Determining Authentic News Articles using Machine Learning

Uday Adusumilli

1st March 2022

Introduction

Overview

A cursory search of Google Trends[*1] indicates that peak interest in the search term "Fake News" occurred in October of 2016. In comparison, the same data reported from January of 2004 through September of 2016 shows a relative average interest rate of just 4.1%, while since October of 2016, the relative interest rate is 38%. The concept of Authentic vs. Non-Authentic News has certainly been around for as long as the news itself, but the buzzword "Fake News" has raised it to a whole new level of awareness.

Today, with the sheer amount of information available, it is more important than ever to distinguish between accurate information and stories that are exaggerated, poorly sourced, or even flatly wrong. Sadly, it has become increasingly challenging; individuals and organizations creating disinformation use advanced machine learning and data analytics techniques to target their messages. In order to reinforce loosely held opinions into firm beliefs, propaganda on a particular subject can be targeted at and delivered to individuals near the verge of believing it using their confirmation bias.

In the same way, such attacks can be distinguished from truer information using this approach. We will need non-partisan fact-checking sources like Politifact[*2] and other non-partisan sources of classifications to teach machine learning algorithms how to sort the deluge of news into what is worth keeping and what should be rejected. This dataset was sourced from its original collectors, at the University of Victoria ISOT Research Lab[1].

Machine Learning Approaches

Authentic News can only be identified by training our machine learning algorithms to read. Current methods involve Natural Language Processing. The technique we've used is called Sentiment Analysis and it is a more rudimentary approach. Parts of the text are broken down into *tokens*, then each token is assigned a sentiment metric. In a *lexicon*, words and the sentiments attached to them are correlated according to word correlations.

Lexicons

Sentiment Analysis relies on lexicons and is a labor-intensive process. In the following analysis, all three lexicons were compiled by crowdsourcing, in which respondents made associations to given words in accordance with a scale or set of options provided to them. Each of them uses a different method of association, and their analyses have produced different outcomes.

[1] https://www.uvic.ca/engineering/ece/isot/

Afinn The Afinn Sentiment Lexicon[1] presents a single scale of values to measure sentiment. There is a range of -5 to 5 on the scale, and it is expressed as an integer. There are 2477 tokens associated with this lexicon.

NRC The NRC Word-Emotion Association Lexicon[2] presents each token with one or more associated emotions. The combined sentiment consists of two emotion components: "negative" and "positive," and eight feelings: "anger," "anticipation," "disgust," "fear," "joy," "sadness," "surprise," and "trust." It has associations for 6468 tokens, which is a significant increase over the Afinn lexicon.

NRC VAD The NRC Valence, Arousal, and Dominance Lexicon[3] is based on influential factor analysis and Best-Worst scoring, each token is scored on three dimensions. As a descriptor of the excitement-calming or active-passive sentiment, the "valence" dimension is a positive-negative scale. As a measure of strength and weakness, the "dominance" dimension is a positive-negative scale. As a decimal value, each dimensional score ranges from 0 to 1. This makes it the finest lexicon available. It is also the largest, with sentiment associations associated with 20,073 tokens across all three dimensions.

Analysis

The Data

The Authentic-News Dataset comes in the form of two csv files: Non-Authentic.csv and Authentic.csv. To begin our analysis we can read these into a single dataset and assign a new column differentiating the rows:

```
# Read Authentic.csv into memory
authentic.news <- fread(file.path("data", "Authentic.csv"))
# Assign a new column denoting this is Authentic news
authentic.news[, is_NonAuthentic := FALSE]

# Read Non-Authentic.csv into memory
nonauthentic.news <- fread(file.path("data", "Non-Authentic.csv"))
# Assign a new column denoting this is NonAuthentic news
nonauthentic.news[, is_NonAuthentic := TRUE]

# Combine the tables
all.news <- rbind(authentic.news, nonauthentic.news)

# Remove intermediary objects
remove(authentic.news, nonauthentic.news)
```

As soon as we inspect the dataset, we can see that the sources have poor character encoding. There are a few issues with the list of titles, such as apostrophes mis-encoded to *â€™M*, and a few examples of a special form of double quote shown to be “. The easiest way to deal with this is to re-encode everything into the same standard up front. Our titles and texts can also be trimmed of leading and trailing whitespace at the same time.

```
all.news[, `:=`(
    title = str_trim(iconv(title, from = "utf8", to = "latin1")),
    text = str_trim(iconv(text, from = "utf8", to = "latin1"))
)]
```

Inspection for missing data....

```
# Empty title cells
all.news[is.na(title) | title == "", .N]
```

```
## [1] 9
```

```
# Empty text cells
all.news[is.na(text) | text == "", .N]
```

```
## [1] 651
```

```
# Empty text cells aggregated by is_NonAuthentic
all.news[
  is.na(text) | text == ""
][
  ,
  .N,
  by = is_NonAuthentic
]
```

```
##    is_NonAuthentic   N
## 1:           FALSE  21
## 2:            TRUE 630
```

There are missing data, as we can see. There are nine blank titles, 630 Non-Authentic News texts, and 21 Authentic news texts. As a final result, we combine these two into one text column to parse, so as long as the observation at least has a title, we can continue analyzing.

```
all.news <- all.news[!is.na(title) & title != ""]
```

Exploration

There were built-in biases, which I was aware of. A scan through the Authentic.csv file reveals one of the most glaring of these flaws: Almost all of the texts mention or begin with the word "Reuters."

```
# Create a column showing whether or not the word "Reuters" is in the text
all.news[, has_reuters := grepl("reuters", text, ignore.case = TRUE)]

# Summarize this column disaggregated by is_NonAuthentic
all.news[, .N, by = list(has_reuters, is_NonAuthentic)]
```

```
##    has_reuters is_NonAuthentic     N
## 1:        TRUE           FALSE 21357
## 2:       FALSE           FALSE    59
## 3:       FALSE            TRUE 23151
## 4:        TRUE            TRUE   322
```

This phenomenon is unreasonably prevalent, threatening the accuracy of our analysis.

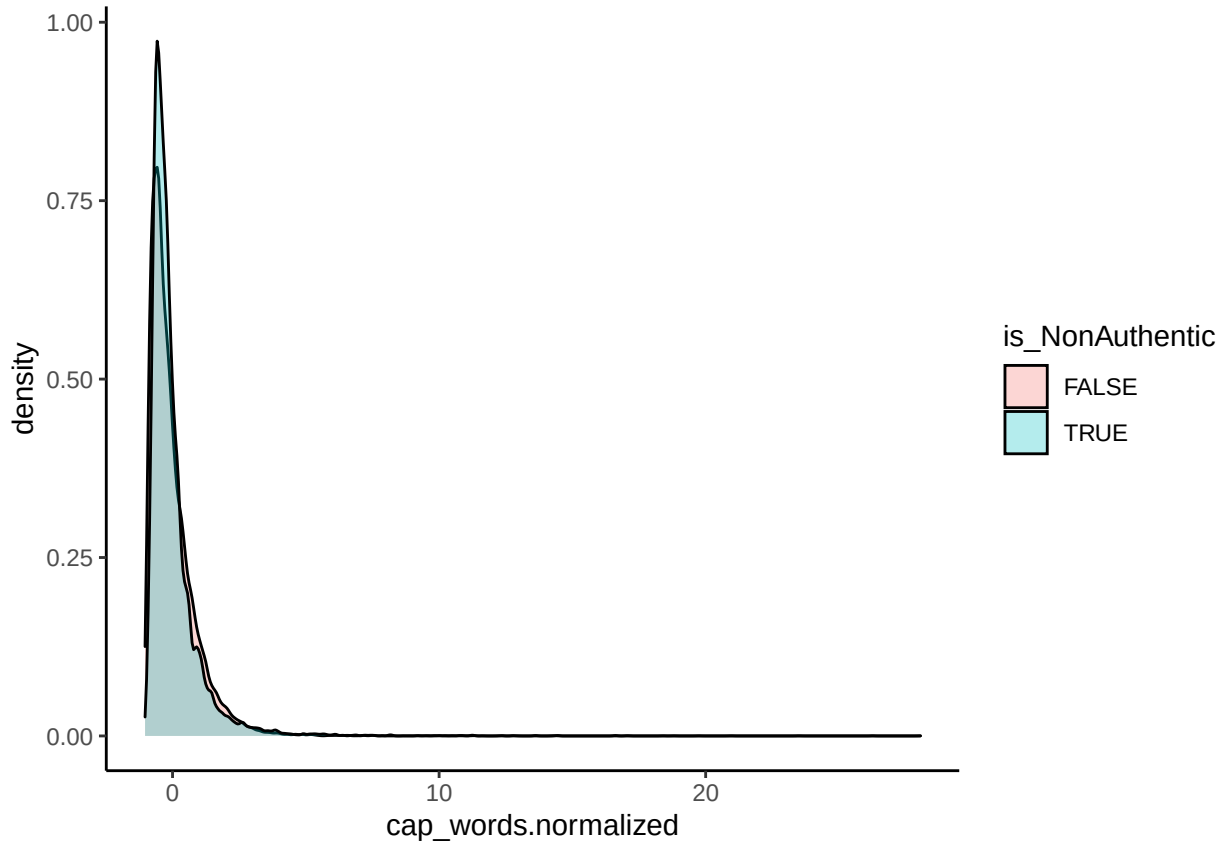
```
reuters.guess <- data.table(
    prediction = as.factor(!all.news$has_reuters),
    observation = as.factor(all.news$is_NonAuthentic)
)

table(reuters.guess$prediction, reuters.guess$observation)
```

```
##
##         FALSE  TRUE
##   FALSE 21357   322
##   TRUE     59 23151
```

As you can see, this level of accuracy is over 99%, but none of the lexicons used in the sentiment analysis below include the word "Reuters" in any form. Thus, our results won't be affected by this.

An additional scan of the data reveals that some articles use capitalized words more often than others. The number of entirely capitalized words can be visualized on a plot by counting them. So that we can examine if there is a different correlation between fake and true news, we will normalize the values within the two realms.

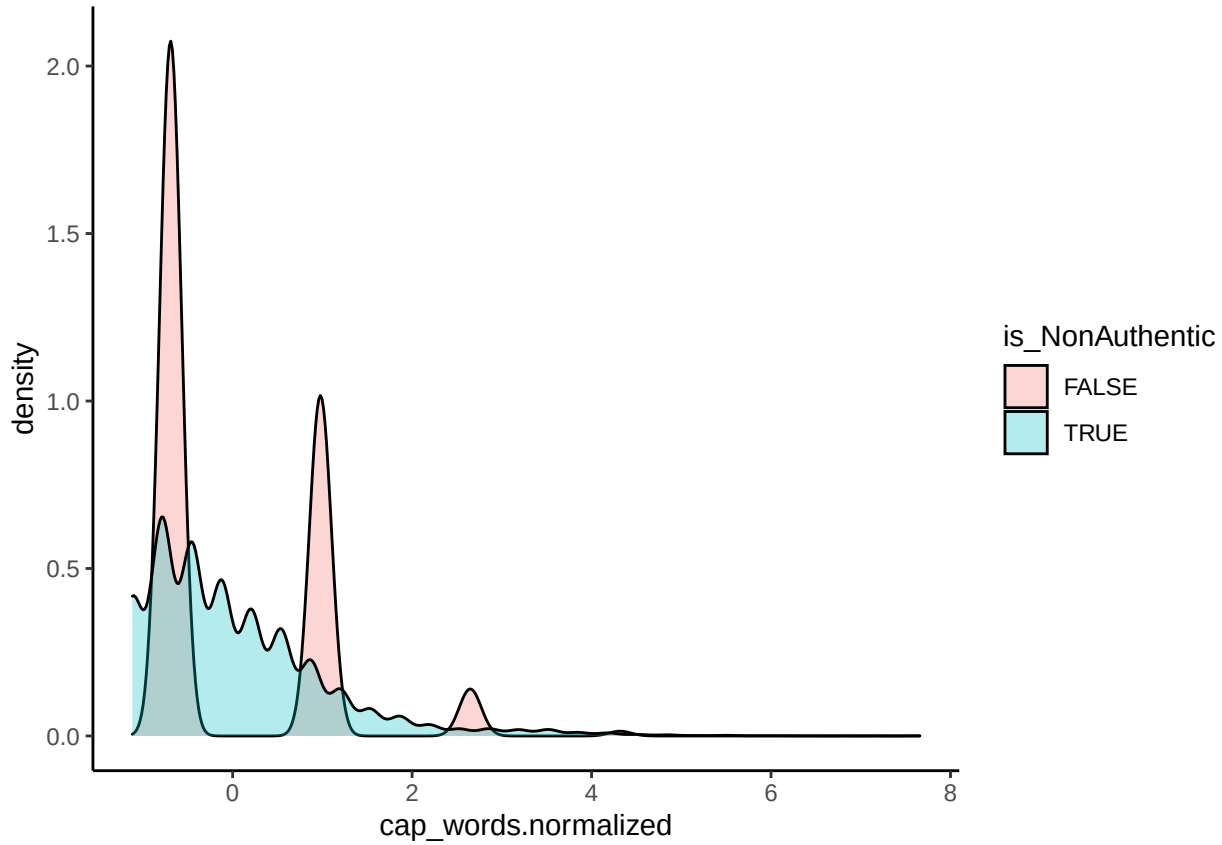
```
# Join the title and text of each observation into a column called full_text
all.news[, full_text := paste(title, text, sep = " ")]

# Count occurrences of whole words that are capitalized in each full_text
all.news[, cap_words := str_count(full_text, "[^a-z][A-Z]+[^a-z]")]

# normalize both sets of data separately, then plot
all.news[
  is_NonAuthentic == TRUE,
  cap_words.normalized := (cap_words - mean(cap_words))/sd(cap_words)
][
  is_NonAuthentic == FALSE,
  cap_words.normalized := (cap_words - mean(cap_words))/sd(cap_words)
]

# plot
all.news %>%
  ggplot(aes(x = cap_words.normalized, fill = is_NonAuthentic)) +
  geom_density(alpha = 0.3)
```



While their magnitudes may differ, their shapes are remarkably similar. Using just the titles, we can perform the same analysis.

```
# Count occurrences of whole words that are capitalized in just the title
all.news[, cap_words := str_count(title, "[^a-z][A-Z]+[^a-z]")]
```

```r
# normalize both sets of data separately, then plot
all.news[
  is_NonAuthentic == TRUE,
  cap_words.normalized := (cap_words - mean(cap_words))/sd(cap_words)
][
  is_NonAuthentic == FALSE,
  cap_words.normalized := (cap_words - mean(cap_words))/sd(cap_words)
]

# plot
all.news %>%
  ggplot(aes(x = cap_words.normalized, fill = is_NonAuthentic)) +
  geom_density(alpha = 0.3)
```



It is possible that titles have fewer words, so we end up with a lot of divergent density plots. One of our predictors will be the number of capitalized words in the title.

Additionally, these data sources included a column of subjects. In addition, it is clear that another bias is inherent in this study, in that the subjects found in the Non-Authentic.csv data are absent from the Authentic.csv data and vice versa.

```
# Viewing all Subjects
all.news[, .N, by = subject]
```

```
##            subject     N
## 1:    politicsNews 11272
## 2:       worldnews 10144
## 3:            News  9049
## 4:        politics  6837
## 5: Government News  1569
## 6:       left-news  4457
## 7:         US_News   783
## 8:     Middle-east   778
```

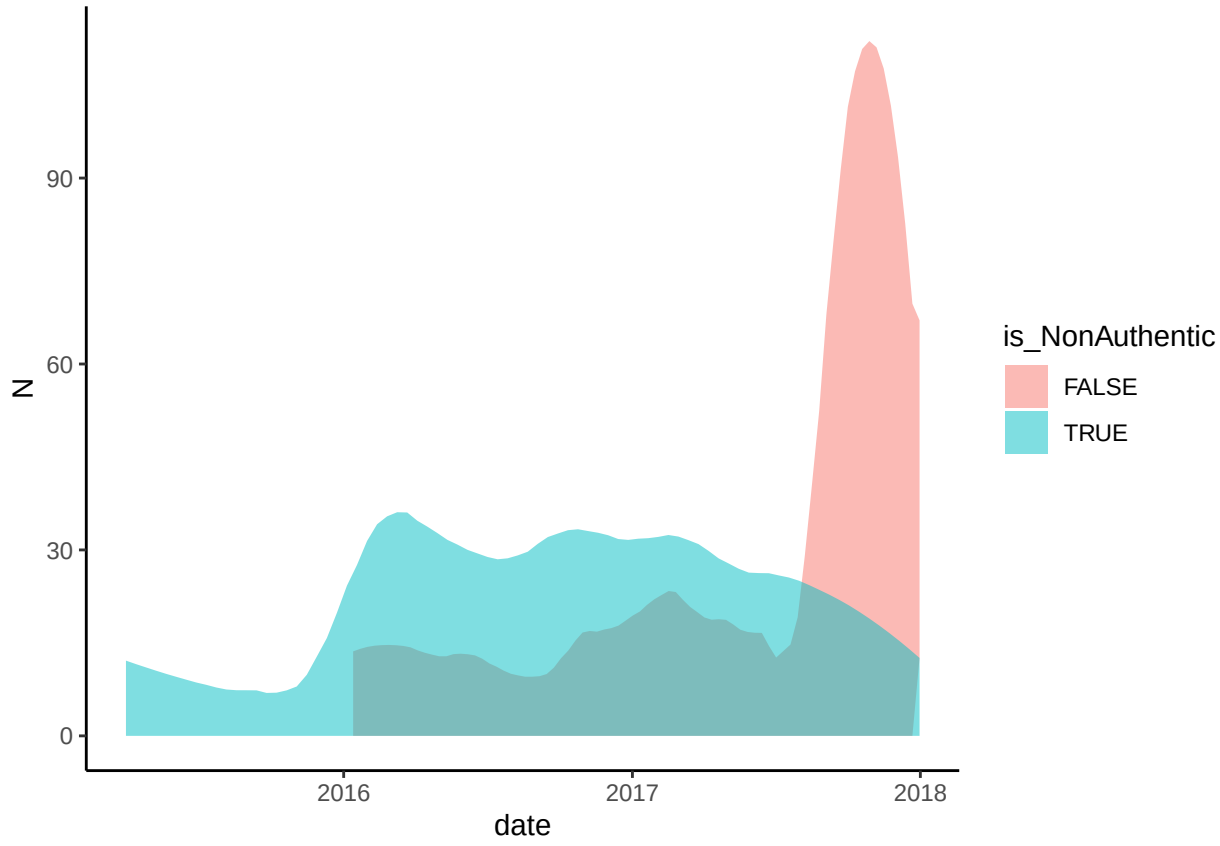
```
# Viewing subjects disaggregated by is_NonAuthentic
all.news[, .N, by = list(subject, is_NonAuthentic)]
```

```
##            subject is_NonAuthentic     N
## 1:    politicsNews           FALSE 11272
## 2:       worldnews           FALSE 10144
## 3:            News            TRUE  9049
## 4:        politics            TRUE  6837
## 5: Government News            TRUE  1569
## 6:       left-news            TRUE  4457
## 7:         US_News            TRUE   783
## 8:     Middle-east            TRUE   778
```

The observations are also all dated in some way. Although the vast majority of dates here are from 2016-2018, the spread is not uniform. Several dates fail to parse when this step is run. The inspection of these failures (not shown here) reveals that 35 observations use a different date format, and ten observations have URLs in the date column. This data needs to be consistently imputed if we wish to use it later. The URLs contain dates, which could be corrected by hand, but after this point in analysis, the date column is not leveraged.

```
# Parse the date in an R date object
all.news[, parsed_date := mdy(date)]

# plot counts over time
all.news[, .N, by = .(is_NonAuthentic, date = date(parsed_date))] %>%
  ggplot(aes(x = date, y = N, fill = is_NonAuthentic)) +
  stat_smooth(
    geom = 'area',
    span = 0.25,
    alpha = 0.5,
    method = 'loess',
    position = "stack"
  )
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Approaches and Models

Data Preparation Our first step in the analysis will be to clean the data, tokenize the texts, bind the tokens with the corresponding sentiments in each of the three lexicons, and finally aggregate the resulting sentiments with respect to each article.

Tokenization will take place per word for this dataset. Tokenization by ngrams or n words is commonly used for in-depth analysis. These are typically represented by bigrams, i.e. pairs of words, allowing a difference to be detected between tokens such as "successful" and their inverse sentiment, "not successful."

Tokenized values also undergo another cleaning step: stopwords are removed. There are several stop words, including "and," "is," and "so." As such, they are no longer included in the analysis.

Additionally, it is important to note that merging tokens and sentiment sets is performed using *inner_join*, which will inherently reduce the size of the tokenized data list to only those rows in which there is an appropriate sentiment. Because of this, lexicon size is a key factor to consider when selecting one for analysis.

Additionally, we will divide the data into a training set and a test set to ensure the accuracy of our models.

```
# Clean and tidy dataset
all.news <- lazy_dt(all.news) %>%
  mutate(
    title = str_trim(iconv(title, from = "utf8", to = "latin1")),
    text = str_trim(iconv(text, from = "utf8", to = "latin1"))
  ) %>%
  filter(!is.na(title) & title != '') %>%
  mutate(
    full_text = paste(text, title),
    is_NonAuthentic = as.factor(is_NonAuthentic),
    title_caps = str_count(title, "[^a-z][A-Z]+[^a-z]")
  ) %>%
  select(title, full_text, is_NonAuthentic, title_caps) %>%
  as.data.table()

# Mark training and test datasets.
train.index <- createDataPartition(
  all.news$is_NonAuthentic,
  p = 0.8,
  times = 1,
  list = FALSE
)

all.news$set <- "testing"
all.news[train.index, set := "training"]

remove(train.index)

# split tokens for joining sentiment, remove stop words.
# This takes a few moments
tokenized <- all.news %>%
  unnest_tokens(token, full_text) %>%
  lazy_dt() %>%
  anti_join(data.table(token = stop_words$word), by = "token") %>%
  as.data.table()
```

Each of the three lexicons will now be attached to these tokens separately. Each lexicon provides a different measure of sentiment, so different aggregations will be required. Affin lexicon provides an integer column with a simple summation method.

```
# Read the data from file
afinn <- fread("./data/afinn.csv")

# Change names of columns for joining
setnames(afinn, c("token", "sentiment"))

# setting data.table keys make joins a lot faster
setkey(afinn, token)
setkey(tokenized, token)

# data.table syntax for doing an inner join on keyed data.tables
afinn <- afinn[tokenized, nomatch = NULL]

# aggregate total sentiment for each article
afinn <- afinn[
  ,
  list(sentiment = sum(sentiment)),
  by  = list(title, is_NonAuthentic, title_caps, set)
]
```

To aggregate the NRC lexicon requires far more effort. Our first step is to join the tokens to the dataset, which eliminates all words that are not present in the NRC lexicon. However, this same join also multiplies many of the rows by the number of sentiments a token has. To aggregate these multiple rows into columns, we must first create an intermediary process that shows whether or not a given token has the attached sentiments. Afterwards, these can be aggregated into per-article totals.

```
nrc <- fread("./data/nrc.csv")

# Change column names for joining
setnames(nrc, "word", "token")

# Set keys for data.table join
setkey(nrc, token)
setkey(tokenized, token)

# data.table syntax for doing an inner join on keyed data.tables
nrc <- nrc[tokenized, nomatch = NULL]

# Tibbles with pivot_wider is a much easier-to-read approach here,
# but there are other more performant ways of doing this if our
# dataset was very large. See `reshape2` package
nrc <- as_tibble(nrc) %>%
  pivot_wider(
    names_from = sentiment,
    values_from = sentiment,
    values_fn = list(sentiment = length),
    values_fill = list(sentiment = 0)
  )
```

```
# View the wide output of this table
head(nrc, 3)
```

```
## # A tibble: 3 x 15
##   token  title     is_NonAuthentic title_caps set   trust  fear negative sadness
##   <chr>  <chr>     <fct>                <int> <chr> <int> <int>    <int>   <int>
## 1 abacus Canada's~ FALSE                    0 trai~     1     0        0       0
## 2 aband~ Spy chie~ FALSE                    0 test~     0     1        1       1
## 3 aband~ Schumer:~ FALSE                    0 trai~     0     1        1       1
## # ... with 6 more variables: anger <int>, surprise <int>, positive <int>,
## #   disgust <int>, anticipation <int>, joy <int>
```

```
# Roll up counts of all sentiments for all tokens for each article, ie Anger: 5, Joy 0, Negative: 3
nrc <- nrc %>%
  group_by(title, is_NonAuthentic, title_caps, set) %>%
  summarize_at(vars(-token), list(sum)) %>%
  as.data.table()

head(nrc, 3)
```

```
##                                                                      title
## 1:     '#1 In Bigotry': Twitter EVISCERATES Mississippi Gov. Over Anti-Gay Law
## 2: '60 MINUTES': 9/11 REPORT Could Incriminate Saudi Arabia With These Details
## 3:          'A better future' - Britain's May tries to rally her Conservatives
##    is_NonAuthentic title_caps      set trust fear negative sadness anger
## 1:            TRUE          1 training    21   15       17      13    14
## 2:            TRUE          2 training    12    9       10       6     6
## 3:           FALSE          1 training    23   10       20      10    11
##    surprise positive disgust anticipation joy
## 1:        2       18      11           10   7
## 2:        3       16       1           10   6
## 3:        4       38       5           10  11
```

As a final note, the NRC VAD lexicon has been aggregated like Afinn's, but with columns for each dimension.

```
# Read the NRC VAD data from file
vad <- fread("./data/nrc_vad.csv")

# This lexicon comes with Title-cased columns
setnames(vad, tolower(names(vad)))

# change this column name for joining
setnames(vad, "word", "token")

setkey(vad, token)
setkey(tokenized, token)

# data.table syntax for an inner join on keyed data.tables
vad <- vad[tokenized, nomatch = NULL]

# Aggregate dimensions by summing across articles
vad <- vad[
```

```
  ,
  list(
    valence = sum(valence),
    arousal = sum(arousal),
    dominance = sum(dominance)
  ),
  by = list(title, is_NonAuthentic, title_caps, set)
]
```

The test and training sets can now be separated. The data sets were kept together for processing because the above data preparation relies heavily on merges. The training and testing identifiers will be separated here.

```
afinn <- split(afinn, by = "set", keep.by = FALSE)
afinn.training <- afinn$training
afinn.testing <- afinn$testing

nrc <- split(nrc, by = "set", keep.by = FALSE)
nrc.training <- nrc$training
nrc.testing <- nrc$testing

vad <- split(vad, by = "set", keep.by = FALSE)
vad.training <- vad$training
vad.testing <- vad$testing

# clean up unneded objects
remove(tokenized, afinn, nrc, vad)
```

Naive Baseline Using the most naive approach as a baseline always proves to be instructive. The intended inference can be summed up by tossing a coin and guessing accordingly. We can predict that based on the observed data, this would be approximately half the time correct given a normal distribution of binary categorical outcomes. Exactly 50% of the time, this holds true for the observed data.

```
# Replicate the naive approach 10000 times
mean(replicate(10000, {
  # Guess True or False for is_NonAuthentic randomly
  predictions <- sample(c(TRUE, FALSE), nrow(all.news), replace = TRUE)

  # Return the accuracy of this replication
  mean(predictions == all.news$is_NonAuthentic)
}))
```

```
## [1] 0.5
```

Random Forests Due to the numerical and continuous nature of our predictors, decision trees are a very common classification algorithm. Each set of data will be mapped to a sentiment lexicon by creating and training a model. Using *parRF*, a parallelized Random Forest implementation, this analysis is accomplished more quickly. This package's caret training method only has one tuning parameter, *mtry*, and I've been satisfied with the default settings for small data sets. I have implemented this here on a consumer PC with a roughly 4Ghz processor running 16 threads.

```
# We can leveraging matrix-based function signatures for all the models we build
# This helper function will create a matrix of all predictors from a given data.table
makePredictors <- function(dt) {
  # drop our title, used only as an identifier column
  dt$title = NULL

  # drop the response column
  dt$is_NonAuthentic = NULL

  # return the rest of the columns as a matrix
  as.matrix(dt)
}

# Start and register parallel threads
# NB: Never set this number higher than your computer
# can handle!
nThreads <- 16
cl <- makeSOCKcluster(nThreads)
registerDoSNOW(cl)

# A `caret` trainControl object, using parallized
# 5-fold cross-validation.
rf.trainControl <- trainControl(
  method = "cv",
  number = 5,
  allowParallel = TRUE
)

# RF model for Afinn sentiments
afinn.rf.model <- train(
  makePredictors(afinn.training),
  afinn.training$is_NonAuthentic,
  method = "parRF",
  trControl = rf.trainControl
)
```

```
## note: only 1 unique complexity parameters in default grid. Truncating the grid to 1 .
```

```
# RF NRC
nrc.rf.model <- train(
  makePredictors(nrc.training),
  nrc.training$is_NonAuthentic,
  method = "parRF",
  trControl = rf.trainControl
)

# RF VAD
vad.rf.model <- train(
  makePredictors(vad.training),
  vad.training$is_NonAuthentic,
  method = "parRF",
  trControl = rf.trainControl
)
```

```
# Stop and de-register parallel computing
stopCluster(cl)
registerDoSEQ()
remove(cl)
```

We can now evaluate the models' accuracy by comparing them to the training data with which they were built. While this is obviously overfit, it does begin to show us if these attempts were even close to succeeding.

```
## Accuracy measures against training datasets
# Afinn RF
confusionMatrix(fitted(afinn.rf.model), afinn.training$is_NonAuthentic)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE  TRUE
##      FALSE 14847  4057
##      TRUE   1523 10859
##
##                Accuracy : 0.822
##                  95% CI : (0.817, 0.826)
##     No Information Rate : 0.523
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.64
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.907
##             Specificity : 0.728
##          Pos Pred Value : 0.785
##          Neg Pred Value : 0.877
##              Prevalence : 0.523
##          Detection Rate : 0.475
##    Detection Prevalence : 0.604
##       Balanced Accuracy : 0.817
##
##        'Positive' Class : FALSE
##
```

```
# 82.2%

# NRC RF
confusionMatrix(fitted(nrc.rf.model), nrc.training$is_NonAuthentic)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE  TRUE
##      FALSE 16724    81
##      TRUE     12 15039
##
```

```
##                Accuracy : 0.997
##                  95% CI : (0.996, 0.998)
##     No Information Rate : 0.525
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.994
##
##  Mcnemar's Test P-Value : 1.77e-12
##
##             Sensitivity : 0.999
##             Specificity : 0.995
##          Pos Pred Value : 0.995
##          Neg Pred Value : 0.999
##              Prevalence : 0.525
##          Detection Rate : 0.525
##    Detection Prevalence : 0.528
##       Balanced Accuracy : 0.997
##
##        'Positive' Class : FALSE
##
```

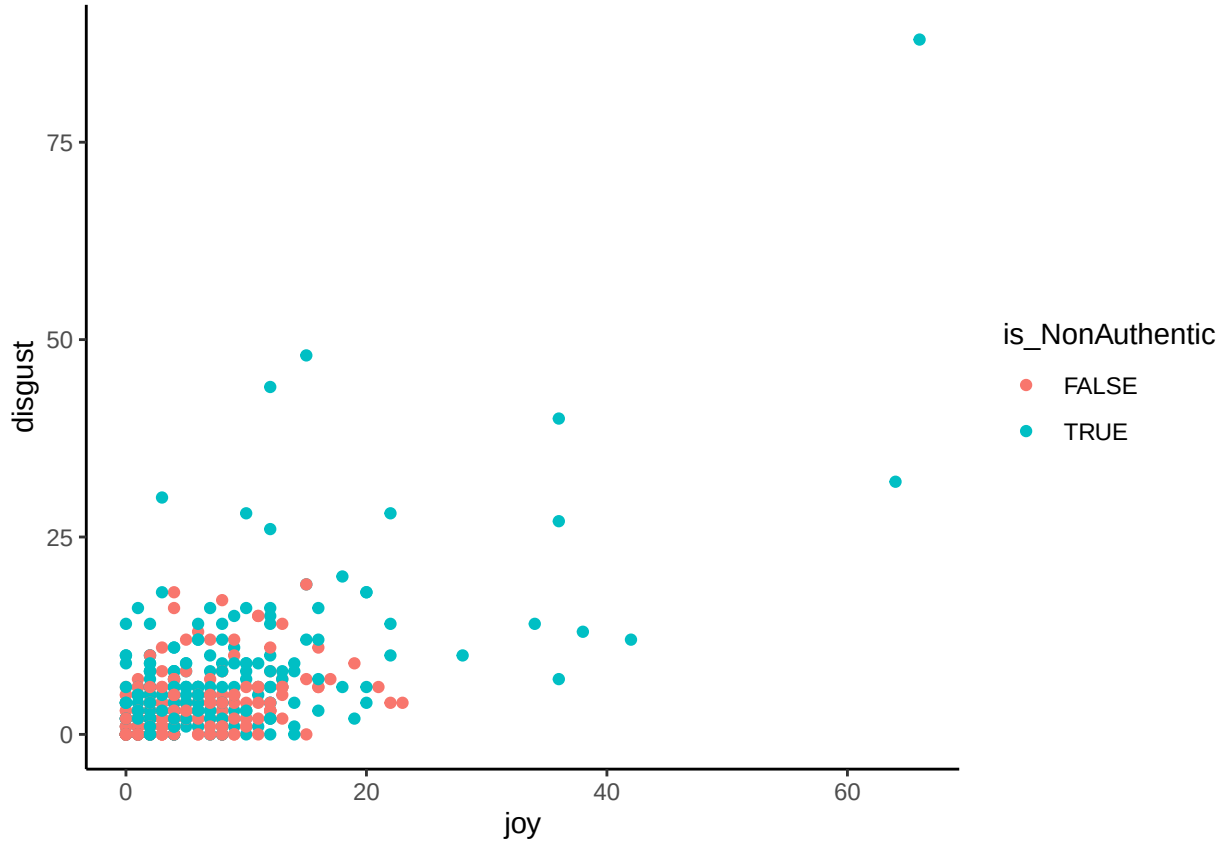
```
# 99.7%

# NRC VAD RF
confusionMatrix(fitted(vad.rf.model), vad.training$is_NonAuthentic)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE  TRUE
##      FALSE 16726    35
##      TRUE     17 15133
##
##                Accuracy : 0.998
##                  95% CI : (0.998, 0.999)
##     No Information Rate : 0.525
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.997
##
##  Mcnemar's Test P-Value : 0.0184
##
##             Sensitivity : 0.999
##             Specificity : 0.998
##          Pos Pred Value : 0.998
##          Neg Pred Value : 0.999
##              Prevalence : 0.525
##          Detection Rate : 0.524
##    Detection Prevalence : 0.525
##       Balanced Accuracy : 0.998
##
##        'Positive' Class : FALSE
##
```

```
# 99.8%
```

Support Vector Machines Support Vector Machines are mathematically based models that draw boundaries around collections of data in higher-order planes. It is important to note that the *ksvm* function used here, as well as the analysis at large, is a type of SVM that leverages something called the *kernel trick*: an ingenious application of mathematical assumptions that makes SVMs work well in higher dimensions. TIt is not my purpose to describe the mathematics of SVMs in this paper, so I will demonstrate their ability to separate clusters of data visually.

```
# Grab a very small chunk of data to demonstrate with
# 500 rows, with only "joy" and "disgust" dimensions as predictors
data.small <- nrc.training[
  sample.int(nrow(nrc.training), 500),
  list(is_NonAuthentic, joy, disgust)
]

# Scatterplot of the predictors, color coded by outcome
data.small %>%
  ggplot(aes(x = joy, y = disgust, color = is_NonAuthentic)) +
  geom_point()
```



As we can see, there does seem to be a general visual trend that one color is more prevalent in the upper-right of the plot, while the cluster in the lower left seems to be more focused on the other color. Now we can train a model on this data.

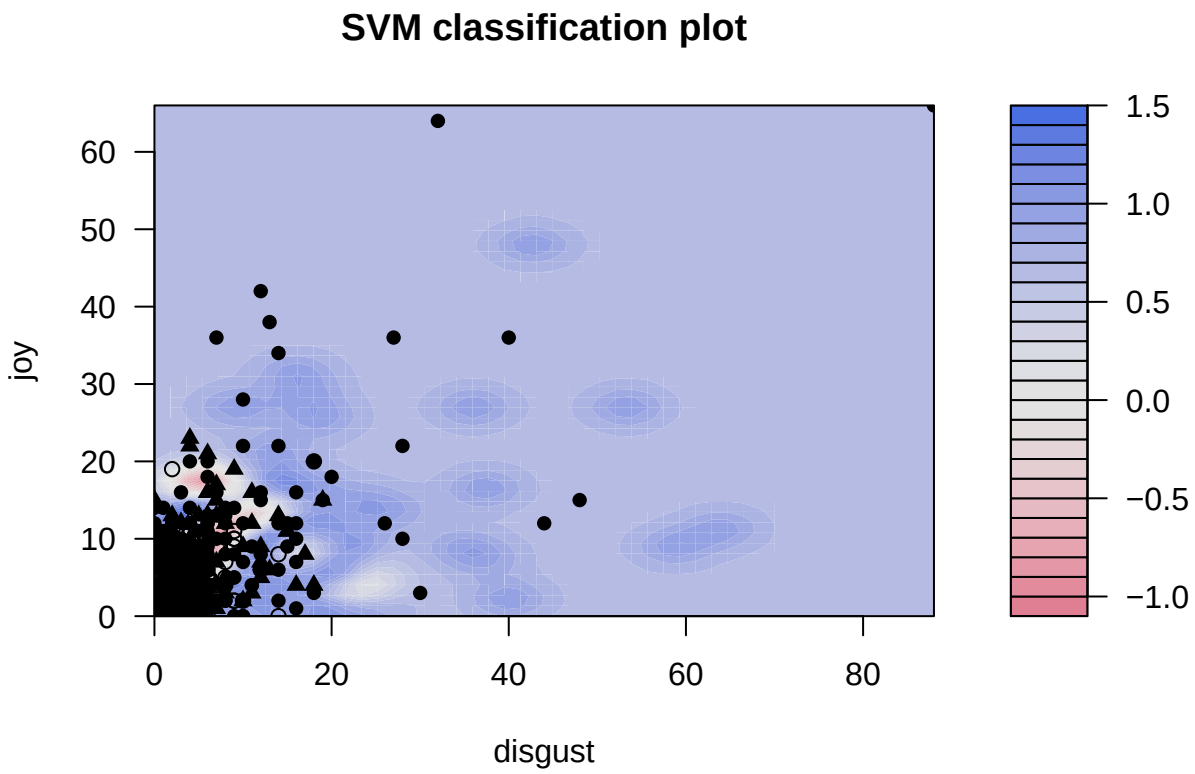
```
# Grab a matrix of predictors and a response vector
predictors <- as.matrix(data.small[, .(joy, disgust)])
response <- data.small$is_NonAuthentic

# Train a basic KSVM model
ksvm.model <- ksvm(
  x = predictors,
  y = response
)

# Make predictions and view accuracy
ksvm.predictions <- predict(
  ksvm.model,
  as.matrix(nrc.training[, .(joy, disgust)])
)
confusionMatrix(ksvm.predictions, nrc.training$is_NonAuthentic)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE  TRUE
##      FALSE 12219  7480
##      TRUE   4517  7640
##
##                Accuracy : 0.623
##                  95% CI : (0.618, 0.629)
##     No Information Rate : 0.525
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.238
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.730
##             Specificity : 0.505
##          Pos Pred Value : 0.620
##          Neg Pred Value : 0.628
##              Prevalence : 0.525
##          Detection Rate : 0.384
##    Detection Prevalence : 0.618
##       Balanced Accuracy : 0.618
##
##        'Positive' Class : FALSE
##
```

```
# 62.3%
```

The model's accuracy isn't surprising given the limited data and only two dimensions. A similar scatterplot can be used to demonstrate the boundaries of the model.

```
# Visualize the Boundaries our KSVM model has created.
plot(ksvm.model, data = predictors)
```



It is clear that the model has a lot of learning to do, but the boundaries drawn seem reasonable. Machine learning problems are often complicated in the middle, so more data, more dimensions, and a better algorithm should improve our results.

Our KSVM models are now ready for training. It is simply a matter of training sigma, which affects how linear or flexible the decision boundary becomes. In addition to C, there is a cost parameter that is used to penalize residuals that are too large after normalization. I have left this as the default value of 1 due to the relatively small size of our data.

As a result of a feature in the KernLab package named *sigest*, the tuning values for *sigma* are estimated based on the data fractions of the training set. This range has been expanded by 25 percent on either side, covering a very large number of possible parameters. Each model is trained using lexicon-bound training data, so this step is done for every model.

```
# Create and register threads for parallel computing
cl <- makeSOCKcluster(nThreads)
registerDoSNOW(cl)

# Create a trainControl object for all KSVM models
ksvm.trainControl <- trainControl(
  method = "cv",
  number = 5,
  allowParallel = TRUE
)

# Create the matrix of predictors
afinn.training.predictors <- makePredictors(afinn.training)
```

```
# Here, and below, sigmas are going to come from an estimation function provided
# in the `kernlab` package. The range of these will be expanded by 25% so that
# we can try a broader set of values for sigma.
afinn.training.sigmas <- sigest(afinn.training.predictors, frac = 1)
afinn.training.sigmas <- seq(
  afinn.training.sigmas["90%"] * 0.75,
  afinn.training.sigmas["10%"] * 1.25,
  length.out = 10
)

# train the model
afinn.ksvm.model <- train(
  afinn.training.predictors,
  afinn.training$is_NonAuthentic,
  method = 'svmRadial',
  trControl = ksvm.trainControl,
  tuneGrid = data.table(
    sigma = afinn.training.sigmas,
    C = 1
  )
)
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

```
# Create the matrix of predictors
nrc.training.predictors <- makePredictors(nrc.training)

# Create set of values for tuning sigma
nrc.training.sigmas <- sigest(nrc.training.predictors, frac = 1)
nrc.training.sigmas <- seq(
  nrc.training.sigmas["90%"] * 0.75,
  nrc.training.sigmas["10%"] * 1.25,
  length.out = 10
)

# Train the model
nrc.ksvm.model <- train(
  nrc.training.predictors,
  nrc.training$is_NonAuthentic,
  method = 'svmRadial',
  trControl = ksvm.trainControl,
  tuneGrid = data.table(
    sigma = nrc.training.sigmas,
    C = 1
  )
)
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

```
# Create the matrix of predictors
vad.training.predictors <- makePredictors(vad.training)

# Create the set of values for tuning sigma
vad.training.sigmas <- sigest(vad.training.predictors, frac = 1)
vad.training.sigmas <- seq(
  vad.training.sigmas["90%"] * 0.75,
  vad.training.sigmas["10%"] * 1.25,
  length.out = 10
)

# Train the model
vad.ksvm.model <- train(
  vad.training.predictors,
  vad.training$is_NonAuthentic,
  method = 'svmRadial',
  trControl = ksvm.trainControl,
  tuneGrid = data.table(
    sigma = vad.training.sigmas,
    C = 1
  )
)
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

```
# Stop and de-register parallel computing
stopCluster(cl)
registerDoSEQ()
remove(cl)
```

Now with our models built, we can check the accuracy achieved with the training set.

```
# Afinn SVM
confusionMatrix(fitted(afinn.ksvm.model$finalModel), afinn.training$is_NonAuthentic)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE  TRUE
##      FALSE 14744  4032
##      TRUE   1626 10884
##
##                Accuracy : 0.819
##                  95% CI : (0.815, 0.823)
##     No Information Rate : 0.523
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.635
##
##  Mcnemar's Test P-Value : <2e-16
##
```

```
##             Sensitivity : 0.901
##             Specificity : 0.730
##          Pos Pred Value : 0.785
##          Neg Pred Value : 0.870
##              Prevalence : 0.523
##          Detection Rate : 0.471
##    Detection Prevalence : 0.600
##       Balanced Accuracy : 0.815
##
##        'Positive' Class : FALSE
##
```

```
# 81.9%

# NRC SVM
confusionMatrix(fitted(nrc.ksvm.model$finalModel), nrc.training$is_NonAuthentic)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE  TRUE
##      FALSE 15830  2594
##      TRUE    906 12526
##
##                Accuracy : 0.89
##                  95% CI : (0.887, 0.894)
##     No Information Rate : 0.525
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.779
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.946
##             Specificity : 0.828
##          Pos Pred Value : 0.859
##          Neg Pred Value : 0.933
##              Prevalence : 0.525
##          Detection Rate : 0.497
##    Detection Prevalence : 0.578
##       Balanced Accuracy : 0.887
##
##        'Positive' Class : FALSE
##
```

```
# 89%

# NRC VAD SVM
confusionMatrix(fitted(vad.ksvm.model$finalModel), vad.training$is_NonAuthentic)
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction FALSE  TRUE
##      FALSE 15604  3317
##      TRUE   1139 11851
##
##                Accuracy : 0.86
##                  95% CI : (0.857, 0.864)
##     No Information Rate : 0.525
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.718
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.932
##             Specificity : 0.781
##          Pos Pred Value : 0.825
##          Neg Pred Value : 0.912
##              Prevalence : 0.525
##          Detection Rate : 0.489
##    Detection Prevalence : 0.593
##       Balanced Accuracy : 0.857
##
##        'Positive' Class : FALSE
##
```

```
# 86.1%
```

These accuracies are not as good as the RF models above, which truthfully surprised me. That said, it's possible that these accuracies are more consistent and will hold out better when using the testing set later.

Ensemble Model Ensemble modeling presents some limitations, especially with this dataset. The relatively small size of the dataset, along with how limiting lexicons can be, makes it unlikely that every model could be applied to every article in the original data. Due to the joining process required to build the models, an article is inherently excluded from the dataset if it contains no matching rows to join to a particular lexicon, e.g. Afinn. The ensemble model will have to account for sparse data - i.e. for every article in the original data, there will be 6 predicted outcomes, but only a few will be true.

Our ensemble model will be able to accommodate this by grabbing all the unique articles from our training data along with the `is_NonAuthentic` flag. Next, we can make the predictions and create a matrix of predicted outcomes by left-joining them. Using a "voting" scheme, we will simply take the prediction that has received the most votes out of 6 predictions (some of which may be *NA*). As we did with our naive approach, we can guess in the event of a tie.

```
# Create a container for our models, gathering all articles in the training set
ensemble <- rbind(
  afinn.training[, list(title, is_NonAuthentic)],
  nrc.training[, list(title, is_NonAuthentic)],
  vad.training[, list(title, is_NonAuthentic)]
)

# Take only the unique articles
ensemble <- unique(ensemble)
```

```
# Create data.tables from the training data with a column for their respective
# predictions
afinn.rf <- cbind(
  afinn.training,
  afinn.rf = predict(afinn.rf.model, makePredictors(afinn.training))
)
nrc.rf <- cbind(
  nrc.training,
  nrc.rf = predict(nrc.rf.model, makePredictors(nrc.training))
)
vad.rf <- cbind(
  vad.training,
  vad.rf = predict(vad.rf.model, makePredictors(vad.training))
)
afinn.ksvm <- cbind(
  afinn.training,
  afinn.ksvm = predict(afinn.ksvm.model, makePredictors(afinn.training))
)
nrc.ksvm <- cbind(
  nrc.training,
  nrc.ksvm = predict(nrc.ksvm.model, makePredictors(nrc.training))
)
vad.ksvm <- cbind(
  vad.training,
  vad.ksvm = predict(vad.ksvm.model, makePredictors(vad.training))
)

# Remove columns not needed for this step
afinn.rf <- afinn.rf[, list(title, afinn.rf)]
nrc.rf <- nrc.rf[, list(title, nrc.rf)]
vad.rf <- vad.rf[, list(title, vad.rf)]
afinn.ksvm <- afinn.ksvm[, list(title, afinn.ksvm)]
nrc.ksvm <- nrc.ksvm[, list(title, nrc.ksvm)]
vad.ksvm <- vad.ksvm[, list(title, vad.ksvm)]

# Set keys
setkey(ensemble, title)
setkey(afinn.rf, title)
setkey(nrc.rf, title)
setkey(vad.rf, title)
setkey(afinn.ksvm, title)
setkey(nrc.ksvm, title)
setkey(vad.ksvm, title)

# a series of left-joins
ensemble <- afinn.rf[ensemble]
ensemble <- nrc.rf[ensemble]
ensemble <- vad.rf[ensemble]
ensemble <- afinn.ksvm[ensemble]
ensemble <- nrc.ksvm[ensemble]
ensemble <- vad.ksvm[ensemble]

# We can look at the matrix we've created
```

```
ensemble[, afinn.rf:vad.ksvm]
```

```
##        afinn.rf nrc.rf vad.rf afinn.ksvm nrc.ksvm vad.ksvm
##     1:    FALSE  FALSE  FALSE      FALSE    FALSE    FALSE
##     2:    FALSE  FALSE  FALSE      FALSE    FALSE    FALSE
##     3:    FALSE  FALSE  FALSE      FALSE    FALSE    FALSE
##     4:    FALSE  FALSE  FALSE      FALSE    FALSE    FALSE
##     5:    FALSE  FALSE  FALSE      FALSE    FALSE    FALSE
##    ---
## 31907:    FALSE  FALSE   TRUE      FALSE    FALSE    FALSE
## 31908:     <NA>  FALSE   TRUE       <NA>    FALSE    FALSE
## 31909:     <NA>   TRUE   TRUE       <NA>    FALSE    FALSE
## 31910:     <NA>   TRUE   TRUE       <NA>    FALSE    FALSE
## 31911:     TRUE   TRUE   TRUE       TRUE     TRUE     TRUE
```

```
# Take the columns of predictions, convert them to a matrix of
# boolean values, then take the mean of each row.
ensemble[
  ,
  ensemble.mean := rowMeans(do.call(
    cbind,
    lapply(ensemble[, afinn.rf:vad.ksvm], as.logical)
  ), na.rm = TRUE)
]

# Convert the means above to predictions as a factor
# Predictions > 0.5 align with predicting is_NonAuthentic = TRUE
ensemble[
  ensemble.mean > 0.5,
  ensemble := "TRUE",
]

# Predictions < 0.5 align with predicting is_NonAuthentic = FALSE
ensemble[
  ensemble.mean < 0.5,
  ensemble := "FALSE",
]

# If the prediction is exactly 0.5, use naive guessing
ensemble[
  ensemble.mean == 0.5,
  ensemble := sample(c("TRUE", "FALSE"), .N, replace = TRUE),
]

# Make this column a factor for use in confusionMatrix
ensemble[, ensemble := as.factor(ensemble)]

# See the results
confusionMatrix(ensemble$ensemble, ensemble$is_NonAuthentic)
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction FALSE  TRUE
##      FALSE 16180  2431
##      TRUE    563 12737
##
##                Accuracy : 0.906
##                  95% CI : (0.903, 0.909)
##     No Information Rate : 0.525
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.811
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.966
##             Specificity : 0.840
##          Pos Pred Value : 0.869
##          Neg Pred Value : 0.958
##              Prevalence : 0.525
##          Detection Rate : 0.507
##    Detection Prevalence : 0.583
##       Balanced Accuracy : 0.903
##
##        'Positive' Class : FALSE
##
```

```
# 90.6%
```

Ensemble models are as accurate as the mean of the individual accuracies for each of our six models, which makes sense. When applied to our testing dataset, it will remain to be seen if this model is consistently accurate.

Results

Random Forests In comparison with the training set, our Random Forest models do poorly on the testing set. The results are still very good, however. We shall see below that, as a matter of fact, the RF models outperformed the SVM models. According to our results, the NRC lexicon yielded the best results for our RF approach.

```
## Make final predictions/measure Acc
# Afinn RF
confusionMatrix(predict(afinn.rf.model, afinn.testing), afinn.testing$is_NonAuthentic)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE  3759 1099
##      TRUE    413 3286
##
##                Accuracy : 0.823
##                  95% CI : (0.815, 0.831)
##     No Information Rate : 0.512
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.648
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.901
##             Specificity : 0.749
##          Pos Pred Value : 0.774
##          Neg Pred Value : 0.888
##              Prevalence : 0.488
##          Detection Rate : 0.439
##    Detection Prevalence : 0.568
##       Balanced Accuracy : 0.825
##
##        'Positive' Class : FALSE
##
```

```
# 82.3%

# NRC RF
confusionMatrix(predict(nrc.rf.model, nrc.testing), nrc.testing$is_NonAuthentic)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE  3920  578
##      TRUE    331 3864
##
##                Accuracy : 0.895
```

```
##                  95% CI : (0.889, 0.902)
##     No Information Rate : 0.511
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.791
##
##  Mcnemar's Test P-Value : 3.37e-16
##
##             Sensitivity : 0.922
##             Specificity : 0.870
##          Pos Pred Value : 0.871
##          Neg Pred Value : 0.921
##              Prevalence : 0.489
##          Detection Rate : 0.451
##    Detection Prevalence : 0.517
##       Balanced Accuracy : 0.896
##
##        'Positive' Class : FALSE
##
```

```
# 89.6%

# NRC VAD RF
confusionMatrix(predict(vad.rf.model, vad.testing), vad.testing$is_NonAuthentic)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE  3813  626
##      TRUE    438 3828
##
##                Accuracy : 0.878
##                  95% CI : (0.871, 0.885)
##     No Information Rate : 0.512
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.756
##
##  Mcnemar's Test P-Value : 9.88e-09
##
##             Sensitivity : 0.897
##             Specificity : 0.859
##          Pos Pred Value : 0.859
##          Neg Pred Value : 0.897
##              Prevalence : 0.488
##          Detection Rate : 0.438
##    Detection Prevalence : 0.510
##       Balanced Accuracy : 0.878
##
##        'Positive' Class : FALSE
##
```

```
# 87.7%
```

Support Vector Machines The KSVM model results predicting against the training set are far closer to their counterparts when predicting for the testing set. We again see below that NRC lexicon gave us the best results with this kind of model.

```
# Afinn SVM
confusionMatrix(predict(afinn.ksvm.model, makePredictors(afinn.testing)), afinn.testing$is_NonAuthentic
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE  3745 1096
##      TRUE    427 3289
##
##                Accuracy : 0.822
##                  95% CI : (0.814, 0.83)
##     No Information Rate : 0.512
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.645
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.898
##             Specificity : 0.750
##          Pos Pred Value : 0.774
##          Neg Pred Value : 0.885
##              Prevalence : 0.488
##          Detection Rate : 0.438
##    Detection Prevalence : 0.566
##       Balanced Accuracy : 0.824
##
##        'Positive' Class : FALSE
##
```

```
# 82.2%

# NRC SVM
confusionMatrix(predict(nrc.ksvm.model, makePredictors(nrc.testing)), nrc.testing$is_NonAuthentic)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE  3971  787
##      TRUE    280 3655
##
##                Accuracy : 0.877
##                  95% CI : (0.87, 0.884)
```

```
##     No Information Rate : 0.511
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.755
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.934
##             Specificity : 0.823
##          Pos Pred Value : 0.835
##          Neg Pred Value : 0.929
##              Prevalence : 0.489
##          Detection Rate : 0.457
##    Detection Prevalence : 0.547
##       Balanced Accuracy : 0.878
##
##        'Positive' Class : FALSE
##
```

```
# 87.7%

# NRC VAD SVM
confusionMatrix(predict(vad.ksvm.model, makePredictors(vad.testing)), vad.testing$is_NonAuthentic)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE  3964  902
##      TRUE    287 3552
##
##                Accuracy : 0.863
##                  95% CI : (0.856, 0.871)
##     No Information Rate : 0.512
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.728
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.932
##             Specificity : 0.797
##          Pos Pred Value : 0.815
##          Neg Pred Value : 0.925
##              Prevalence : 0.488
##          Detection Rate : 0.455
##    Detection Prevalence : 0.559
##       Balanced Accuracy : 0.865
##
##        'Positive' Class : FALSE
##
```

```
# 86.6%
```

Ensemble Model We can again make all 6 sets of predictions, and gather an ensemble model. The results are not much better than the results of the individual models, but this isn't unexpected since the accuracies of each individual model are not hugely different.

```
# Ensemble model for testing dataset
# Create a container for our models, gathering all articles in the training set
ensemble <- rbind(
  afinn.testing[, list(title, is_NonAuthentic)],
  nrc.testing[, list(title, is_NonAuthentic)],
  vad.testing[, list(title, is_NonAuthentic)]
)

# Take only the unique articles
ensemble <- unique(ensemble)

# Create data.tables from the testing data with a column for their respective
# predictions
afinn.rf <- cbind(
  afinn.testing,
  afinn.rf = predict(afinn.rf.model, makePredictors(afinn.testing))
)
nrc.rf <- cbind(
  nrc.testing,
  nrc.rf = predict(nrc.rf.model, makePredictors(nrc.testing))
)
vad.rf <- cbind(
  vad.testing,
  vad.rf = predict(vad.rf.model, makePredictors(vad.testing))
)
afinn.ksvm <- cbind(
  afinn.testing,
  afinn.ksvm = predict(afinn.ksvm.model, makePredictors(afinn.testing))
)
nrc.ksvm <- cbind(
  nrc.testing,
  nrc.ksvm = predict(nrc.ksvm.model, makePredictors(nrc.testing))
)
vad.ksvm <- cbind(
  vad.testing,
  vad.ksvm = predict(vad.ksvm.model, makePredictors(vad.testing))
)

# Remove columns not needed for this step
afinn.rf <- afinn.rf[, list(title, afinn.rf)]
nrc.rf <- nrc.rf[, list(title, nrc.rf)]
vad.rf <- vad.rf[, list(title, vad.rf)]
afinn.ksvm <- afinn.ksvm[, list(title, afinn.ksvm)]
nrc.ksvm <- nrc.ksvm[, list(title, nrc.ksvm)]
vad.ksvm <- vad.ksvm[, list(title, vad.ksvm)]

# Set keys
```

```
setkey(ensemble, title)
setkey(afinn.rf, title)
setkey(nrc.rf, title)
setkey(vad.rf, title)
setkey(afinn.ksvm, title)
setkey(nrc.ksvm, title)
setkey(vad.ksvm, title)

# a series of left-joins
ensemble <- afinn.rf[ensemble]
ensemble <- nrc.rf[ensemble]
ensemble <- vad.rf[ensemble]
ensemble <- afinn.ksvm[ensemble]
ensemble <- nrc.ksvm[ensemble]
ensemble <- vad.ksvm[ensemble]

# We can look at the matrix we've created
ensemble[, afinn.rf:vad.ksvm]
```

```
##       afinn.rf nrc.rf vad.rf afinn.ksvm nrc.ksvm vad.ksvm
##    1:     TRUE   TRUE   TRUE       TRUE     TRUE     TRUE
##    2:     TRUE   TRUE   TRUE       TRUE     TRUE     TRUE
##    3:     <NA>   TRUE   TRUE       <NA>     TRUE     TRUE
##    4:     TRUE   TRUE   TRUE       TRUE     TRUE     TRUE
##    5:    FALSE  FALSE  FALSE      FALSE    FALSE    FALSE
##   ---
## 8701:    FALSE  FALSE  FALSE      FALSE    FALSE    FALSE
## 8702:     <NA>  FALSE  FALSE       <NA>    FALSE    FALSE
## 8703:     <NA>  FALSE  FALSE       <NA>    FALSE    FALSE
## 8704:    FALSE  FALSE  FALSE      FALSE    FALSE    FALSE
## 8705:    FALSE  FALSE  FALSE      FALSE    FALSE    FALSE
```

```
# Take the columns of predictions, convert them to a matrix of
# boolean values, then take the mean of each row.
ensemble[
  ,
  ensemble.mean := rowMeans(do.call(
    cbind,
    lapply(ensemble[, afinn.rf:vad.ksvm], as.logical)
  ), na.rm = TRUE)
]

# Convert the means above to predictions as a factor
# Predictions > 0.5 align with predicting is_NonAuthentic = TRUE
ensemble[
  ensemble.mean > 0.5,
  ensemble := "TRUE",
]

# Predictions < 0.5 align with predicting is_NonAuthentic = FALSE
ensemble[
  ensemble.mean < 0.5,
  ensemble := "FALSE",
```

```
]

# If the prediction is exactly 0.5, use naive guessing
ensemble[
  ensemble.mean == 0.5,
  ensemble := sample(c("TRUE", "FALSE"), .N, replace = TRUE),
]

# Make this column a factor for use in confusionMatrix
ensemble[, ensemble := as.factor(ensemble)]

# See the results
confusionMatrix(ensemble$ensemble, ensemble$is_NonAuthentic)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE  4001  868
##      TRUE    250 3586
##
##                Accuracy : 0.872
##                  95% CI : (0.864, 0.879)
##     No Information Rate : 0.512
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.744
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.941
##             Specificity : 0.805
##          Pos Pred Value : 0.822
##          Neg Pred Value : 0.935
##              Prevalence : 0.488
##          Detection Rate : 0.460
##    Detection Prevalence : 0.559
##       Balanced Accuracy : 0.873
##
##        'Positive' Class : FALSE
##
```

```
#  87.1
```

Conclusion

This analysis was only a superficial exploration of using machine learning to classify Non-Authentic News. Besides bigram analysis and natural language processing, there are a great many other techniques to be applied. Many news outlets and social media companies are engaging in similar research on their own content in an effort to respond to the needs of their consumers as well as to be responsible for the platform they provide their content creators.

This report suffers most from limitations in the source data itself. Conclusions should be interpreted with a grain of salt - the original data selection was skewed and biased, and the aggregated source data was not large enough to consider this an in-depth analysis. It would be possible to use these approaches as a starting point to explore these and other techniques if one had a larger, more robust, and objectively classified set of data.

Citations

1. Nielsen F Årup (2011) A New ANEW: Evaluation of a Word List for Sentiment Analysis in Microblogs. CoRR abs/1103.2903:

2. Mohammad SM, Turney PD (2013) CROWDSOURCING a WORD–EMOTION ASSOCIATION LEXICON. Computational Intelligence 29:436–465. https://doi.org/10.1111/j.1467-8640.2012.00460.x

3. Mohammad SM (2018) Obtaining Reliable Human Ratings of Valence, Arousal, and Dominance for 20,000 EnglishWords