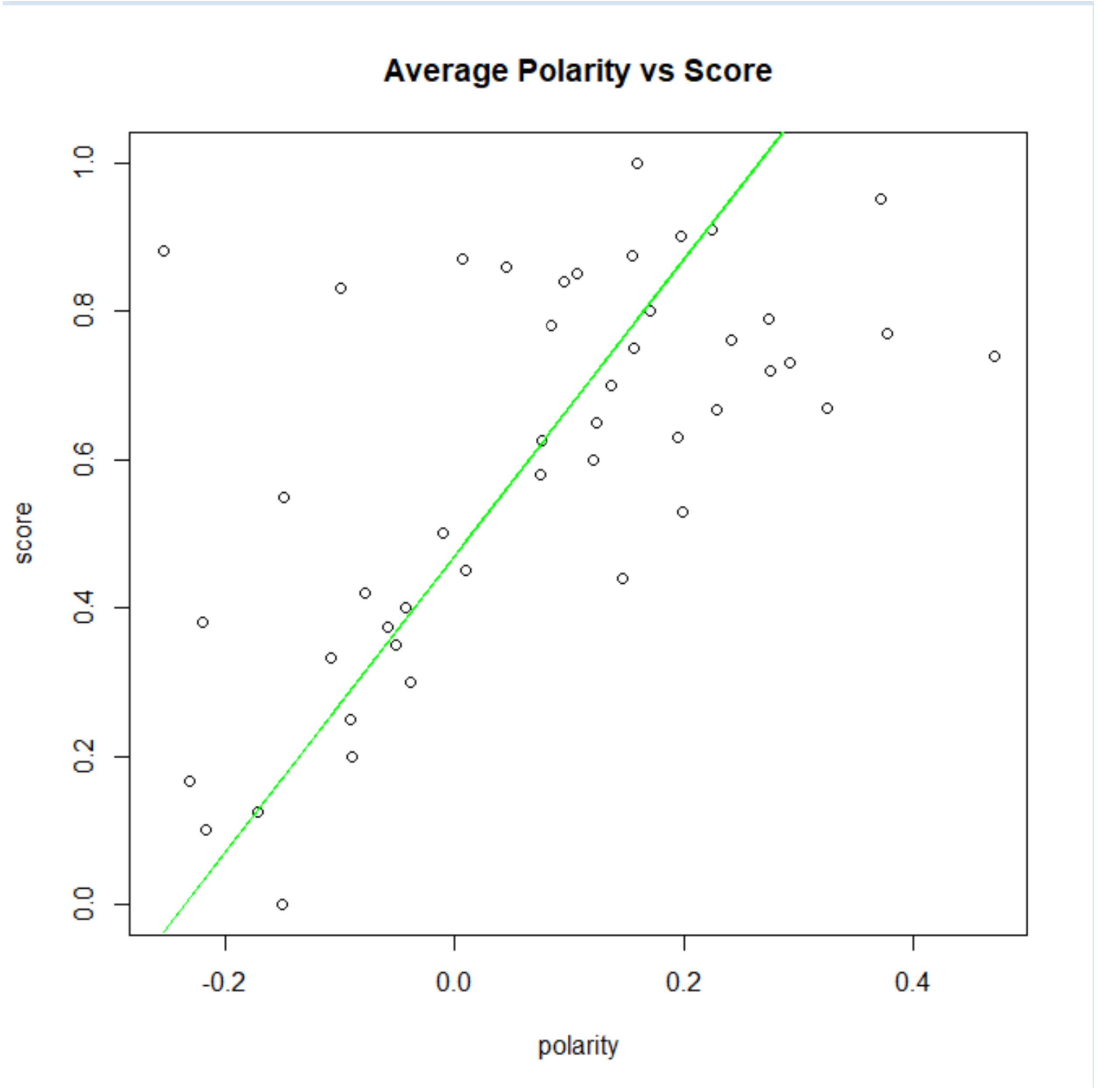




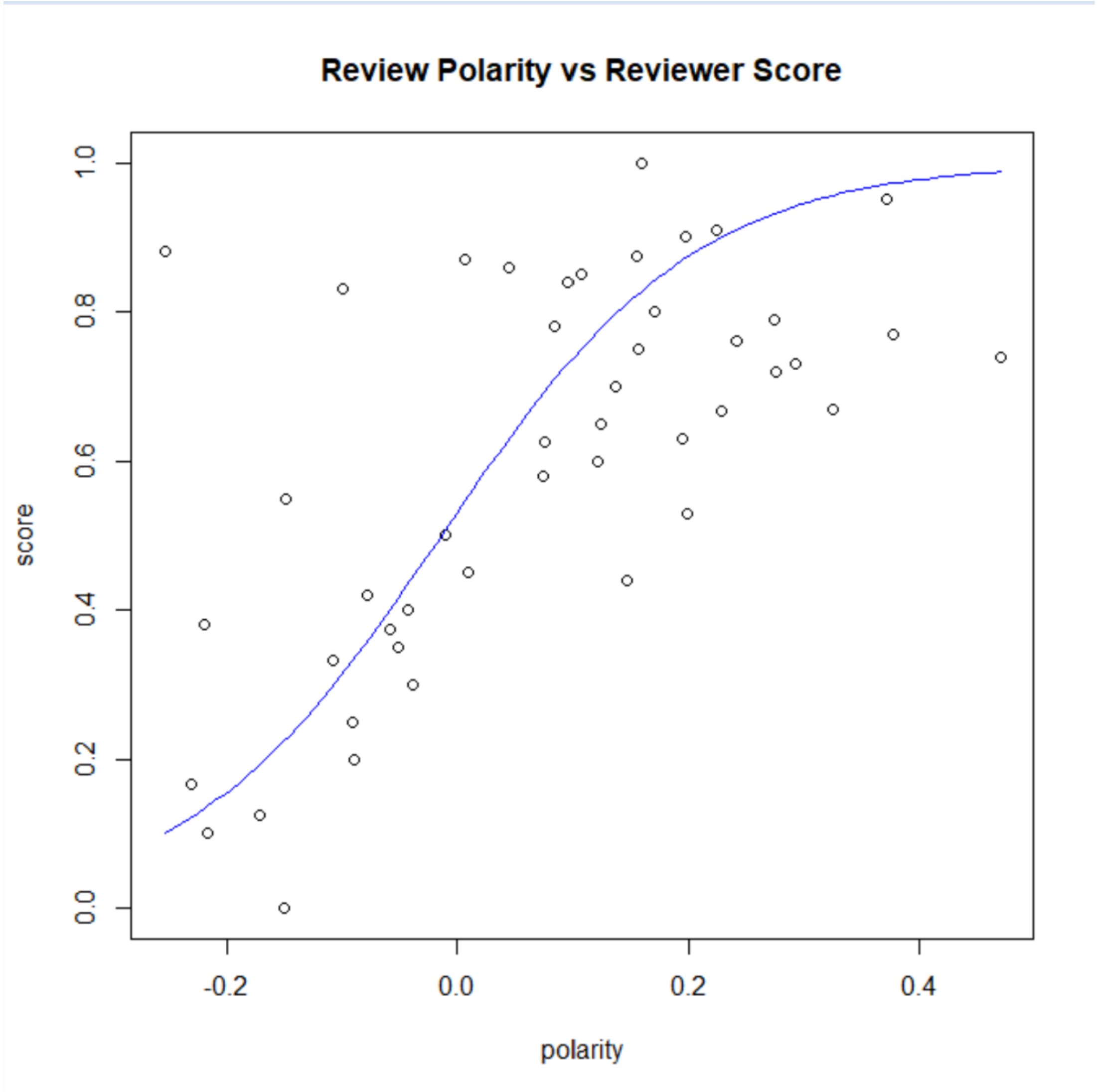
However, we thought that if we reduced some of the variation in the plot, we may be able to find a predictive model. To reduce the variation, we decided to average all of the polarities of a certain score to find one polarity to contribute to each score. Then, we created a `create_points` function. This is because after we grouped each score and averaged them, we were left with vectors that only had one polarity per one score. If we tried to find a model based on this, it would be inaccurate because scores that only had less than five reviews would be of the same value to the model as scores that had thousands of reviews. By using the `create_points` function, we were able to make sure the density of each value was still taken into account in our model.



After we graphed our points, we noticed that there was a positive relationship between polarity and score in our model. To look at the relationship, we used two different models. The first one we explored was a linear model. We found the model `y=0.470173 + 1.994591*x` best fit our data with an R^2 of 0.8234.



Since we were predicting a continuous variable between 0 and 1, we also decided to use logistic regression. To do this, we used the `glm()` function with a logit link function. The `glm()` function is used to make relationships that are non-linear linear by relating the response variable to some link function instead of directly to the explanatory variable so that linear regression can be used. For our study, we used the logit link function or our log odds where $logit(p) = ln(\frac{p}{1-p})$. From the model, we found the log odds were `0.1253133 + 9.0645165x`. However, this function gives us the linear representation. In order to find the predicted values of the nonlinear representation, we must transform this function into the inverse of the link function. The inverse of our link function is $logit(\alpha) = \frac{1}{1+exp(-\alpha)}$. Therefore, our function was $y = \frac{1}{1+exp(-(0.1253133+9.0645165x)}$. The AIC of this model was 6020.8.

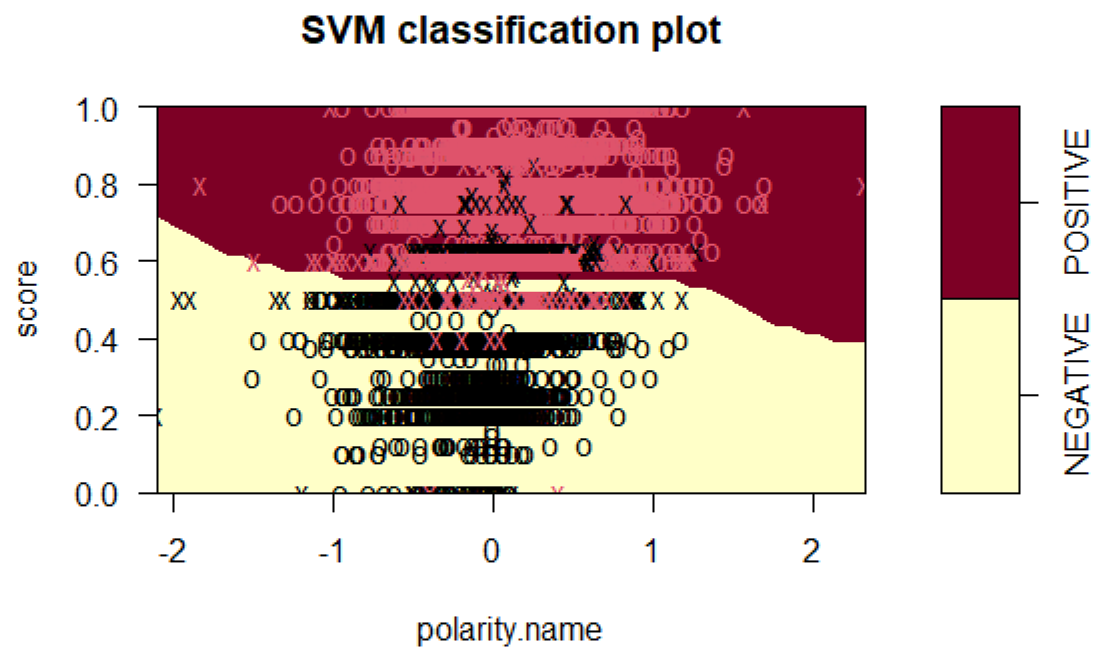


After we plotted these two functions, we wanted to see how well they could predict the score of a review given a certain polarity. To do this, we applied both of these functions to the test data to find the predicted values. We considered them accurate if they fell within 0.1 or 0.05 of the actual score. With the linear model, we found that it was 23% accurate within 0.1 of the score and 7% accurate within 0.05 of the score. With the logistic model, we found that it was 20% accurate within 0.1 of the score and 9% accurate within 0.05 of the score.

These models are based on the averages of the polarities for each score, not actual polarities. Therefore, the low accuracy of our models is reasonable.

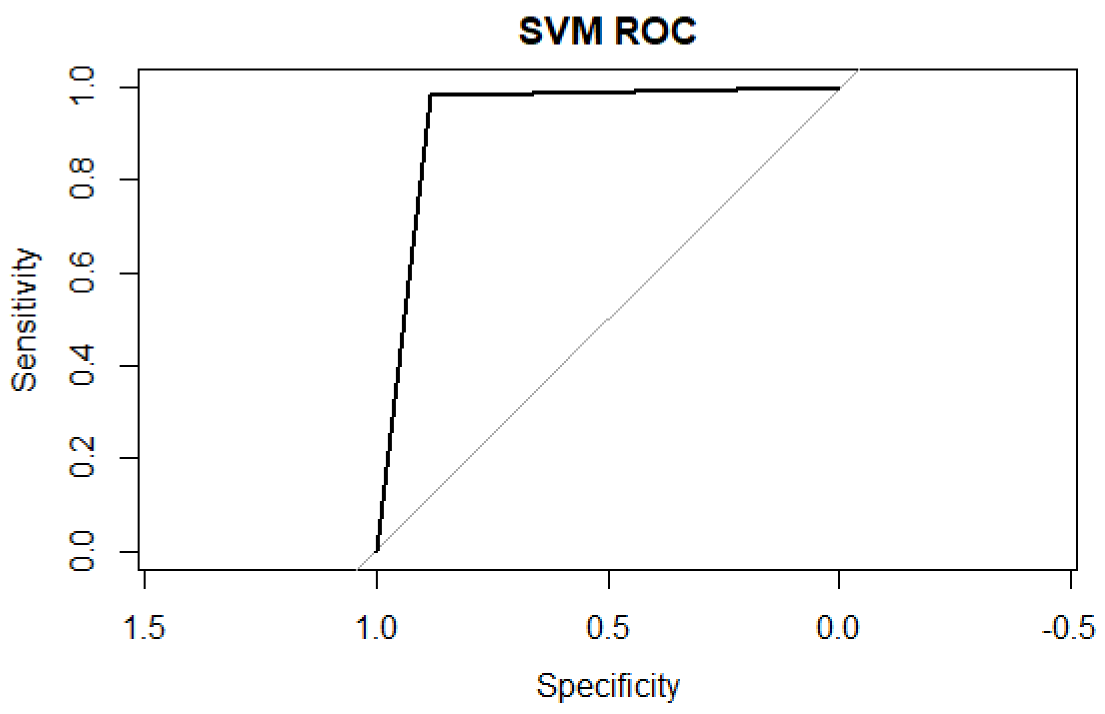
II. Classification Using SVM, Naïves Bayes & Logit

In the context of this dataset, we were curious how a review could be classified using an SVM. One thing we noticed was that the data already included a column called `scoreSentiment`, which was a binary value: either "positive" or "negative." We wanted to see if we could classify a given review as positive or negative, and thought the two most relevant variables for an SVM would be `score` and `polarity` with `scoreSentiment` being our response variable (*note:* here, `score` is the score that the reviewer/critic gave it, not the overall Audience score). We hypothesized that those with a lower score and lower polarity should be classified as a negative review.



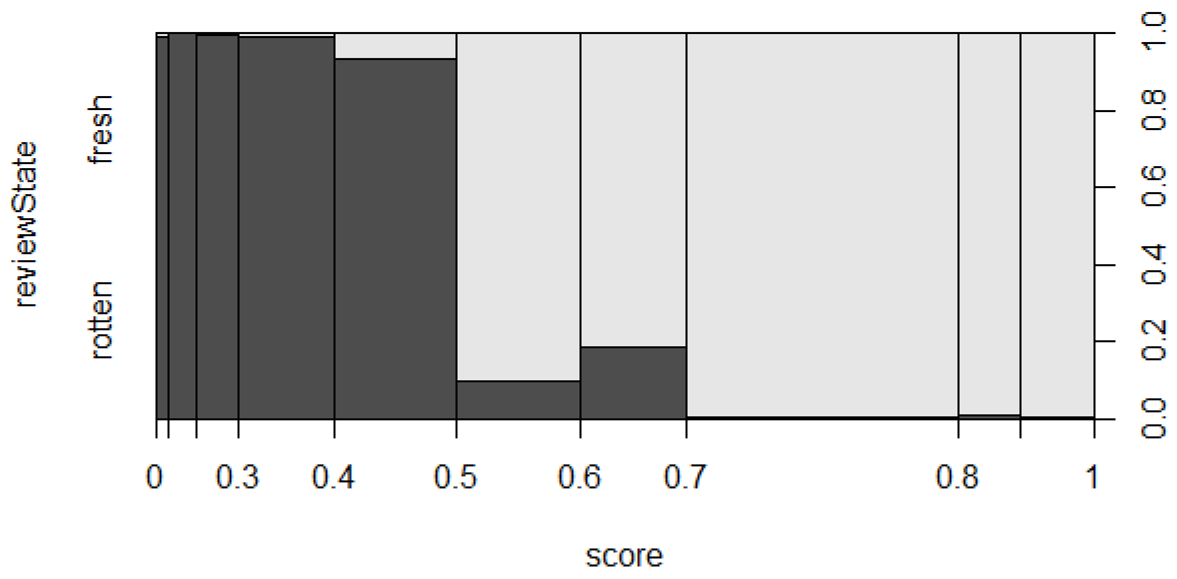
Result from running soft margin SVM. "X" is a support vector.

As it turns out, the polarity did not have as big of an influence as we were thinking. However, the score played a very important role in determining whether the review was negative or not. We can see this because the separator is almost linear with a slope close to zero, meaning that the divider is heavily reliant on just the slope variable. We think this may be due to the fact that the intricacies of language is a complex thing to understand, and so, the calculated sentiment may not be 100 percent accurate nor directly match the overall score that a reviewer gave it. However, a poor score will almost certainly result in a negative review. For example, a reviewer may have used sarcasm with many positive-sounding words but actually rated the movie poorly. Still, the SVM does a pretty good job of finding some sort of separation between classes.



The ROC Curve for SVM.

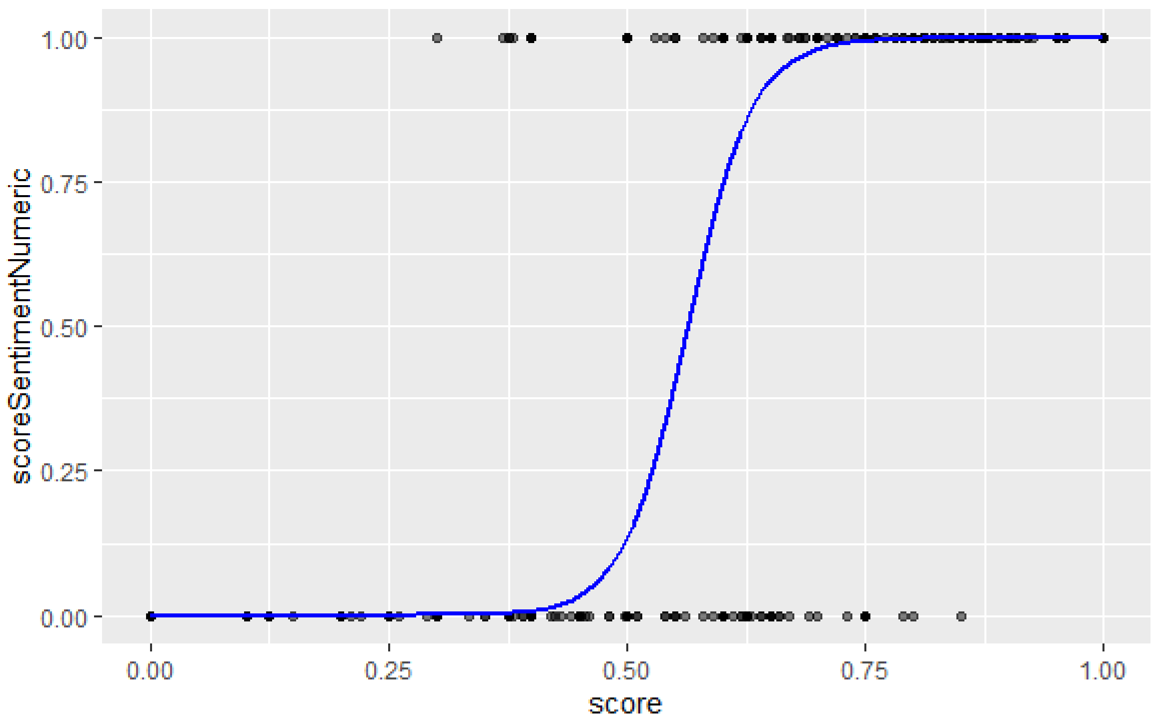
As we can see, the SVM is a very powerful tool and has a great AUC of 93 percent. Regarding its performance, it only has a 5 percent misclassification rate on the testing data.



Ratio of negative to positive reviews given score.

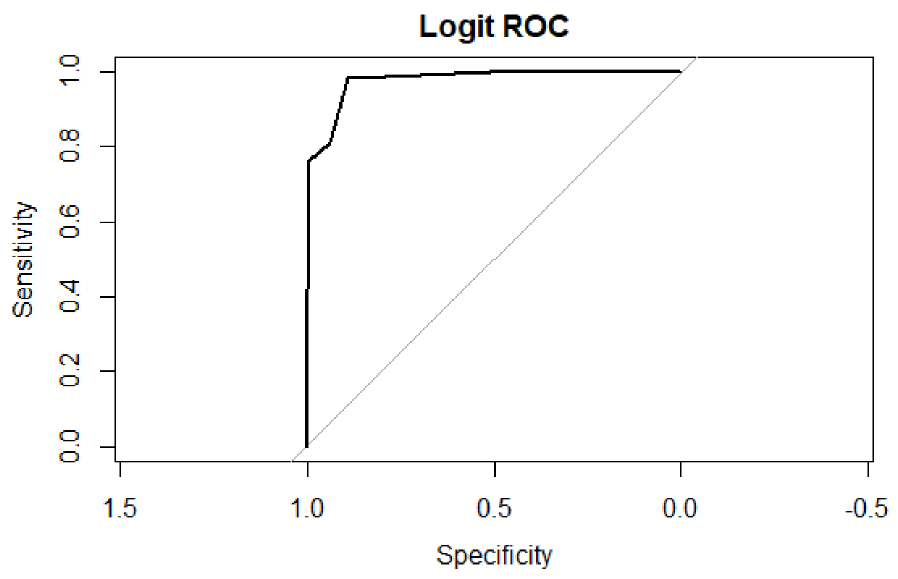
The plot above shows the ratio between "rotten" to "fresh" reviews given the score interval. It seems that overwhelmingly, reviews that had a score of 50 percent or greater were classified as "fresh" and everything else was "rotten" (also known as positive and negative; same thing, different wording). This supports the conclusion that score is a very important factor when classifying whether a review is positive or negative. This is also simpler to understand than an SVM, even though both arrive at the same conclusion.

This leads us to consider applying a logistic regression here. As mentioned before, SVM and logistic regression serve similar functions in binary classification, but differ in how they reach a classification rule. So it's worth taking a look how a logistic regression would perform, especially since there appears to be a clear-cut distinction between score and positive/negative in the figure above.



Logit regression for sentiment on score.

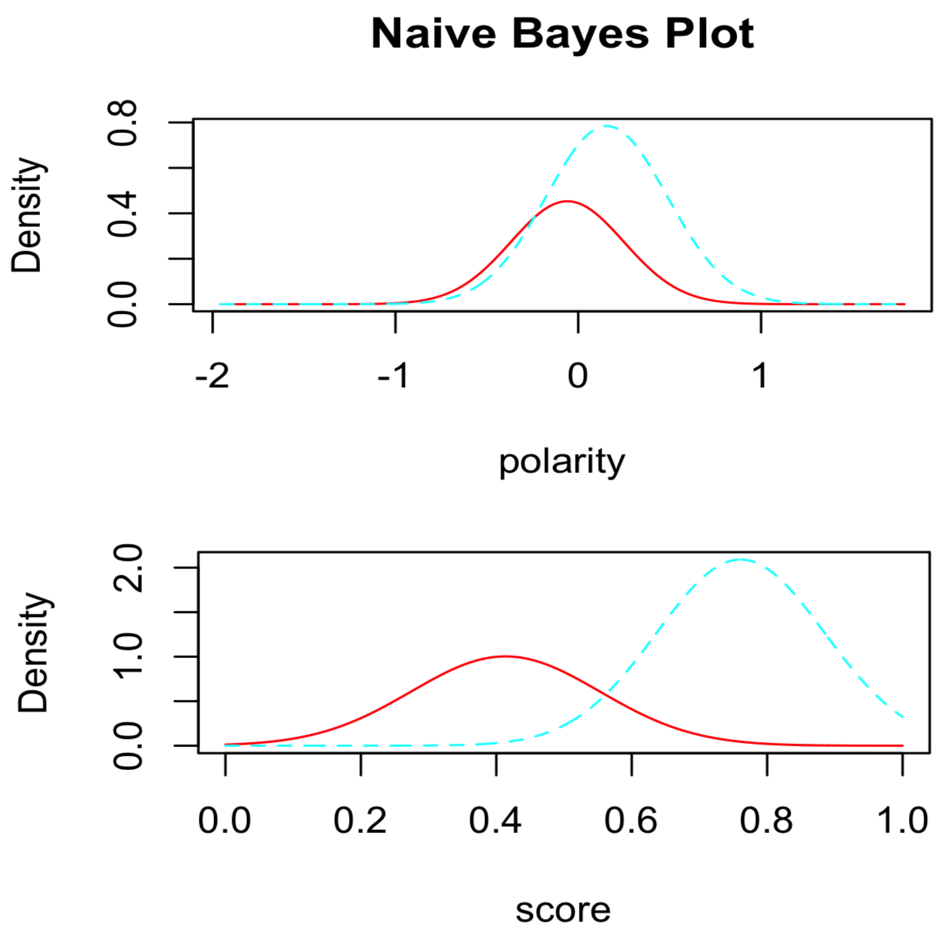
Turns out, this approach is also not bad – and seems to be the perfect use-case for logistic regression. As we can see, the model is much simpler to understand than the SVM. However, simplicity may come at the cost of more misclassifications (around 30 percent); this logistic regression is modeled using one predictor variable, whereas the SVM used both score and polarity. Still, it is nearly as effective as the SVM. This result should not be entirely surprising, as we concluded that the score would be a very good predictor to use for classification. And while polarity did not have much influence in the SVM, it is possible it might influence the logistic regression a bit more. But despite having more misclassifications than with the SVM, this model is also powerful, given its simplicity and the fact that the sigmoid function fits really nicely there.



The Logistic Regression ROC Curve.

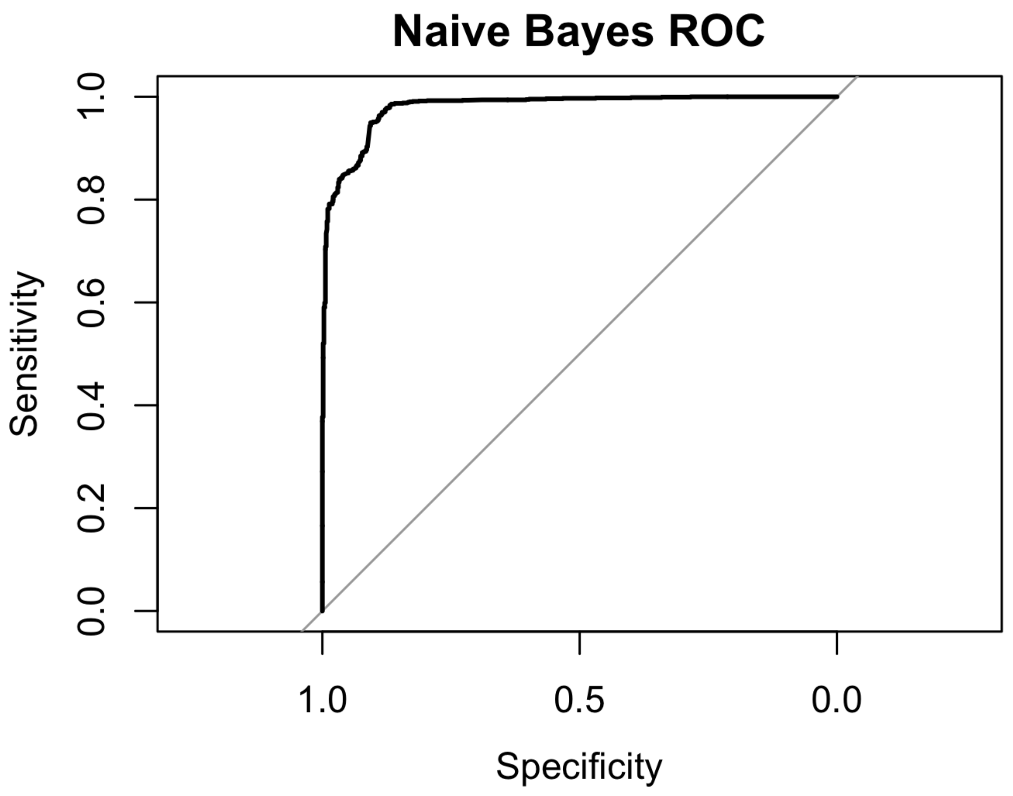
Here we can see that even the logistic regression performs well on the testing data. It has an AUC of 97 percent.

Binary classification can also be completed with Naïve Bayes, which unlike SVM, utilizes conditional probabilities to classify (positive or negative) based on predictors. The following plots show the likelihood curves **<u>(Red = Negative and Blue = Positive).</u>**



Result of classification using Naïve Bayes.

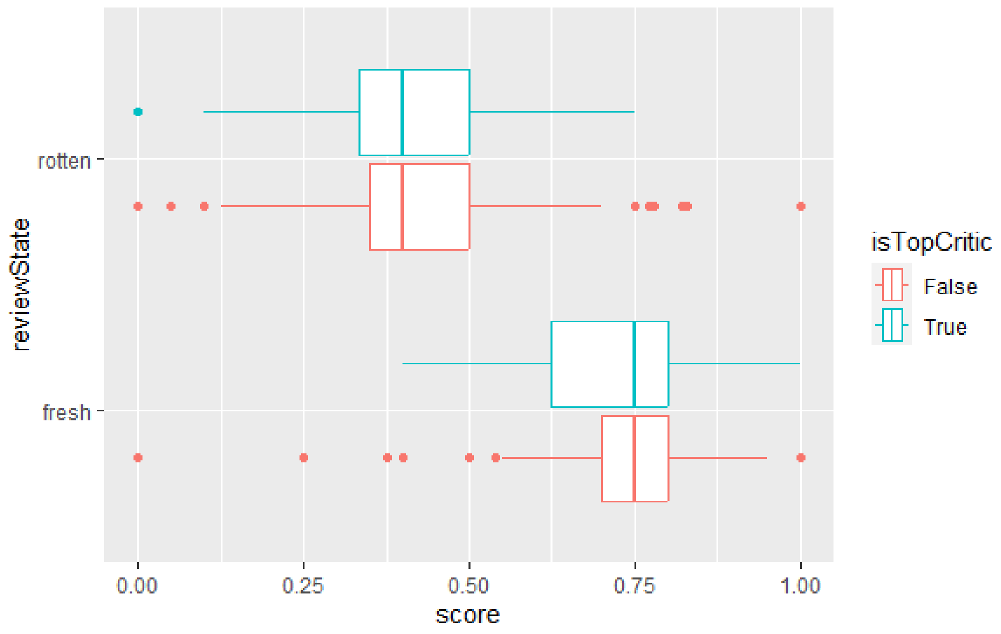
The conclusions that we came to after running the other two classifier models still hold true, score is a good predictor considering that the likelihood curve of the negative scoreSentiment is skewed to left towards lower scores, a low score will likely ascertain a poor review. And then most surprisingly still `polarity` did not prove to be a good predictor of the `scoreSentiment` even though in theory we had hypothesized that it would have a bigger impact and serve as a more useful predictor.

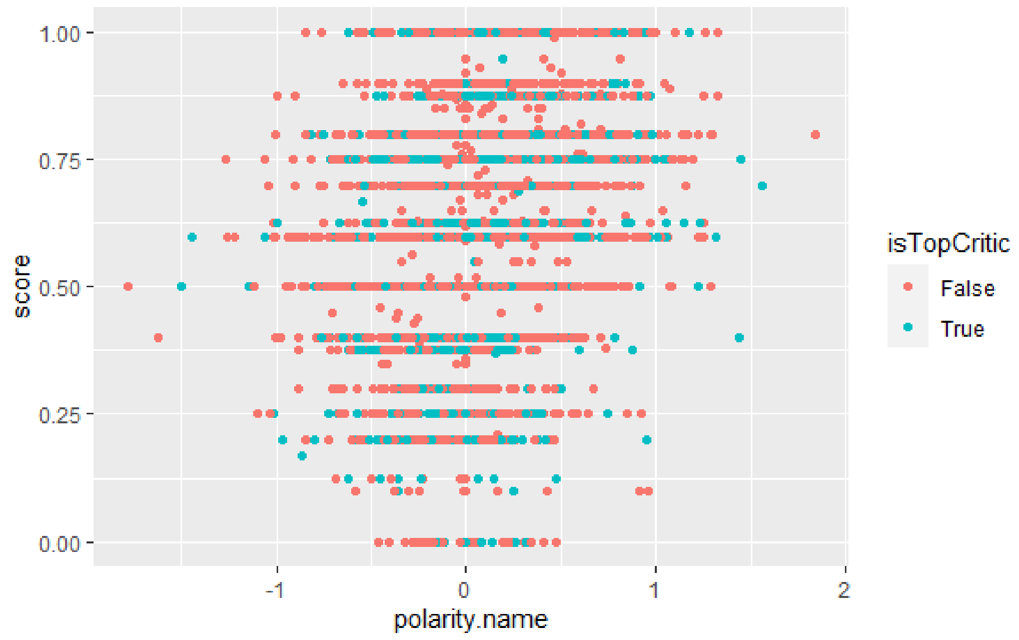


The Naïve Bayes classification model is the best out of all three that we tested because it has the highest AUC of 98% and a misclassification rate of 6%.

III. Top Critics vs. Non-Top Critics

Another thing we wanted to know was whether the type of critic was an important factor to consider in determining whether a review was positive or negative. We thought that top critics would use more polarizing language and give harsher reviews.





Part of that was true. Top critics tended to be more consistent with reviews. For reviews that were positive/"fresh," top critics generally had a higher score, with very little variation away from the rest of the top critics. A handful of non-top critics, however, were often all over the place, with some negative reviews having high scores, and some positive reviews having low scores. Both groups were generally concentrated in the same spot in terms of score grouped by positive/negative reviews.

But it seems that both groups used the same level of polarizing words, which was confirmed with a T-test. The T-test resulted in a p-val of 0.2942 therefore, at a significance level of $\alpha = 0.05$, we do not reject the null hypothesis and we can conclude that there is no significant difference between the mean polarity of top critics and non-top critics. Overall, if you want a trustworthy review, it makes sense to stick with the top critics.

CONCLUSION

In our project, we used several tools to conduct our analyses that we learned about both in and out of class such as logistic regression, SVM, and Naïve Bayes for classification problems, as well as external code libraries that aided in data manipulation, data cleansing, and NLP/Sentiment Analysis.

In the future, we could further explore alternatives to using sentimentr to extract polarity from review. For example, now having utilized SVM and Naive Bayes, we could classify new movie reviews into positive or negative scoreSentiment by extracting NLP features, creating a Document Term matrix and applying SVM and Naive Bayes. (rather than calculating polarity and then applying the classification models)

Overall, we learned a great deal about how to deal with large amounts of text data and how we can use these tools for statistical learning purposes. And while there are many different ways to interpret text data as numerical values – like different word embeddings using bag-of-words or one-hot encoding, for instance – we still ended up addressing important questions we had and have laid the groundwork to complete more complex sentiment analysis projects.

CODE

```
library(dplyr)
library(tidyverse)
library(sentimentr)  # Sentiment Analysis
library(stringr)  # Regular Exp. for Cleaning
library(e1071)  # SVM
library(pROC) # For ROC curve


# function to make all scores given as strings into decimal values.
## Converts the originalScore string to a number.  To be used in apply() fn.
  # Ex.: "5/10" to 0.5
fix_score <- function(x) {

if (grepl("/", x)) {
      split<-strsplit(x, "/")
      score = as.double(split[[1]][1])/as.double(split[[1]][2])
      return(score)
}

if (grepl("F",x)) {
      return(0)
} else if (grepl("D",x)) {
      return(0.25)
}else if (grepl("C",x)) {
      return(0.5)
}else if (grepl("B",x)) {
      return(0.75)
}else if (grepl("A",x)) {
      return(1)
}


return(NA)

}

# apply score to dataframe and convert to vector
score<-unlist(lapply(data$originalScore,fix_score))
data$score<-score

## Takes the first genre for each movie
  # Ex: "Mystery & thriller, Comedy, Romance" to "Mystery & thriller"
```

```
extract_genre <- function(x) {
  if(grepl(",", x)) {
    return(sub(",","",str_extract(x, ".*?,+")))
  } else { return(x); }
}

# remove all invalid score values
data <- data[(!(is.na(data$score))) & (data$score >= 0)&(data$score <= 1),] #
restrict to scores between 0 and 1.

## Getting Training and Testing Data
data$reviewState <- as.factor(data$reviewState)   # make sure to type cast to
factor, not keep as string.
data2$genre <- apply(as.matrix(data2$genre), extract_genre, MARGIN = 1)  #
extract the genre
no_nulls <- inner_join(data, data2,"id", relationship = "many-to-many")  #
join the data sets
no_nulls <- na.omit(no_nulls) # keep the non-null data
data_mixed <- no_nulls[sample(c(1:nrow(no_nulls)), 10000, replace = F),] #
take 10,000 random rows to work with a smaller dataset
data_mixed$score <- unlist(lapply(data_mixed$originalScore,fix_score))   #
turn list to vector
data_mixed$scoreSentiment <- as.factor(data_mixed$scoreSentiment)
train <- data_mixed[1:8000,]     # save training set
test <- data_mixed[8001:10000,]  # save testing set
# Attach polarities to respective datasets
train$polarity.name <- (train[,"reviewText"] %>% get_sentences() %>%
sentiment() %>% group_by(element_id) %>% summarise_at(vars(sentiment),
list(name = mean)))$name  # mean polarity over all sentances in review
test$polarity.name <- (test[,"reviewText"] %>% get_sentences() %>%
sentiment() %>% group_by(element_id) %>% summarise_at(vars(sentiment),
list(name = mean)))$name  # mean polarity over all sentances in review



# Get polarities - takes into account valence shifters (like ``never
better``)
polarity <- (train[,9] %>% get_sentences() %>% sentiment() %>%
group_by(element_id) %>% summarise_at(vars(sentiment), list(name = mean)))
polarities<- polarity %>% pull(name)
train$polarity<-polarities

polarity <- (test[,9] %>% get_sentences() %>% sentiment() %>%
group_by(element_id) %>% summarise_at(vars(sentiment), list(name = mean)))
polarities<- polarity %>% pull(name)
test$polarity<-polarities
```

```
# find the average polarity for each score and remove invalid polarities
p <- group_by(train, score) %>% filter(n()>1)
q <- summarise(p, total_count = n(), polarity = mean(polarity))
x <- q %>% pull(polarity)
y <- unlist(q %>% pull(score))
counts <- q %>% pull(total_count)


# create the correct number of points to plot
create_points <- function(v, counts){
      newv <- NULL
            for (i in (1:(length(counts)))) {
                  count = counts[i]
                  value = v[i]
                  for (i in (1:(count))) {
                        print(newv)
                        newv<-append(newv, value)
                  }
            }
      newv
}

x <- create_points(x,counts)
y <- create_points(y,counts)

# linear model
> summary(lm(y~x))

Call:
lm(formula = y ~ x)

Residuals:
     Min       1Q   Median       3Q      Max
-0.66886 -0.03918 -0.01168  0.05158  0.91671

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.470173   0.001262   372.6   <2e-16 ***
x           1.994591   0.009761   204.3   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
Residual standard error: 0.08865 on 8958 degrees of freedom
Multiple R-squared:  0.8234,    Adjusted R-squared:  0.8233
F-statistic: 4.176e+04 on 1 and 8958 DF,  p-value: < 2.2e-16

> lines(x,y_predicted, col='green')


# use glm() function to fit a model to the data
model <- glm(y ~ x, family = 'binomial')

> summary(model)

Call:
glm(formula = y ~ x, family = "binomial")

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.01910  -0.08600  -0.01609   0.11196   1.92611

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.12531    0.03065  -4.089 4.34e-05 ***
x            9.06452    0.25454  35.612  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1991.27  on 8959  degrees of freedom
Residual deviance:  552.45  on 8958  degrees of freedom
AIC: 6020.8

Number of Fisher Scoring iterations: 4

# find the predicted values of y
y_predicted <- predict(model, list(x), type="response")
coef <- coefficients(model)
> coef
(Intercept)           x
 -0.1253133   9.0645165
# plot the points and the curve found using glm() function
> plot(x,y, xlab='polarity', ylab='score', main = 'Review Polarity vs
Reviewer Score')
```

```
> curve((1/(1+exp(-(-coef[1] + coef[2]*x)))), add=T, col = 'blue')
```

```
# find accuracy of models found using test data
test_data <- function(test, func, params) {

      accuracy0.05 <- NULL
      accuracy0.1 <- NULL
      y_predicted = func(test$polarity, params)
      y = test$score
      for (i in (1:nrow(test))) {
            if (abs(y_predicted[i]-y[i]) > 0.1) {
                  accuracy0.1<-append(accuracy0.1, 0)
            } else {
                  accuracy0.1<-append(accuracy0.1, 1)
            }

            if (abs(y_predicted[i]-y[i]) > 0.05) {
                  accuracy0.05<-append(accuracy0.05, 0)
            } else {
                  accuracy0.05<-append(accuracy0.05, 1)
            }
      }
      within_0.1 <- mean(accuracy0.1)
      within_0.05 <- mean(accuracy0.05)

      print("Score accurate within 0.1")
      print(within_0.1)
      print("Score accurate within 0.05")
      print(within_0.05)

}


glmfunc <- function(x, coef) {
      y = 1/(1+exp(-(-coef[1] + coef[2]*x)))
      y
}


linfunc <- function(x, params) {
      y=0.470173+1.994591*x
      y
}



> test_data(test,linfunc,0)
[1] "Score accurate within 0.1"
[1] 0.2277722
[1] "Score accurate within 0.05"
[1] 0.06293706


> test_data(test, glmfunc, coef)
[1] "Score accurate within 0.1"
[1] 0.1958042
[1] "Score accurate within 0.05"
[1] 0.09390609


### SVM and Logit Comparison code
      # Fit the SVM
model <- svm(scoreSentiment ~ polarity.name + score, data = train, type =
"C")
plot(model, train, score ~ polarity.name)
pred <- predict(model, test)  # make predictions using SVM model

> table(ifelse(pred == test$scoreSentiment, 1, 0))

   0    1
  97 1903

> 97/(2000)  # SVM testing misclassifications
[1] 0.048

# Convert to numeric predictions for ROC curve
pred <- as.numeric(ifelse(pred == "POSITIVE", 1, 0))
train$scoreSentimentNumeric <- ifelse(train$scoreSentiment == "POSITIVE", 1,
0)
test$scoreSentimentNumeric <- ifelse(test$scoreSentiment == "POSITIVE", 1, 0)
roc_score <- roc(test$scoreSentimentNumeric, pred) # get ROC curve

> auc(roc_score)  # get the AUC
Area under the curve: 0.9278
```

```
plot(roc_score, main = "SVM ROC")

## Logit Model stuff
      # To do logistic regression, we need to first convert the classes to
{0, 1}
train$scoreSentimentNumeric <- ifelse(train$scoreSentiment == "POSITIVE", 1,
0)
test$scoreSentimentNumeric <- ifelse(test$scoreSentiment == "POSITIVE", 1, 0)
model <- glm(scoreSentimentNumeric ~ score, family=binomial, data = train)

      # Obtain the logit predictions
pred <- predict(model, test, type="response")
Pred_df <- data.frame(test$score)
Pred_df$pred <- ifelse(pred >= 0.5, 1, 0)
plot(scoreSentimentNumeric ~ score, data = train)
lines(pred ~ test.score, Pred_df, lwd=2)

> table(ifelse(pred >= 0.5, 1, 0))  # place in correct classes based on
probabilities obtained from sigmoid (generated from glm function)
   0    1
 599 1401

> 599/2000  # logit testing misclassifications
[1] 0.2995

roc_score <- roc(test$scoreSentimentNumeric, pred) # Get ROC curve

> auc(roc_score)
Area under the curve: 0.9714

plot(roc_score, main = "Logit ROC")

      # Use GGPlot to have a prettier plot
ggplot(train, aes(x=score, y=scoreSentimentNumeric)) +
  geom_point(alpha=.5) +
  stat_smooth(method="glm", se=FALSE, method.args = list(family=binomial),
              col="blue")

# Naive Bayes model and plotted likelihood functions
library(klaR)
library(pROC)
model <- NaiveBayes(scoreSentiment ~ polarity + score, data = train)
```

```
> plot(model)
#ROC And AUC
pred <- predict(model, test, type = "raw")
# Extract the posterior probabilities for the positive class
score <- pred$posterior[, 2]
roc_obj <- roc(test$scoreSentiment, score)
plot(roc_obj,  main = "Naive Bayes ROC")

> auc(roc_obj)
Area under the curve: 0.9795

pred_class <- ifelse(pred$posterior[, 2] > 0.5, "POSITIVE", "NEGATIVE")

# calculate misclassification rate
> table(ifelse(pred_class == test$scoreSentiment, 1, 0))
   0    1
 117 1883

> (117/2000) # Naive Bayes test misclassification_rate
[1] 0.0585


# use ggplot to plot polarity against score with color distinguishing top
critic
ggplot(data, aes(x = polarity, y = score, color = isTopCritic)) +
geom_point()
# for bar plot
 ggplot(train, aes(x = score, y = reviewState, fill = isTopCritic)) +
+     geom_boxplot() +
+     labs(x = "Score", y = "Freshness") +
+     scale_fill_discrete(name = "Top Critic")

> t.test(train$score ~ train$isTopCritic)

      Welch Two Sample t-test

data:  train$score by train$isTopCritic
t = 5.0458, df = 3901.2, p-value = 4.723e-07
alternative hypothesis: true difference in means between group FALSE and
group TRUE is not equal to 0
95 percent confidence interval:
 0.01635393 0.03713912
sample estimates:
```

```
mean in group FALSE  mean in group TRUE 
          0.6464024           0.6196559 

> t.test(train$polarity ~ train$isTopCritic)

	Welch Two Sample t-test

data:  train$polarity by train$isTopCritic
t = 1.0492, df = 4017.8, p-value = 0.2942
alternative hypothesis: true difference in means between group FALSE and
group TRUE is not equal to 0
95 percent confidence interval:
 -0.007680361  0.025364113
sample estimates:
mean in group FALSE  mean in group TRUE 
         0.08271341          0.07387153 
```