Experiments on Natural Language Processing and Prediction of Sentiment Labelled Sentences

Assignment Submission for Course CP8305 Instructed by Dr. Cherie Ding

Richard Wen

Ryerson University

April 13, 2017

Contents

1 Objectives

This assignment was provided by Dr. Cherie Ding for the CP8305 Knowledge Discovery course at Ryerson University. The purposes of the assignment was to:

1. Identify a practical dataset from the University of California, Irvine (UCI) Machine Learning Repository (Lichman 2013) or Knowledge Discovery in Databases (KDD) Cup Archives (SIGKDD 2013)

2. Identify a machine learning problem with the chosen dataset in (1)

3. Apply various machine learning algorithms to the problem in (2) to find algorithms that can solve (2) well and provide insight into the data in (1)

2 Data

The data chosen for this assignment was the Sentiment Labelled Sentences (SLS) Dataset donated on May 30, 2015 and downloaded from the UCI Machine Learning Repository (Kotzias et al. 2015). There are 3 text files (amazon_cells_labelled.txt, imdb_labelled.txt, yelp_labelled.txt) with a combined total of 3000 instances, absent of missing values. Each file consists of 2 attributes with the first attribute being sentences from (string type) and the second being a binary class of either 0 for negative sentiment or 1 for positive sentiment (numeric type). The data in each file had attributes separated by a mixture of inconsistent spaces and tabs, and instances separated by rows. An example of the first 5 rows are shown in Table 1[1]. Sentences were extracted by Kotzias et al. (2015) from imdb.com, amazon.com, and yelp.com. These websites represent a movie database, online retailer, and a online business directory with crowd-sourced reviews, respectively. The SLS files are summarized in Table 2[2].

[1] In the table, (tab) indicates that a tab character was present in the data sample.

[2] For clarification, a double indicates type numeric and a character indicates type string.

Table 1: SLS Dataset Example for amazon Data

Line	Sample
1	So there is no way for me to plug it in here in the US unless I go by a converter. (tab) 0
2	Good case, Excellent value. (tab) 1
3	Great for the jawbone. (tab) 1
4	Tied to charger for conversations lasting more than 45 minutes.MAJOR PROBLEMS!! (tab) 0
5	The mic is great. (tab) 1

Table 2: SLS Dataset Summary

File	Instances	Attributes
amazon_cells_labelled.txt	1000	2
imdb_labelled.txt	1000	2
yelp_labelled.txt	1000	2

3 Problem

The goal of the SLS dataset was to predict the values in attribute 2 (sentiment) which contain 0 or 1 for negative and positive sentiment respectively. The binary values of attribute 2 defined the SLS dataset as a classification problem. In order to predict sentiment, sentences from three different websites (imdb.com, amazon.com, and yelp.com) were given as the explanatory attribute 1. The text values of attribute 1 further defined the problem as a Natural Language Processing (NLP)[3] problem where the attribute values were unstructured, and required pre-processing before the machine could read and learn to model the SLS data. For simplicity, attribute 1 was referred to as the sentence attribute and attribute 2 was referred to as the sentiment attribute. The problem was then known as a supervised classification[4] problem for the sentiment attribute that required NLP of the sentence attribute. The following sections more formally define the problem framework.

[3] NLP seeks to extract meaning from textual data involving language communication with low level tasks such as identification of individual words and high level tasks such as spelling correction (Nadkarni, Ohno-Machado, and Chapman 2011)

[4] A supervised classification, in this case, refers to the target sentiment classes being known, and the target sentiment values being discrete rather than continuous or numeric in meaning

3.1 Sentiment Attribute

The sentiment attribute was the second attribute in the SLS dataset. It is the target vector $y^{(n)}$ containing binary values of either 0 or 1 given n instances as seen in Equation (1):

$$y^{(n)} \in \{0, 1\} \tag{1}$$

3.2 Sentence Attribute

The sentence attribute was the first attribute in the SLS dataset. It is the raw text data $x^{(n)}$ given n instances such that it contains k number of words $w^{(k,n)}$ (separated by spaces[5]), where the word lengths[6] l_w of $w^{(k,n)}$ and k are less than the length l_x of the raw texts $x^{(n)}$ as seen in Equation (2):

$$w^{(k,n)} \in x^{(n)} \mid \ \ 0 < l_w^{(k)} \leq l_x^{(n)} \ \ and \ \ 0 < k \leq l_x^{(n)} \tag{2}$$

3.3 Measures of Prediction Quality

The classification problem given $x^{(n)}$ as the explanatory data and $y^{(n)}$ as the target classes was to obtain measurement values that define an algorithm to predict $y^{(n)}$ well. Predicted classes $y^{(n)}_{pred}$ are obtained using $x^{(n)}$ training features seen in Equation (3):

$$y^{(n)}_{pred} = f(x^{(n)}) \tag{3}$$

The classification prediction quality used measurements that were based on the f_{eq} counts of $y^{(n)}_{pred}$ that

[5] For example, the text instance “hello goodbye now”, contain 3 words “hello”, “goodbye”, and “now” that are separated by spaces

[6] Word lengths and text lengths are measured in the number of characters, excluding spaces and symbols that define punctuation, which are more specifically the number of alphanumeric characters in this case (e.g. the word “apple” has a length of 5 alphanumeric characters and the text “!*–” has a length of 0 characters containing non-alphanumeric characters)

were equal to $y^{(n)}$ given the total number of instances N expressed in Equation (4):

$$f_{eq} = \sum_{n=1}^{N} y_{eq} \mid y_{eq} = \begin{cases} 1: & \text{if } y_{pred}^{(n)} = y^{(n)} \\ 0: & \text{otherwise} \end{cases} \tag{4}$$

Accuracy measurements were defined as a maximization problem, where higher values are better and lower values are worse. Error measurements were defined as a minimization problem, where lower values are better and higher values are worse.

An accuracy measurement f_{acc}, where C_{acc} is a constant representing the best prediction quality, increases the more times $y_{pred}^{(n)}$ is equal to $y^{(n)}$ given f_{eq} as seen in Equation (5):

$$\lim_{f_{acc} \to C_{acc}} f_{acc} \;\; as \;\; f_{eq} \to \infty \quad \mid f_{acc} = f(y_{pred}^{(n)}, y^{(n)}) \tag{5}$$

An error measurement f_{err}, where C_{err} is a constant representing the worse prediction quality, decreases the more times $y_{pred}^{(n)}$ is equal to $y^{(n)}$ given f_{eq} as seen in Equation (6):

$$\lim_{f_{err} \to C_{err}} f_{err} \;\; as \;\; f_{eq} \to \infty \quad \mid f_{err} = f(y_{pred}^{(n)}, y^{(n)}) \tag{6}$$

4 Methods

The methods described in this section attempted to experiment with several solutions to the classification problem defined in Section 3. The unstructured nature of the sentence attribute presented required preprocessing to create features which are then further processed to search for adequately useful features (selection). These features were then randomly split into approximately equal number of instances for cross validation training sets. These training sets were then used as input for a selected number of algorithms and evaluated for prediction quality. A summary of the methods is shown in Figure 1. See

Appendix A for the R code.

4.1 Preprocessing

The preprocessing steps involved getting the data ready for the machine learning algorithms to train on. This involved creating features from the instances in the sentence attribute and selecting only the features that are measured to be useful in terms of predicting the target sentiment attribute. Furthermore, the data was also split into cross validation training sets and used as the resulting training data for the machine learning algorithms.

4.1.1 Parsing

The data was defined as a tab delimited text file that held string based text as the first sentence attribute and numeric binary numbers as the second sentiment attribute. The unstructured nature of the sentence attribute made parsing the file less straight-forward as instances in the sentence attribute were not quoted and could contain any number of tab characters. The data was:

1. Parsed line by line
2. Cleaned by removing the last occurence of tab characters and punctuation
3. Extracted for sentiment and sentence instances where the last character was defined as the sentiment, and the rest of the text was defined as the sentence

4.1.2 Features

The features[7] were created using a simple bag of words model. The bag of words model constructs a feature for each unique word in the sentence attribute and counts the occurrence per sentence instance in the SLS dataset (Nadkarni, Ohno-Machado, and Chapman 2011). Each k word feature $w^{(k)}$ is a count

[7] Features in this report are similar to attributes, except that features refer to the machine-constructed columns to differ from the original sentence and sentiment attributes

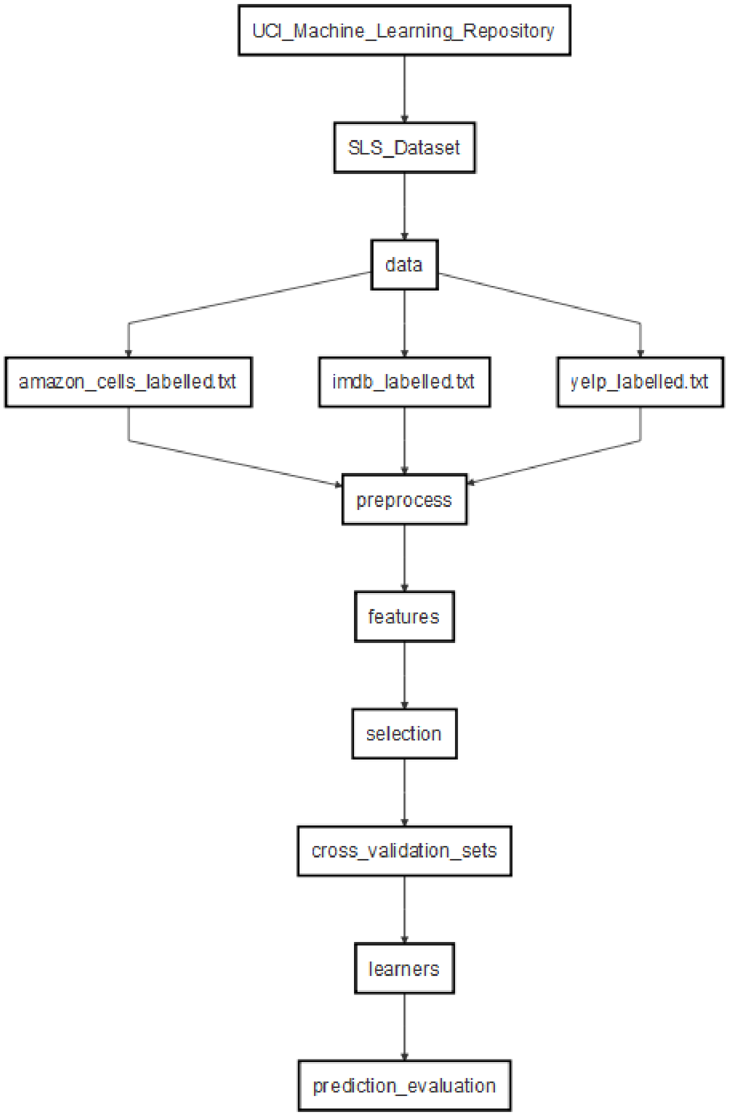


Figure 1: Flowchart of Methods

Table 3: Example of Word Features

Word-1	Word-2	Word-k
2	5	10
5	2	5
10	10	2

of the occurrence of k unique words in the sentence attribute $x^{(n)}$ given n instances. For clarification, all words were considered, including stop words[8], as these would be removed in the feature selection process if they were measured as being not very useful - thus, the possibility of them being useful was considered. An example of word features is given in Table 3. Feature construction was applied using the *text2vec* R package (Selivanov 2016).

4.2 Algorithms

The algorithms applied to solve the classification problem involved optimizing features and a measure of prediction quality based on training and testing on cross validation folds.

4.2.1 Feature Selection

Feature selection was used to filter the word features $w^{(k)}$ for the most important word features. A measure of importance was defined by using the Random Forest (RF) algorithm. The RF algorithm ensembles multiple decision trees by randomly subsetting portions of the complete data to construct each tree (Breiman 2001). For each portion of data subsetted, there were remaining portions that were used to calculate an Out Of Bag (OOB) error by using the constructed decison trees to predict on the remaining data portions. These remaining portions were referred to as the OOB samples, and were used in calculating the feature importances. Each k word feature $w^{(k)}$ was then calculated for feature importances by randomly permuting prediction errors on the OOB samples, and comparing their differences on the permuted and un-permuted errors. These feature importances quantify the usefulness

[8] Stop words are commonly used words in the language that are often, but not always, removed as they may not hold useful information, but there is a possibility that stop words may be useful

of word features for the classification problem, where higher values represent more useful features and lower values represent less useful features. Initially, features in which the occurrence of the word is less than 0.1% of the total number of occurrences were removed, followed by strict removal of features that were slightly to highly correlated, where features with correlation values of greater than 0.05 were removed. This enabled independence assumptions to be better satisfied as there were an abundance of features. The feature selection then filtered for the top 10 features with the highest feature importance based on a random forest classifier with 500 trees. The feature selection algorithm was applied using the *caret* package (Kuhn 2016).

4.2.2 Cross Validation

The data from the feature selection was split into a standard 10-fold cross validation scenario. In the 10-fold cross validation scenario, the data was randomly split into 10 equal parts, trained on 9 parts, and tested on the 1 part not in the 9 training parts until all testing parts have been tested for (Borra and Di Ciaccio 2010). Cross validation enabled prediction quality measures to evaluate the generalization of algorithms as different parts of the data were used to learn from and predict on. This generalization quality more accurately represented real-world scenarios as there are often missing portions of data, and a complete dataset is often not available as data is continuously updated. Cross validation was also applied using the *caret* R package.

4.2.3 Selected Learners

The learners selected for the experiments were standard machine learning choices that were relatively efficient and simple to run as the strict correlation removal procedure and feature selection process enabled relatively independent features of low dimensionality. The selected machine learning algorithms are the Naive Bayes Classifier (NBC), Support Vector Machine (SVM) and Random Forest (RF) from the R package *e1071* (Meyer 2017). NBC is a conditional probability model that applies Baye's theorem with the assumption that the features being trained on are independent of each other (McCallum and

Kamal 1998). RF is a model that constructs many decision trees from random subsamples of the instances and features, and ensembles them for a majority vote of the predicted class (Breiman 2001). SVM is a non-probabilistic binary model that constructs a set of hyperplanes in high dimensional space to separate target classes (Tong and Koller 2001). The selected algorithms were evaluated based on their F1 score to account for potential binary class imbalances in the sentiment attribute. The F1 score (Equation (9)) is the harmonic mean of the precision (Equation (8)) and recall (Equation (7)) given N instances, Y_s correctly predicted instances for a sentiment s, and the number of instances for a sentiment N_s. See the learners section under Appendix A for more details.

$$Precision = Y_s/N \tag{7}$$

$$Recall = Y_s/N_s \tag{8}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{9}$$

5 Results

Figure 2 shows the distribution of classes in the sentiment attribute. The results ranged from CV F1 scores of 0.26 to 0.71 per fold. Figure 3 shows the average CV F1 scores obtained from the experiments. The word feature importances are also shown in Figure 4. The top ten word features based on feature importances was selected from these options, which produced a training feature matrix of 10 columns (reduced from 5189) by 3000 rows, with each k column representing w^k word vectors.

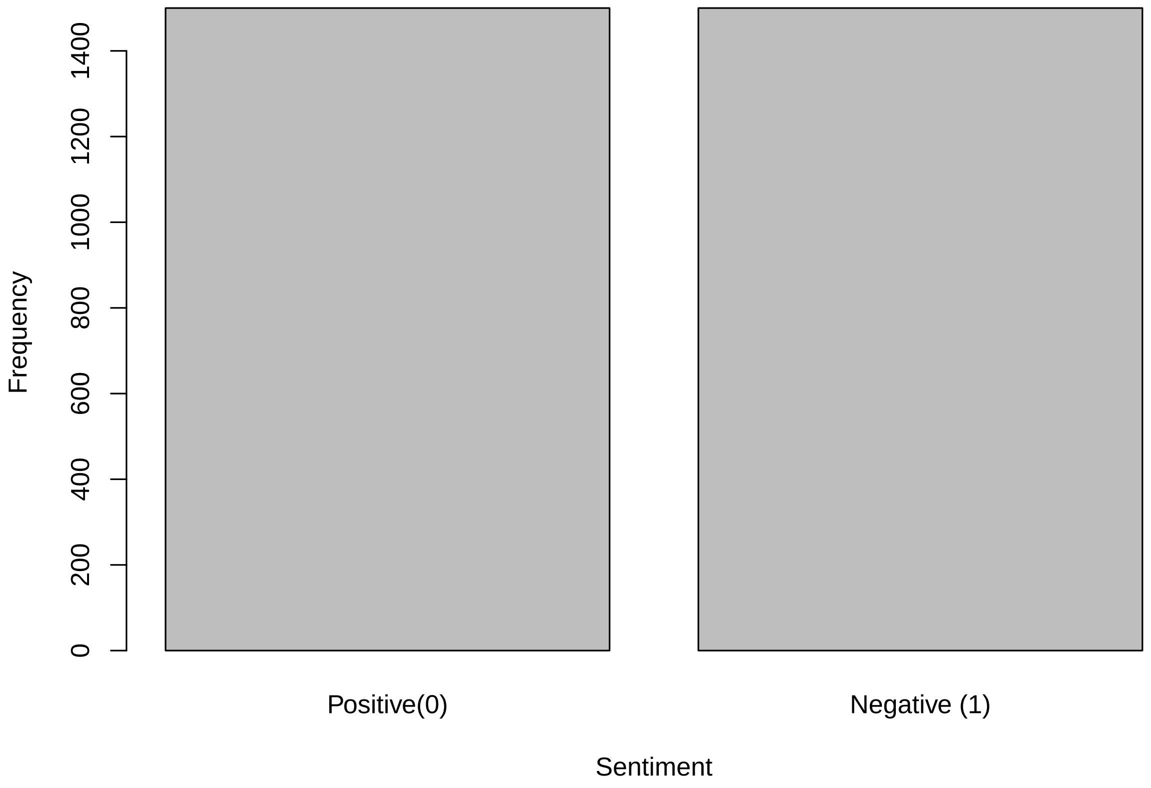


Figure 2: Sentiment Class Distribution

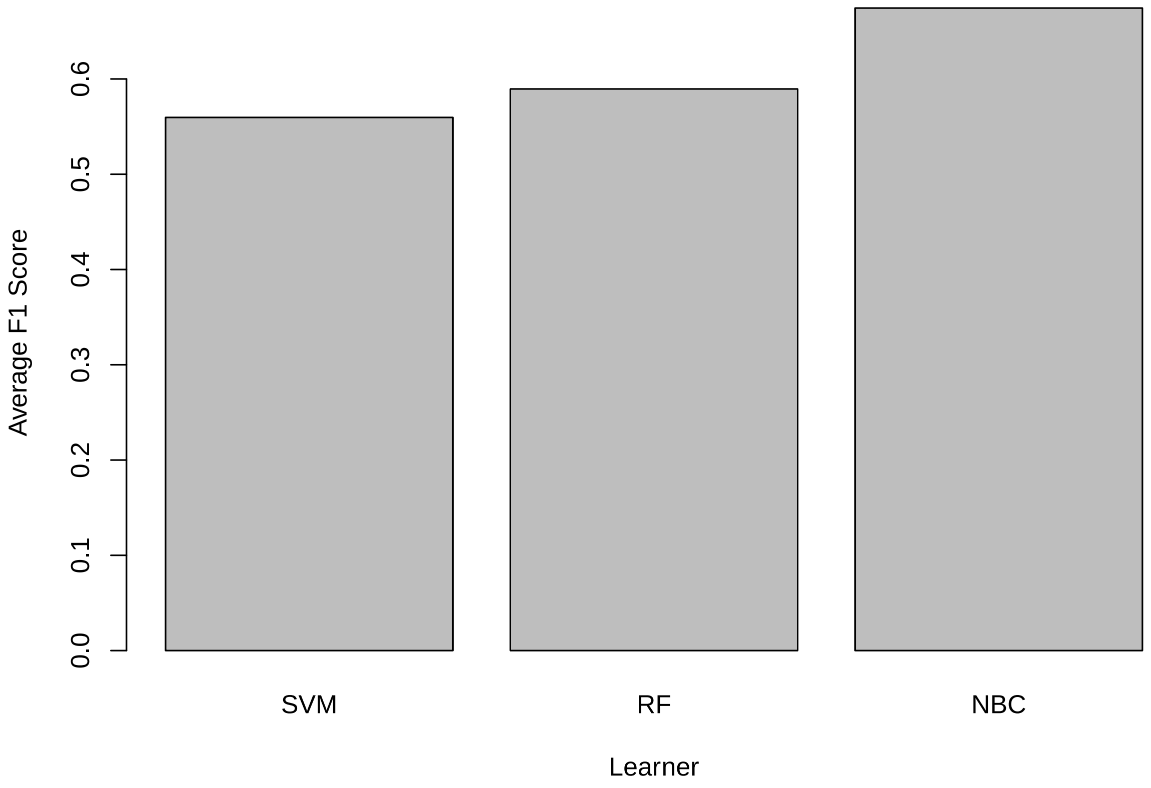


Figure 3: Average F1 Scores

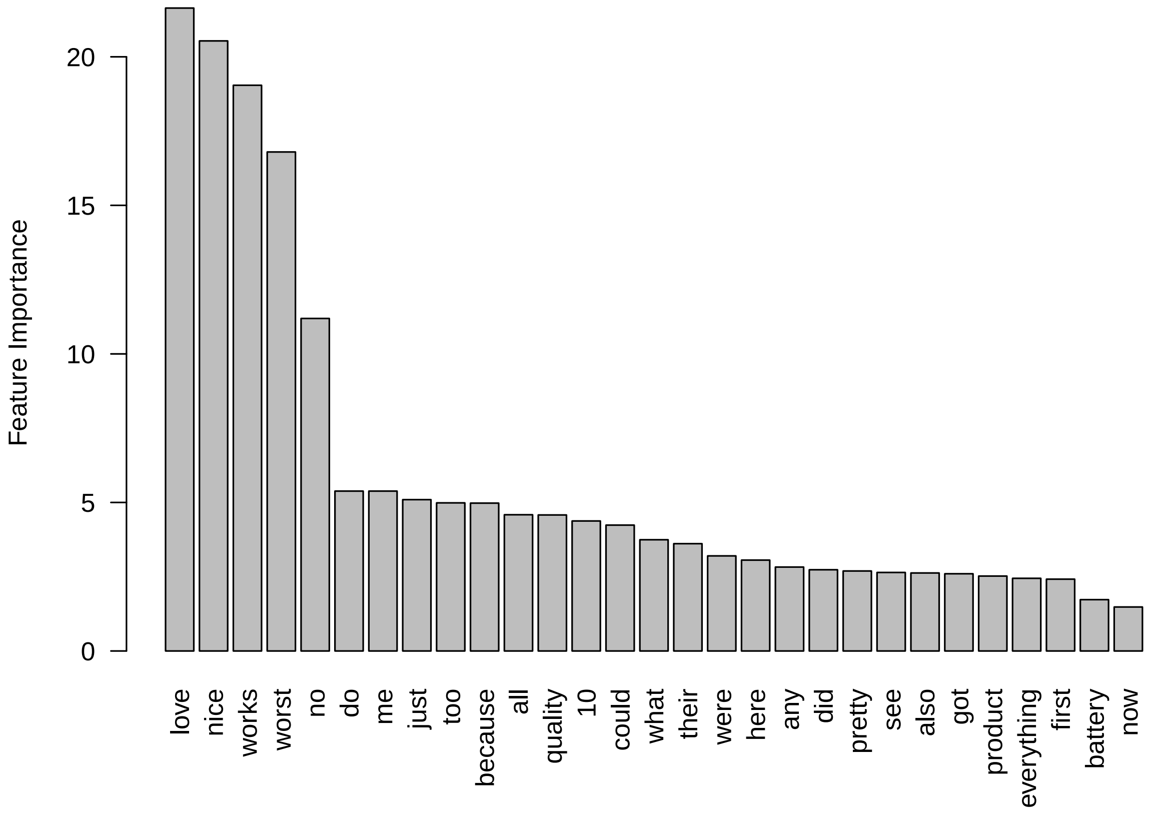


Figure 4: Feature Importances

6 Discussion and Conclusion

Both the RF and SVM algorithms performed similarly, while the NBC algorithm performed better of the three selected algorithms. However, the random forest was useful in calculating feature importances used in the feature selection process to reduce the number of dimensions on the data, as well as quantify how much more important one word feature was compared to another word feature. In particular, the word features *love, nice, works, no,* and *worst* were very useful in classifying the sentiment attribute consisting of 0 or 1s indicating negative and positive respectively. Although, the RF and SVM performed rather poorly, the results were reasonable given the large reduction in features and minimal hyperparameter tuning. This is done in exchange for interpretability of results to see what influences positive and negative sentiment without manually reviewing thousands of online reactions, or having advanced knowledge of specific machine learning algorithms. The high dimensionality of the data after processing for the bag of words model made it difficult to obtain quick and efficient methods of feature reduction, however the abundance of features made it more flexible when arbitrary rules were imposed - such as the strict correlation values imposed in the feature selection process. This led to better features for the NBC algorithm, which resulted in the highest average CV F1 score without heavy hyperparameter tuning compared to the RF and SVM. The NBC was a simple, yet efficient and effective algorithm, when features were selected to meet its assumptions of feature independence. Algorithms that can efficiently either utilize high dimensional data or reduce dimensionality with minimal information loss, but still enable the results to be interpreted would be very desirable in machine learning problems involving text based attributes that can be converted into a bag of words model.

7 References

Borra, Simone, and Agostino Di Ciaccio. 2010. “Measuring the Prediction Error. A Comparison of Cross-Validation, Bootstrap and Covariance Penalty Methods.” *Computational Statistics and Data Analysis* 54 (12). Elsevier: 2876–2989. doi:10.1016/j.csda.2010.03.004.

Breiman, Leo. 2001. “Random Forests.” *Machine Learning* 45 (1). Springer: 5–32. doi:10.1023/A:1010933404324.

Kotzias, Dimitrios, Misha Denil, De Freitas Nando, and Padhraic Smyth. 2015. “From Group to Individual Labels Using Deep Features.” *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 597–606. doi:10.1145/2783258.2783380.

Kuhn, Max. 2016. “Caret: Classification and Regression Training.” *Journal of Statistical Software* 28 (5): 1–5. https://CRAN.R-project.org/package=caret.

Lichman, Moshe. 2013. “(UCI) Machine Learning Repository.” http://archive.ics.uci.edu/ml.

McCallum, Andrew, and Nigam Kamal. 1998. “A Comparison of Event Models for Naive Bayes Text Classification.” *AAAI-98 Workshop on Learning for Text Categorization*.

Meyer, David. 2017. “E1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien.” https://cran.r-project.org/package=e1071.

Nadkarni, Prakash M, Lucila Ohno-Machado, and Wendy W Chapman. 2011. “Natural Language Processing: An Introduction.” *Journal of the American Medical Informatics Association* 18 (5). PubMed Central: 544–51. doi:10.1136/amiajnl-2011-000464.

Selivanov, Dmitriy. 2016. “Text2vec: Modern Text Mining Framework for R.” https://cran.r-project.

org/package=text2vec.

SIGKDD. 2013. “KDD Cup Archives.” www.kdd.org/kdd-cup.

Tong, Simon, and Koller. 2001. “Support Vector Machine Active Learning with Applications to Text Classification.” *Journal of Machine Learning Research*.

8 Appendix A: R Code

A0. Dependencies

The following R code installs required package dependencies if not installed and loads them into the R environment.

```r
# Dependencies ----

# (package_install) Required packages and loading
packages <- c(
  "randomForest",
  "text2vec",
  "caret",
  "e1071"
)
for (pkg in packages) {
  if (!requireNamespace(pkg, quietly=TRUE)) {
    install.packages(pkg,
                     dependencies=TRUE,
                     repos="http://cran.rstudio.com/")
  }
  library(pkg, character.only=TRUE)
}
```

A1. Data

The SLS dataset is downloaded as a .zip file into a temporary directory, unzipped, and collected for file path information stored as variable **slsFiles**. See Data section.

```r
# Data ----

# 1. Download the zipped SLS data from UCI into a temp dir
temp <- tempdir()
src <- paste0("http://archive.ics.uci.edu/ml/machine-learning-databases/",
              "00331/sentiment%20labelled%20sentences.zip")
zipped <- file.path(temp, basename(src))
download.file(src, zipped)
unzip(zipped, exdir=temp)

# 2. Obtain SLS data paths from unzipped folder
slsFolder <- file.path(temp, "sentiment labelled sentences")
slsIgnore <- file.path(slsFolder, c(".DS_Store", "readme.txt"))
slsFiles <- list.files(slsFolder, full.names=TRUE)
slsFiles <- slsFiles[!slsFiles %in% slsIgnore]
```

SLS Dataset Files

Files	Bytes
amazon_cells_labelled.txt	58226
imdb_labelled.txt	85285
yelp_labelled.txt	61320

A2. Parsing

Parsing is done to read the SLS data, given by the file path information **slsFiles** of the previous code, into a R dataframe named **ds** with columns sentence and sentiment, where each row represents the instances. The sentence column contains the explanatory text data and the sentiment column contains the binary data of 0s and 1s. See Parsing section.

```r
# Parsing ----

# 1. Read the data on a line by line basis
sls <- sapply(slsFiles, readLines)

# 2. Extract sentence and sentiment attributes
x <- c() # sentence attribute
y <- c() # sentiment attribute
for (n in sls) { # each instance n
  lx <- nchar(n) # sentence length
  xn <- substr(n, 1, lx - 2) # sentence instance
  xn <- gsub('[[:punct:] ]+',' ', xn) # remove punctuation
  yn <- as.numeric(substr(n, lx - 1, lx)) # sentiment instance
  x <- c(x, xn)
  y <- c(y, yn)
}
ds <- data.frame(sentence = x, sentiment = y, stringsAsFactors=FALSE)
```

Example of Parsed Dataframe

sentence	sentiment
So there is no way for me to plug it in here in the US unless I go by a converter	0
Good case Excellent value	1
Great for the jawbone	1
Tied to charger for conversations lasting more than 45 minutes MAJOR PROBLEMS	0
The mic is great	1

A3. Features

Features are then constructed using the ?text2vec word tokenizer to create a bag of words model. This creates variable **wnk** as a matrix where each column are the unique words in all sentence columns of the **ds** data, and each row represents the counts of the words for an instance. See Features section.

```r
# Features ----

# 0. Variables
xn <- ds$sentence

# 1. Obtain words wn for each xn using a tokenizer
wn <- itoken(xn, tolower, word_tokenizer)

# 2. Vocabulary of wk words for wn
wk <- vocab_vectorizer(create_vocabulary(wn))

# 3. Obtain word features matrix wnk given n instances and k words
wnk <- as.matrix(get_dtm(create_corpus(wn, wk)))
```

Example of bag of words matrix

love	nice	works	worst	no
0	0	0	0	1
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

A4. Selection

Feature selection was then done to reduce the dimensionality of the **wnk** bag of words matrix into the most useful features for learning. **wnk** then has features removed in which:

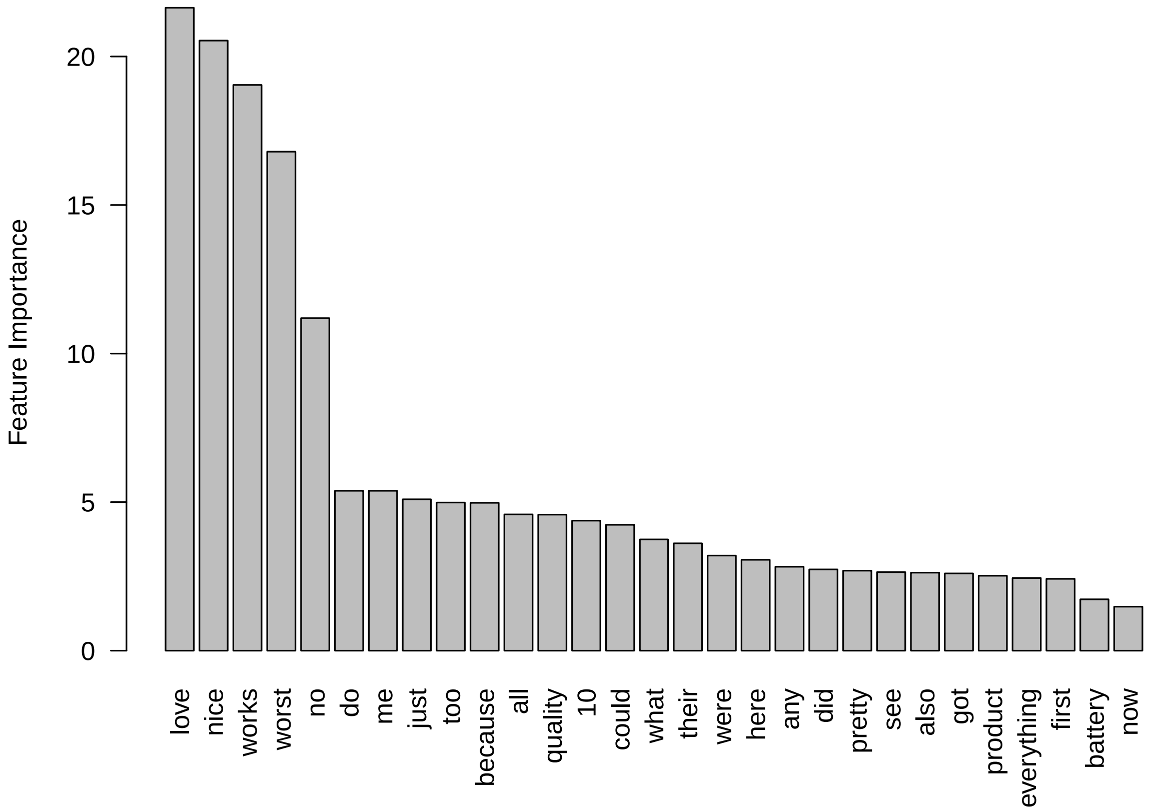
1. Words occurence is less than 0.1% of the total occurences
2. Correlation value `?caret:findCorrelation` with another feature is over 0.05
3. Random Forest variable importance measure `?randomForest::importance` is not in the top 10 highest values

See Feature Selection section.

```
# Selection ----


# 0. Variables

yn <- as.factor(ds$sentiment)


# 1. Keep only words that occur more than 0.1% of occurences

freqc <- apply(wnk, 2, sum)

wnk <- wnk[, which(freqc > sum(freqc) * 0.001)]


# 2. Remove highly correlated features over 0.05

wcor <- findCorrelation(cor(wnk), cutoff = 0.05)

wnk <- wnk[, -wcor]


# 4. Calculate RF feature importances and select top 10

rfFit <- randomForest(

  x = wnk,

  y = yn,

  importance = TRUE,
  proximity = TRUE,
  ntree = 500
)
impw <- importance(rfFit)[, 4]
impw <- impw[order(-impw)]
wnk <- wnk[, names(impw)[1:10]]
```

Plot of the Feature Importances

```
barplot(impw, names.arg = names(impw), ylab = "Feature Importance", las=2)
```



A5. Cross Validation

The row indices of the **wnk* bag of words matrix is then divided into 10 equal parts using `?caret::createFolds` to use as training and testing data for cross validation performance measures. See Cross Validation section.

```r
# Cross Validation ----


# Each element are the row ids of a 10 fold CV
cv <- createFolds(1:nrow(wnk),
                  k = 10,
                  list = TRUE,
                  returnTrain = TRUE)
```

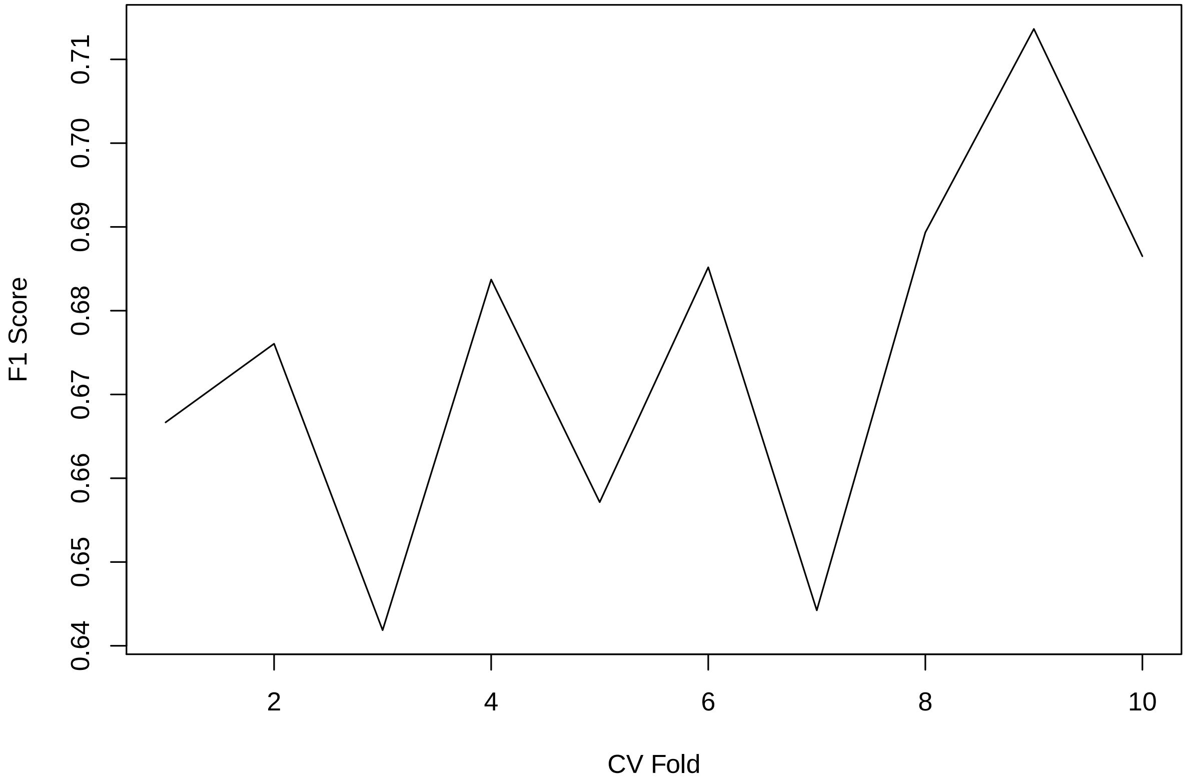
A6. Learners

Learners were constructed using `?e1071::naiveBayes`, `?e1071::svm`, and `?randomForest::randomForest`. These were trained on 9 folds of the feature selected *wnk* bag of words matrix, and tested on the 1 remaining fold. This is done until all folds have been tested for. During the testing of each fold, `?caret::precision` and `?caret::recall` is used to calculate the F1 score and stored in a **results** variable. This **results** variable holds the F1 scores for each fold for each algorithm *nbc∗*, *∗rf* and *$svm*. See Selected Learners section.

```r
# Learners ----

# 1. Set performance measures
precision <- caret::precision
recall <- caret::recall

# 2. Results list to collect Predictions
results <- list(
  svm = c(),
  rf = c(),
  nbc = c()
)

# 3. Predict on 10 fold CV
for (ids in cv) {

  # 3.1 Training data
  xtrain <- wnk[ids, ]
  ytrain <- yn[ids]

  # 3.2 Testing data
  xtest <- wnk[-ids, ]
  ytest <- yn[-ids]

  # 3.3 Create models
  nbcModel <- naiveBayes(x = wnk, y = yn, laplace = 1)
  svmModel <- svm(x = xtrain, y = ytrain, kernel = "linear")
  rfModel <- randomForest(x = xtrain, y = ytrain, ntrees = 500)

  # 3.3 Predictions
  nbcPredict <- predict(nbcModel, xtest)
  svmPredict <- predict(svmModel, xtest)
  rfPredict <- predict(rfModel, xtest)

  # 3.4 Evaluate Precision
  nbcPrecision <- precision(nbcPredict, ytest)
  svmPrecision <- precision(svmPredict, ytest)
  rfPrecision <- precision(rfPredict, ytest)

  # 3.5 Evaluate Recall
  nbcRecall <- recall(nbcPredict, ytest)
  svmRecall <- recall(svmPredict, ytest)
  rfRecall <- recall(rfPredict, ytest)

  # 3.6 Evaluate F1 Score
  nbcF1 <- (2 * nbcPrecision * nbcRecall) / (nbcPrecision + nbcRecall)
  svmF1 <- (2 * svmPrecision * svmRecall) / (svmPrecision + svmRecall)
  rfF1 <- (2 * rfPrecision * rfRecall) / (rfPrecision + rfRecall)

  # 3.7 Add F1 score of fold to results
  results$nbc <- c(results$nbc, nbcF1)
  results$svm <- c(results$svm, svmF1)
  results$rf <- c(results$rf, rfF1)
}
```

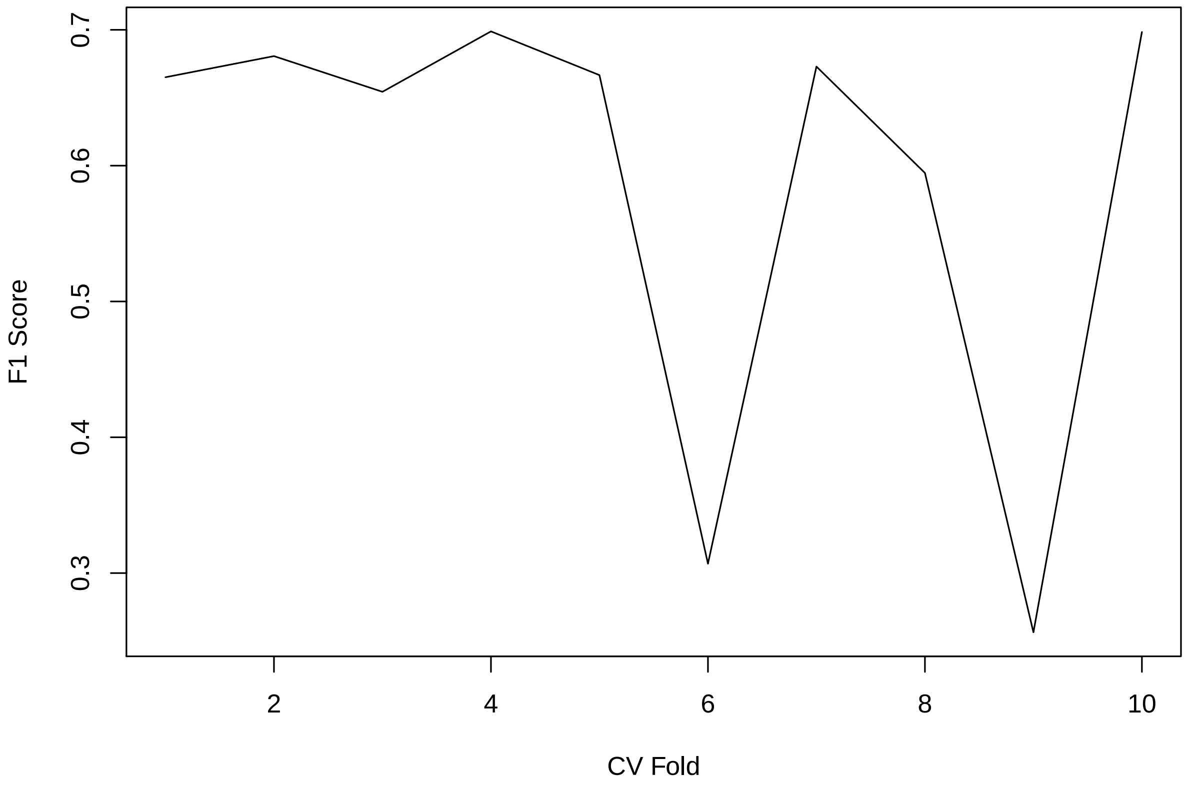
Plot of the Naive Bayes Classifier Results

```r
plot(results$nbc, type = 'l', ylab = "F1 Score", xlab = "CV Fold")
```



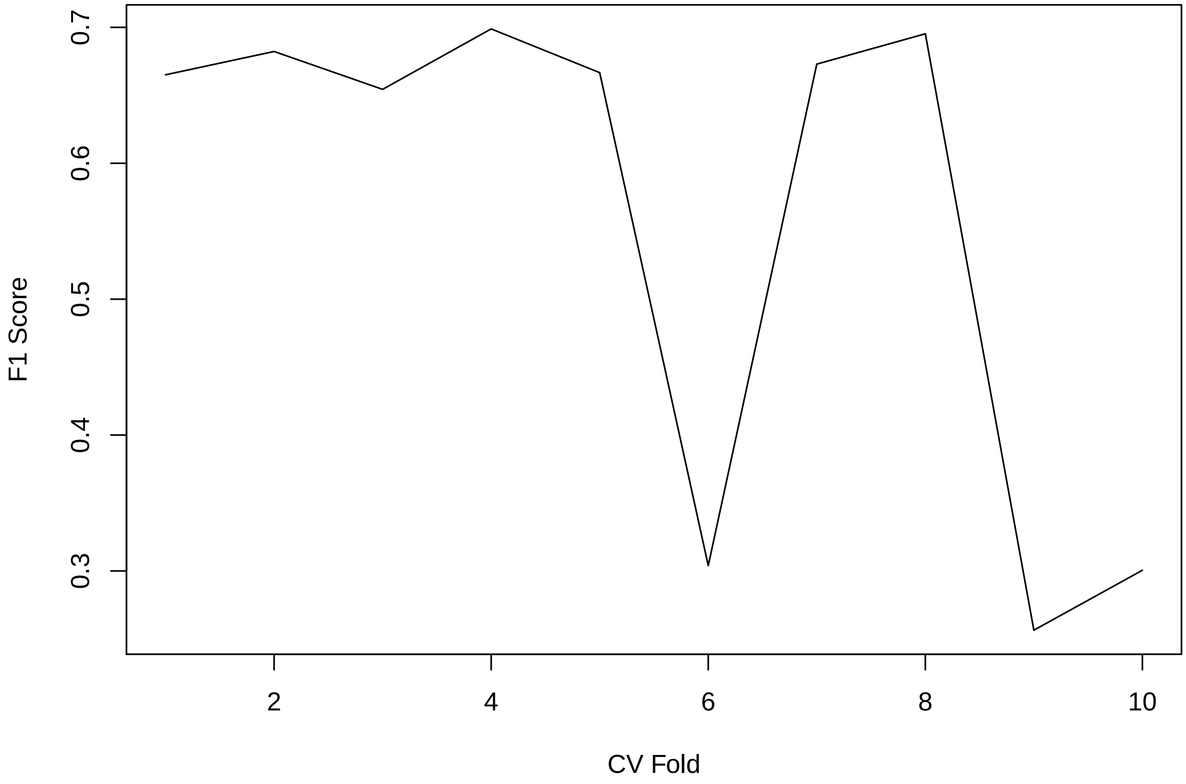
Plot of the Random Forest Results

```r
plot(results$rf, type = 'l', ylab = "F1 Score", xlab = "CV Fold")
```



Plot of the Support Vector Machine Results

```
plot(results$svm, type = 'l', ylab = "F1 Score", xlab = "CV Fold")
```


F1 Score
CV Fold