

My Comprehensive Evaluation

A Comprehensive Evaluation Report

Presented to
The Statistics Faculty
Amherst College

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Arts
in
Statistics

Ian Fayorsey

November 2018

Acknowledgements

I would like extend my graditude to several people. Professor Nick Horton, and Professor Amy Wagaman, for their guidance, and encouragement over the past 4 years at Amherst. Since declaring my major sophomore year, Professor Horton has been my advisor and confidant, helping me through any and all adversities. During challenging times, Professor Wagaman helped me remain confident and hopeful. My academic development from probability to data analysis is a result of your unwavering support. To the entire statistics department, thank you for equipping me with the skills and thought process to tackle future problems in a formulaic and interpretable manner. Finally, to my mother, father, and sister - for never allowing me to accept failure and helping me become the man I am today.

Table of Contents

Introduction	1
0.0.1 Previous Work	2
Chapter 1: Data	3
1.0.1 Data Preprocessing	3
1.1 Exploratory Data Analysis:	4
Chapter 2: Theoretical Background	9
2.1 Support Vector Machines:	9
2.2 Decision Trees	11
2.3 Bagging	11
2.4 Random Forests	13
2.5 Linear Discriminant Analysis	13
Chapter 3: Results	15
3.0.1 Run Time	15
3.0.2 Classification Accuracy	16
Conclusion	19
Appendix A: Appendix	21
References	31

List of Tables

List of Figures

1.1	Pie Chart	5
1.2	Word Count Distribution	6
1.3	TD-IDF per Category	6
1.4	Framework of Text Classification	7
2.1	Support Vector Machine.	10
2.2	Decision Tree	12
2.3	LDA	14
3.1	Bar Plot of Missclasification Rate	16

Abstract

A collection of unstructured data is analyzed using six machine learning algorithms, that classify responses to pre-defined categories. Performance of these models is evaluated in terms of misclassification rate and learning speed. It was discovered that simple machine learning algorithms are more computational efficient than complex models. Support Vector Machines were the most efficient method employed. We conclude by examining the factors that make text classification challenging.

Introduction

By 2022, 93% of all data in the digital universe will be unstructured (Big Data, 1). As the wealth of information stored in comments, tweets, and reviews increases, so does the growing interest in extracting insight from this unstructured data. Organizations such as Amazon, Apple, and Facebook have begun recruiting researchers to investigate how data can be used to better reach their customers. Over the years, this research has led to break throughs in text categorization – the assignment of natural language texts to predefined categories based on content – which has applications in areas such as search engines and customer relationship management. Some of the biggest challenges when working with unstructured data pertain to the volume of data. By nature, a large percentage of the data companies collect is unverified, and remains in its uncleaned, user generated state. When drawing insights from this data, it is vital that the data is transformed into an actionable form. In the case of analyzing text, there are 171,476 words in the English language, making natural language processing a difficult task when dealing with its large volume. In this paper, I look to find the most computationally efficient methods of analyzing natural language.

Six supervised methods were used in this analysis: Support Vector Machines (SVM), Multinomial Logistic Regression, Decision Trees, Bagging, Random Forest, and Linear Discriminant Analysis (LDA). Misclassification rates, precision-recall, and runtimes were used to evaluate each algorithm's computational efficiency. Misclassification rate is defined as 1 minus classification accuracy (the proportion of correctly predicted categories). The report is organized as follows:

- Chapter 1 provides background information on the chosen dataset, how it was mined, how it was transformed, and an exploratory analysis of the features within the data
- Chapter 2 provides the theoretical background of each learning algorithm used in the experiment
- Chapter 3 shows and interprets the results, and concludes

0.0.1 Previous Work

My exposition draws from several noteworthy analyses of textual data. This list includes the role of SVMs in learning text classifiers (Joachim, 1994), the use of semantic orientation in classifying documents (Turney, 2002; Pang et al., 2002), and the benefits of term-frequency transformations (Leopold and Kinderman, 2002).

Chapter 1

Data

The data used in this analysis is a series happy moments. In collaboration with the University of Tokyo, Massachusetts Institute of Technology researchers interview thousands of people and asked them to list ten happy moments that occurred within the last 24 hours. Their responses were recorded and compiled into *HappyDB*, a corpus of 100,535 happy moments. The data set is host publicly on github, as a zip file (<https://rit-public.github.io/HappyDB/>).

It is important to note that this curated data set contains cleaned textual data. In it includes 9 variables, many of which identify the author and qualities of the texts. The variables of interest in this analysis are `cleaned_hm`, the list of happy moments, and its associated predicted category. For my analysis, I sampled 15,000 of these responses and trained on 80% of the data.

Response	Predicted Category
I was happy when my son got 90% marks in his examination	Acheivement
I went to the gym this morning and did yoga	Excercise
My dad and I went fishing	Bonding

1.0.1 Data Preprocessing

In order to analyze how words influence a predicted category, the data set had to be transformed. First, each character vector response is converted into a corpus. A corpus is simply a collection of natural language constructed with a specific purpose. The corpus consists of 15,000 responses samples from the larger dataset.

```
as.character(corpus[[1]])
```

```
[1] "i went to the farm store and found fruit trees on sale for 70% off. "
```

Entries in the corpus are then cleaned of symbols, punctuations, and stop words. Once accomplished the corpus is reduced a collection of key words.

```
as.character(a[[1]])
```

```
[1] "    farm store    fruit trees    sale    "
```

The next step is to create a document term matrix. A document term matrix is a corpus transformation that represents each word as a feature, and each response as a row. The matrix is populated with values 0 or 1, depending on the absence or presence of that word in the document. Traditionally, document term matrices are high dimensional due to thousands of words that can be used as a feature. Often times many of these cells are empty to represent the absence of a word from that document. High sparsity (the proportion of entries which are zero) is another known condition of document terms matrices.

	Terms			
Docs	farm	fruit	sale	store
1	1	1	1	1
2	0	0	0	0
3	0	0	0	0

1.1 Exploratory Data Analysis:

In this section, I present some preliminary results from summaries and exploratory analysis of the cleaned data obtained above. First we look at the response variable **predicted category**. Next, I explore statistics regarding the explanatory variables **terms**.

Predicted Category

Affection and achievement were the most talked about categories (33.9% and 34.1% respectively). This follows conventional logic given that love and sense of accomplishment are vital traits to a good quality of life. Nature and exercises accounted for just 1.2% and 1.8% of labeled responses in the entire data set.

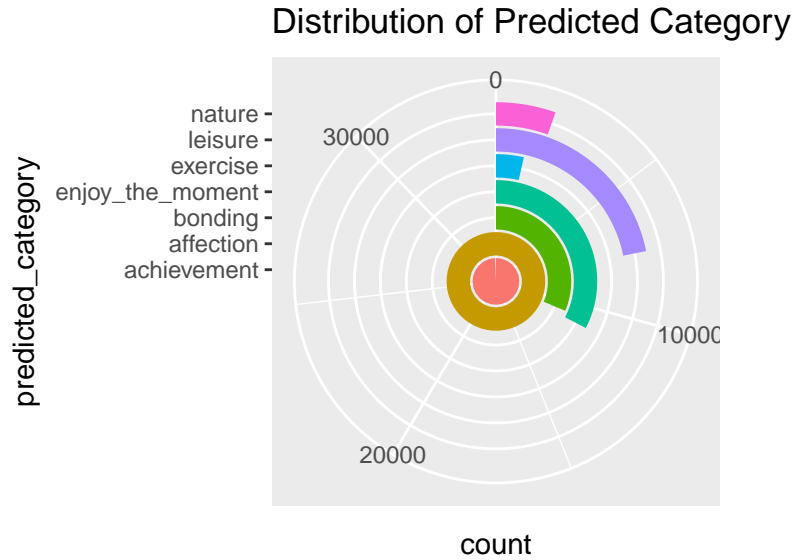


Figure 1.1: Pie Chart

In a predictive setting, having highly disproportionate classes can lead to errors when making predictions based on that training data. When classes are unequally represented, models may simply decide to always predict a certain class in order to achieve a high accuracy. This accuracy paradox does not reflect effective predictions for the model, but rather the state of the underlying distribution. While this is not the case for our data set, consideration must be taken for the classifications of minority labels.

Term Count

In figure 1.2, the majority of entries are between 5 and 14 words. Over 4,000 reviews contain over 50 words. Many of these words add little to no value to an algorithmic interpretation of the sentence. These are the commonly used words such as “a”, “I”, “was” that were removed from the corpus.

Terms used in each category

Two metrics frequently used to quantify the importance of a word are its term frequency (tf), and inverse document frequency (idf). Tf measures how frequently a word appears in a document, while idf weights frequently used words less than words rarely used (Leopold, 2002). When combined, the tf-idf is the frequency of a term adjusted by how rarely it is used.

Figure 1.2 shows a breakdown of the highest tf-idf word per category yields further

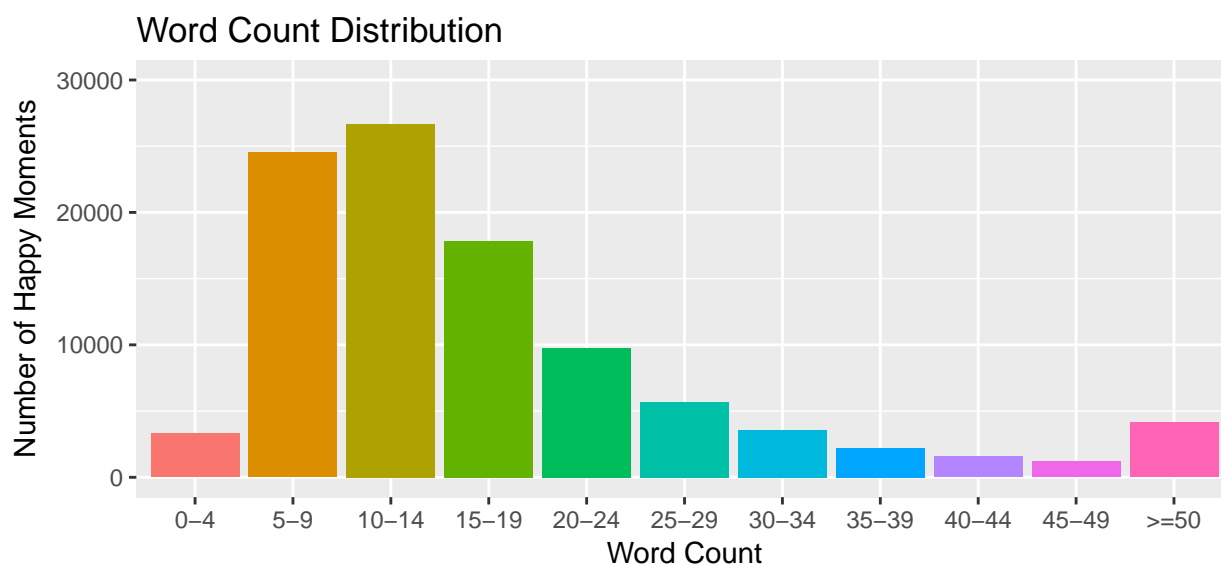


Figure 1.2: Word Count Distribution

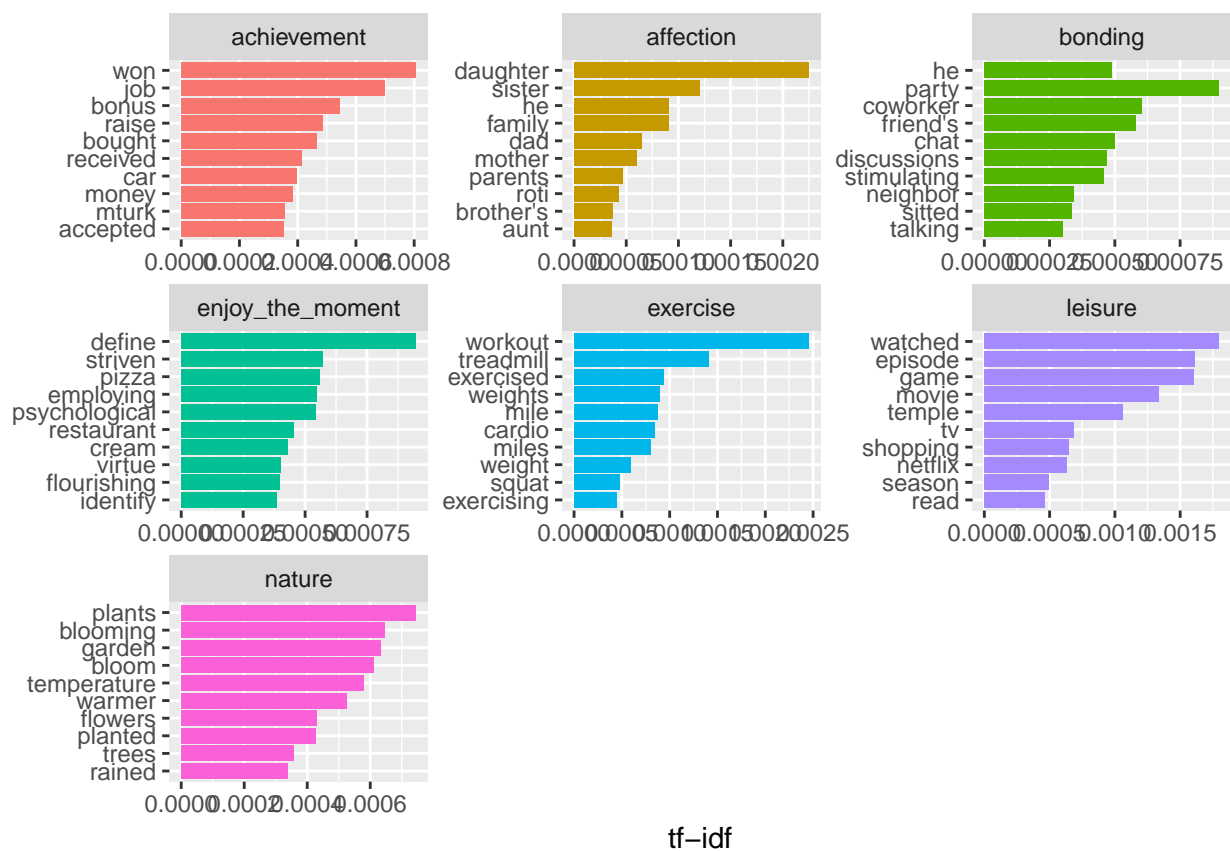


Figure 1.3: TD-IDF per Category

insight into words associated with each label. For phrases related to achievement, words like “won”, “received”, and “accepted” have the highest adjusted usage. In responses categorized as exercise, we see terms related to working out and equipment used. These key terms provide natural separators by which the algorithms can use to classify future responses. In the next section, I discuss some of the methods used to predict the categories of happy moments.

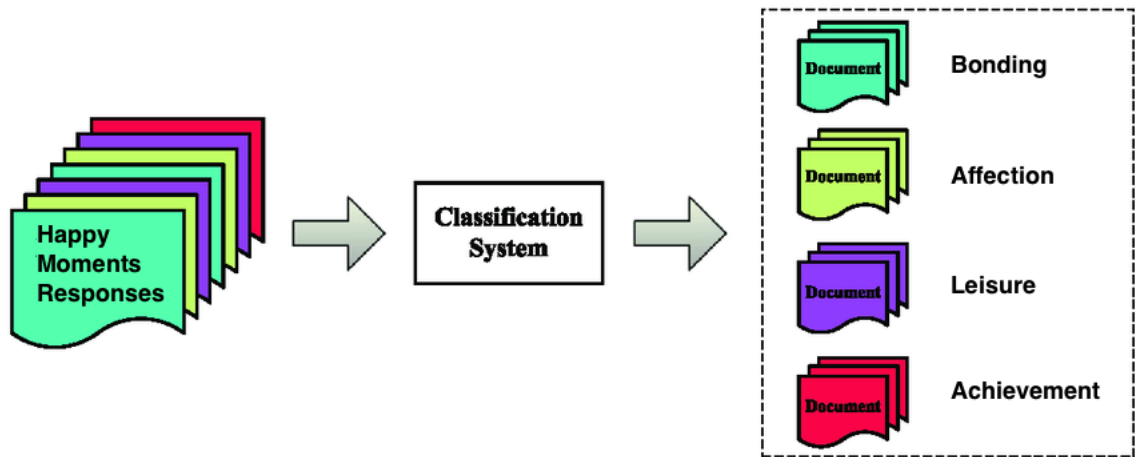


Figure 1.4: Framework of Text Classification

Chapter 2

Theoretical Background

Broadly speaking, there are two types of machine learning techniques: supervised, and unsupervised methods. In a supervised setting, inputs are mapped to a predefined label, and models are trained to predict labels for a new set of inputs. In the case of text classification, each response and associated category are learned by a classifier which then predicts labels for new responses. In an unsupervised setting, there is no corresponding labels to the inputs. These methods cluster response based on associations between responses. Given that this study aims to compare the performance of classifiers when labeling texts, we remain in a supervised setting.

The objective is to minimize the classification error rate. The classification error rate is defined as the proportion of training observations in the region that do not belong to the most common class.

2.1 Support Vector Machines:

The Support Vector Machines (SVM) approach to classification is a generalization of the maximal margin classifier to non-linear decision boundaries. It is first defined by a hyperplane, which is a flat subspace existing in a $p - 1$ dimensional setting. The mathematical definition of a hyperplanes is a line with:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p = 0$$

The hyperplane essentially divides a p-dimensional space into two halves depending on whether or not the linear equation is greater than zero or less than zero for a given $X = (X_1, X_2, \dots, X_p)^T$. In the scenario where X is an $n * p$ matrix where observations fall into two classes y_1, y_2, \dots, y_n between $-1, 1$. If a separating hyperlane exists, it can

be used to construst an intuitive classifier based on the $G(x) = \text{sign}[x_i^T \beta + \beta_0]$. The optimization problem

$$\max M$$

subject to

$$y_i(x_i^T \beta + \beta_0) \geq M, i = 1, \dots, N$$

Where M is the width of the margin.

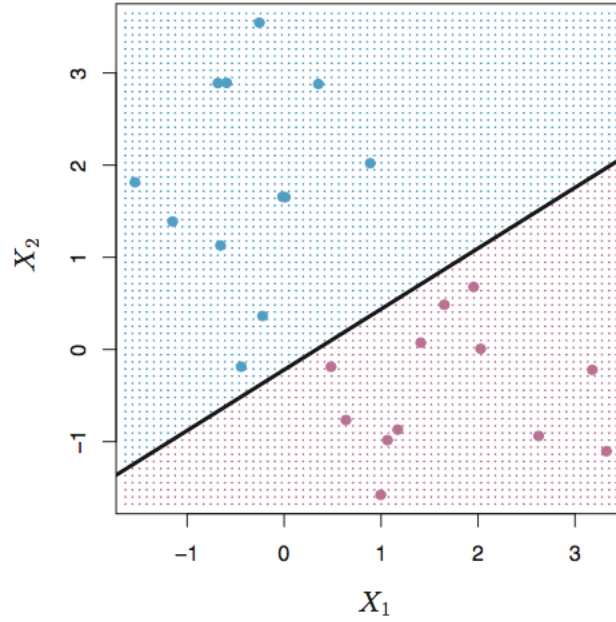


Figure 2.1: Support Vector Machine.

In a high dimensional setting however, this is unlikely to be the case. A soft margin classifier aims to address this by allowing some observations to follow on the wrong side of the margin or even hyperplane. These observations correspond to training points that have been misclassified by the support vector classifier. This constraint can now be modified to be:

$$y_i(x_i^T \beta + \beta_0) \geq M - \xi$$

Where ξ is a slack variable representing observations on the wrong side of the margin/hyperplane. In this setting, points on the wrong side of the margin are penalized proportionally to the distance from the boundary.

Tuning parameter, C can be introduced restraining the number of points that

violate the hyperplane. As C increases we allow for more observations to appear on the wrong side of the hyperplane so the margin widens. Conversely, as C decreases less violations result in a smaller margin. Increasing the cost parameter additionally makes models more expensive and increases the risk of losing model generability.

In situations where there are more than $k > 2$ classes, k SVMs are fit and compared to the remaining $k-1$ classes. Observations are assigned to a class based on which $\beta_0 k + \beta_{1k} X_{1k} + \dots + \beta_{pk} X_{pk}$ is the greatest (Hastie, 337-356).

SVMs hold a unique property in that they can classify observations independent of the dimensionality of the feature space (Joachim). By separating the data on the largest margin rather than the number of features, our model can be easily generalized, irrespective of the size of the feature space.

The issue of non-linearity is typically addressed by enlarging the feature space by using higher order polynomial functions of the predictors.

2.2 Decision Trees

Decision trees partition a feature space into a number of simpler regions. Decision trees are composed of two main parts: a node and leaf. Each node in a decision tree represents a variable by which observations can be group based on conditions associated with that variable. In this setting, every node is a word in the document term matrix. In a classification setting, the tree is split based on which variable minimizes the classification error rate. If we let $\hat{p}_{mk} = 1/N_m * \sum I(y_i = k)$ be the proportion of class k observations in node m , observations are classified to the class where $k(m) = \underset{k}{\operatorname{argmax}} \hat{p}_{mk}$ (Hastie, 312). This can also be described as the majority class for node m . This greedy algorithm chooses which variable to split on based on the largest drop in misclassification rate.

One benefit of decision trees are their interpretability. However due to their high variance, resulting trees can vary greatly if the training data is modified, and can lead to drastically different test labels. Traditionally, classification and regression decision trees are associated with high variance and low bias.

2.3 Bagging

Bootstrap aggregation (bagging) is an ensemble method that combines the predictions of several smaller algorithms to make an improved estimate. Given a set of independent observations $X_1 \dots X_n$ each with variance σ^2 , the variance of the mean \bar{X} is $\frac{\sigma^2}{n}$. This

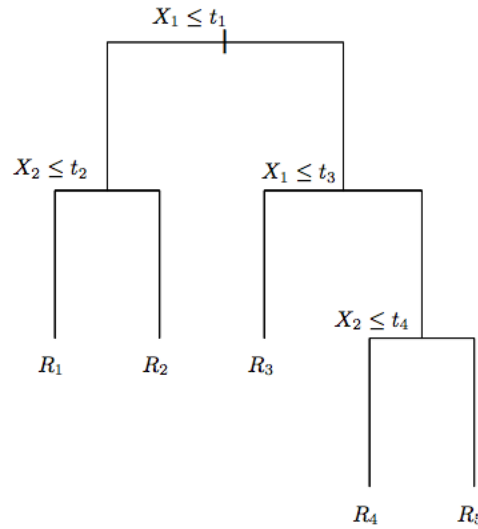


Figure 2.2: Decision Tree

illustrates how averaging over a set of observations can reduce variance. In this approach, several samples are taken from the training data with replacement and a classification tree is trained on each. Given a new data set, the bagging method aggregates the average prediction across the models for each observation and assigns it to the class with the majority vote (Hastie, 317).

Because the prediction is an average of several trees, we are less concerned with one tree overfitting. This allows for trees to remain unpruned, with high variance. In this setting the only adjustable parameter is the number of samples drawn, in essence, how many decision trees are averaged. Traditionally, this number is chosen through cross-validation (increased until accuracy stops showing improvement). It must be kept in consideration however that more samples will require more time to train the models, and may be computational infeasible.

Due to the algorithms greedy nature, a series of decision trees may have highly correlated predictions as a result of the method dividing the data based on the largest drop in missclassification rate. Imagine a scenario where there is only one strongly correlated predictor. Even with a high number of samples, we can expect most trees generated to split the first node on this variable. Situations where there is structural similarity between bootstrap samples can lead to predictions that are highly correlated. **Note:** The Bagging model did not provide reproducible results, therefore it is omitted from the remainder of this experiment. Nonetheless, it is an important algorithm that establishes the framework for the next method discussed in this chapter.

2.4 Random Forests

Random Forests algorithms improve on the bootstrap aggregation by decorrelating trees. The main difference is that in a random forest setting, only a random sample of m predictors from p variables can be chosen as a splitting feature. This alleviates the issue described earlier by not considering the one strong predictor in approximately $(p - m)/p$ of the splits. Situations where $m = p$ is simply a bootstrap aggregation (Hastie, 319). Smaller values of m are traditionally helpful when there is a large number of correlated features.

2.5 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is most commonly used as a dimension reduction technique. In the case of $p > 1$ classes, each class density is assumed to follow a multivariate gaussian distribution $N(\mu_k, \Sigma)$, where μ_k is a class mean vector, and Σ is common covariance matrix.

$$\begin{aligned} \log(Pr(G = k|X = x)/Pr(G = l|X = x)) = \\ \log(f_k(x)/f_l(x)) + \log(\pi_k/\pi_l) \\ \log(\pi_k/\pi_l) - 1/2(\mu_k + \mu_l)^T \Sigma^{-1}(\mu_k - \mu_l) + x^T \Sigma^{-1}(\mu_k - \mu_l) \end{aligned}$$

When simplified further, the above becomes:

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - 1/2 \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k$$

This equation assigns an observation to the class for which its $\delta_k(x)$ is the largest (Hastie, 143). Since the parameters of the gaussian are unknown, following estimates are used:

$$\begin{aligned} \hat{\pi}_k &= N_k/N \text{ where } N_k \text{ is the number of class-}k \text{ observations} \\ \hat{\mu}_k &= \sum_{g=k} x_i / N_k \\ \hat{\Sigma}_k &= \sum_{k=1}^K (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T / (N - K) \end{aligned}$$

As you can see the LDA decision rule depends on x only through a linear combination of its elements. One drawback however of the LDA is the restrictive conditions associated with the algorithm. LDA assumes that features within the data set are a series of independent multivariate gaussians with identical covariance matrices. Additionally, LDA requires the number of features to be less than the sample size. As

p approaches n , the performance of the model declines. If the assumption of uniform variance is highly off, then LDA can suffer high bias (Hastie, 153). It will be interesting to see how these violations play out in a high dimensional, sparse setting.

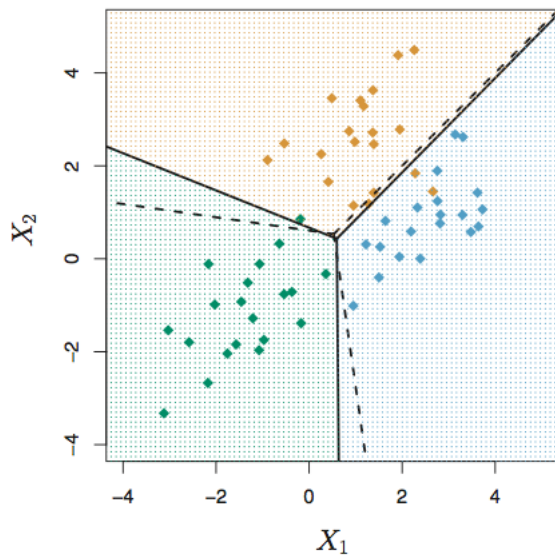


Figure 2.3: LDA

Chapter 3

Results

Five models are trained on a random sample of 8,000 observations, and tested on a sample of 2,000 unseen responses. Both sets are comprised of indicator variables that denote the presence or absence of words in each response.

3.0.1 Run Time

The amount of time taken to learn the 8,000 responses in the training set varied substantially across all methods. The linear SVM and decision tree were amongst the fastest trained models taking just 69.7414799 and 94.1912999 seconds respectively. The multinomial logistic was the fastest model, taking just 134.086921 seconds.

The higher runtime for the random forest model follows convention given that it is an aggregation of several learning methods. The runtime of the LDA model initially came as a surprise, however LDA computes discriminant scores $\delta_k(x)$ by finding a linear combination of the independent variables. In a highly sparse setting, this becomes a linear combination of over 10,000 variables for each of the 6 categories. The training time of the SVM, decision tree, and multinomial models were quite impressive, given the nature of the problem when applied to larger data sets.

model	accuracy	runtime
SVM	0.2000	69.74148
Multinomial	0.2250	94.19130
Decision Trees	0.6430	134.08692
Random Forests	0.2565	5234.51824
LDA	0.2610	15096.38790

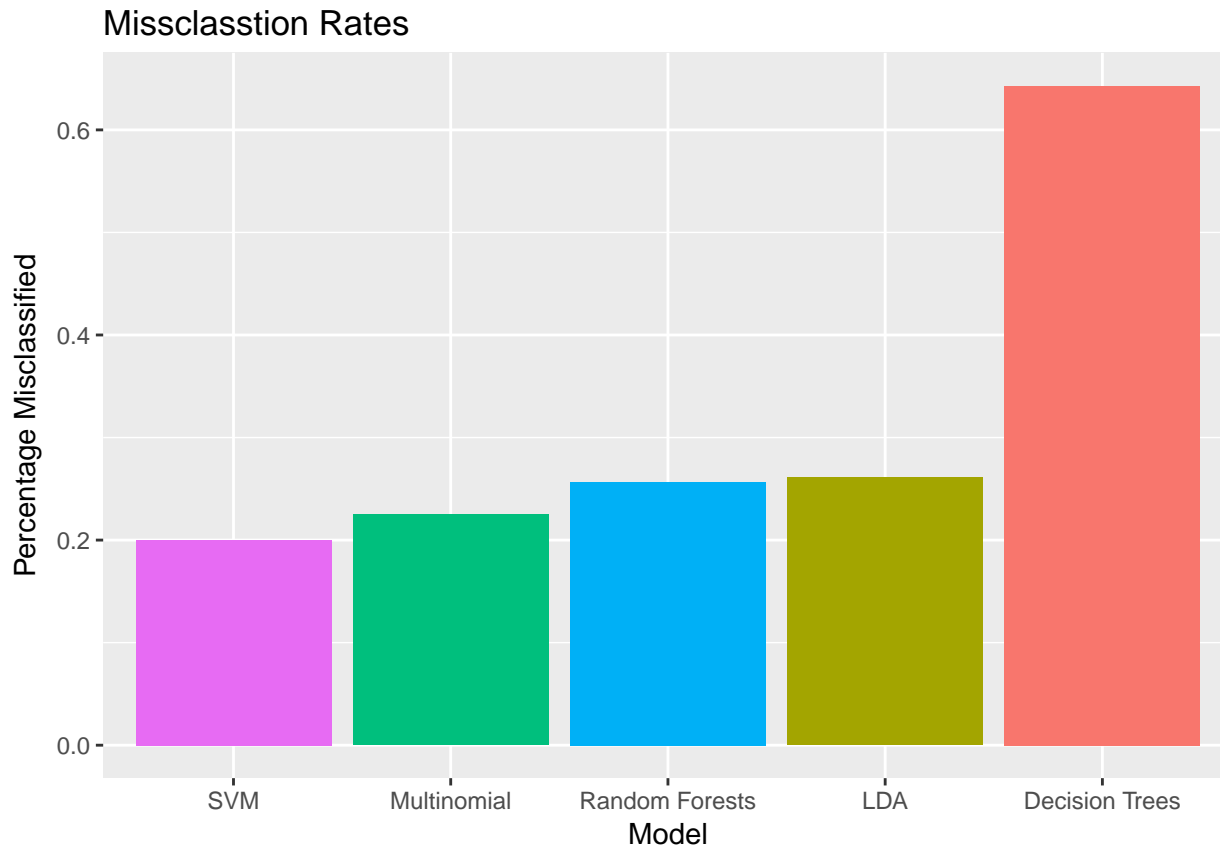


Figure 3.1: Bar Plot of Missclassification Rate

3.0.2 Classification Accuracy

As stated in the introduction the goal of this analysis was to compare classification error rates between models. SVMs had the lowest misclassification rate with 0.2. This was a slight improvement on the multinomial model (0.225 misclassification rate). LDA and Random Forest performed surprisingly well (0.261 for LDA and 0.2565 for rf). The decision tree produced the highest misclassification rate with 0.643 incorrectly categorized. Figure 3.1 shows the misclassification rates for all five algorithms tested.

While accuracy is a valid measure for information retrieval, in situations where classes are imbalanced, always predicting one class may yield a high accuracy. Precision and recall can be used to focus the evaluation on the correctly label categories (true positives). Precision is defined as the proportion of items i placed in the category correctly out of every algorithmic declaration of i , while recall is the proportion of items correctly guessed when it truly belonged in that category (Platt, 5). Low precision-high recall systems guess a label frequently, however a significant portion of labels are incorrect. High precision-low recall systems are the opposite, predicting a class less frequently but more accurately. Ideal systems have both high precision and

high recall, will return many results, with all results labeled correctly. The table below consists of precision-recall values for the category achievement for all 5 algorithms.

	SVM_PRECISION	SVM_RECALL
SVM	0.81	0.88
Multinomial	0.64	0.96
Decision Tree	0.49	0.95
Random Forests	0.72	0.89
LDA	0.75	0.78

The most efficient classifier (SVM) had high precision and high recall for the category achievement. Although the decision tree predicted achievement frequently, only 0.49 were correctly labeled. Interestingly enough, this was also this case for the multinomial model, which had a low precision of 0.64 and high recall of 0.96. However unlike the decision tree, the overall misclassification rate for the multinomial was quite low (0.225% labeled incorrectly). When their confusion matrices are compared, it is immediately apparent that two classes were not predicted in the decision tree model. Due to its greedy nature, the decision tree appeared to overfit on the noise in the training data. This leads to poor generalizability, and the algorithm only predicting the majority categories. On the other hand, the multinomial accurately predicted a high proportion of the other classes.

	achievement	affection	bonding	enjoy_the_moment	exercise	leisure	nature
achievement	625	242	34	208	13	110	31
affection	19	400	1	1	0	2	2
bonding	0	20	180	2	0	0	0
leisure	13	16	2	5	7	62	5

	achievement	affection	bonding	enjoy_the_moment	exercise	leisure
achievement	633	93	27	140	9	71
affection	16	567	5	13	0	14
bonding	1	7	183	1	0	0
enjoy_the_moment	5	5	1	58	1	2
exercise	0	0	0	0	9	0
leisure	2	6	1	4	1	87

	achievement	affection	bonding	enjoy_the_moment	exercise	leisure
nature	0	0	0	0	0	0

Conclusion

This experiment compares the performance of six machine learning algorithms tasked with classifying unstructured textual data. First, I discuss how the transformation of texts into document term matrices allows for words to be used as features in supervised methods. I also show how each of the algorithms creates a classifier based on these features, and predicts labels for new data. Due to issues with reproducibility, the bagging model discussed theoretically and was not evaluated further. My results demonstrate that accurate text classifiers can be trained from relatively simple models. Following Joachim's findings, it was empirically shown that SVM models consistently outperform competing methods in precision, recall. The high number of features and sparsity of the data aided in the creation of decision boundaries cast by the SVM by relying on the fact that the features are linearly separable. Although other methods such as Random Forests and LDA had high classification rates, their runtimes made them poor alternatives. A more detailed look at precision and recall for the LDA model highlights how its failed assumptions impacted the accuracy of the model. Likewise, the high variance of the decision tree is evident high error rate.

The two most important aspects of this experiment are the structure of the data, and the assumptions of the models. Further exploratory analysis may have shown why methods like LDA were not suitable for this data. Most of the limitations are a result of the high dimensionality of the data, or its incompatibility with the algorithms. The presence of the zero value in a sparse matrix provides no information while allocating memory for each 32-bit value. A document term matrix of the entire dataset occupies 25 megabytes of space. Using 15,000 observations, a more manageable 3.1 mb document term matrix is created. While this made computations feasible, reducing the size of the corpus and document term matrix reduces the number of unique words encountered in the training data.

For future studies, it would be advisable to test other natural language processing techniques. For example, how might multi-word (n-grams), noun phrases, and weighting affect the classification accuracy? Another interesting factor to consider is

the role of dimension reduction techniques such as principal component analysis and support vector decomposition. By projecting data onto lower dimensional subspaces, these methods may alleviate some of the issues related to sparse matrices, and provide improved algorithmic runtimes.

Appendix A

Appendix

In the main Rmd file:

```
library(acstats)
library(RJSONIO)
library(RCurl)
library(ngram)
library(tm)
library(wordcloud)
library(ggplot2)
library(glmnet)
library(text2vec)
library(data.table)
library(magrittr)
library(dplyr)
library(mosaic)
library(RTextTools)
library(e1071)
# library(caret)
library(tidyr)
library(stringr)
library(jsonlite)
library(tidytext)
library(rpart)
library(broom)
library(kableExtra)
```

```
#library(MASS) #lda
```

```
#Loading Data
```

```
setwd("~/Desktop/Statistics/Comps/Comps - Fayorsey/Comps-Fayorsey19E")
data <- read.csv("cleaned_hm.csv", stringsAsFactors = FALSE)

data$predicted_category <- as.factor(data$predicted_category)
```

```
#Creating Term Document Matrix
```

```
set.seed(7)
random_hm <- sample(1:nrow(data), 15000)
corpus <- Corpus(VectorSource(data$cleaned_hm[random_hm]))
skipWords <- function(x) removeWords(x,
  words = c(stopwords(kind = "en"), 'happy',
    'day', 'got', 'went', 'today', 'made', 'one',
    'two', 'time', 'last', 'first', 'going', 'getting',
    'took', 'found', 'lot', 'really', 'saw', 'see', 'month',
    'week', 'day', 'yesterday', 'year', 'ago', 'now', 'still',
    'since', 'something', 'great', 'good', 'long', 'thing',
    'toi', 'without', 'yesteri', '2s', 'toand', 'ing'))
funcs <- list(skipWords, stripWhitespace, removeNumbers,
  removePunctuation, tolower)
a <- tm_map(corpus, FUN = tm_reduce, tmFuns = funcs)
a_tdm <- TermDocumentMatrix(a)
m <- as.matrix(a_tdm)
v <- sort(rowSums(m), decreasing = TRUE)
d <- data.frame(word = names(v), freq = v)
d <- head(d, 10);d
```

```
#Displaying TDM for first three observations
```

```
a_dtm <- DocumentTermMatrix(a)
wa <- as.matrix(a_dtm)
```

```
wa[1:3,1:4]
```

```
#PieChart of Categories
```

```
ggplot(data, aes(x=predicted_category, fill=predicted_category))+
  geom_bar()+
  labs(title = "Distribution of Predicted Category")+
  guides(fill="none")+
  coord_polar(theta = "y", start=0)
```

```
#Summary of wordcount
```

```
count <- sapply(data$cleaned_hm, wordcount) # Counts number of words
summary(count)
```

```
#Distribution of Word Counts
```

```
category <- c("0-4", "5-9", "10-14", "15-19", "20-24", "25-29", "30-34",
              "35-39",
              "40-44", "45-49", ">=50")
count_class <- cut(count, breaks = c(0,4,9,14,19,24,29,34,39,44,49,
                                     Inf),
                  labels = category, include.lowest = TRUE)
ggplot()+
  geom_bar(aes(x = count_class, fill = count_class))+
  ylim(0,30000)+
  labs(x = "Word Count", y = "Number of Happy Moments",
       title = "Word Count Distribution")+
  guides(fill = "none")
```

```
#TD-IDF
```

```
words <- data %>%
  unnest_tokens(word, cleaned_hm) %>%
  count(predicted_category, word, sort = TRUE) %>%
```

```

ungroup()

totalwords <- words %>%
  group_by(predicted_category) %>%
  summarize(total = sum(n))

words <- left_join(words, totalwords);words

tf <- words %>%
  bind_tf_idf(word, predicted_category, n);tf

tf %>%
  select(-total) %>%
  filter(n >=30) %>%
  arrange(desc(tf_idf));tf

```

#TD-IDF Graph

```

tf %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word)))) %>%
  group_by(predicted_category) %>%
  top_n(10) %>%
  ungroup %>%
  ggplot() +
  geom_col(aes(word, tf_idf, fill = predicted_category),
           show.legend = FALSE) +
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~predicted_category, ncol = 3, scales = "free") +
  coord_flip()

```

#Creating containers for models

```

set.seed(57)

bin_data <- data[sample(1:nrow(data), 15000), ]

```

```
my.matrix1 <- create_matrix(bin_data$cleaned_hm, language="english",
                             removeNumbers=TRUE, stemWords=FALSE,
                             weighting=tm::weightTfIdf)

container1 <- create_container(my.matrix1, bin_data$predicted_category,
                              trainSize=1:8000, testSize =8001:10000,
                              virgin = FALSE)
```

#SVM model

```
set.seed(6)
start_svm <- Sys.time()
svm_model <- train_model(container1, "SVM")
end_svm <- Sys.time()

svm_results <- classify_model(container1, svm_model)
ac_svm <- mean(svm_results$SVM_LABEL != bin_data[8001:10000,9])

table(svm_results$SVM_LABEL, bin_data[8001:10000,9])

end_svm - start_svm

pr_svm <- create_precisionRecallSummary(container1, svm_results)
```

#Tree Model

```
set.seed(1)
start_tree <- Sys.time()
tree_model <- train_model(container1, "TREE")
end_tree <- Sys.time()

tree_results <- classify_model(container1, tree_model)
table(tree_results$TREE_LABEL, bin_data[8001:10000,9])

ac_tree <- (682+363+202+39)/2000
```

```
end_tree - start_tree
```

```
pr_tree <- create_precisionRecallSummary(container1, tree_results)
```

```
#Multinomial Model
```

```
set.seed(2)
```

```
start_multi <- Sys.time()
```

```
multinomial_model <- train_model(container1, "GLMNET",  
                                family="multinomial")
```

```
end_multi <- Sys.time()
```

```
multi_results <- classify_model(container1, multinomial_model)
```

```
table(multi_results$GLMNET_LABEL, bin_data[8001:10000,9])
```

```
ac_multi <- mean(multi_results$GLMNET_LABEL != bin_data[8001:10000,9])
```

```
end_multi - start_multi
```

```
pr_multi <- create_precisionRecallSummary(container1, multi_results)
```

```
#Bagging Model
```

```
# set.seed(3)
```

```
# start_bagging <- Sys.time()
```

```
# lda_model <- train_model(container1, "BAGGING")
```

```
# end_bagging <- Sys.time()
```

```
#
```

```
# bagging_results <- classify_model(container1, lda_model)
```

```
# table(bagging_results$BAGGING_LABEL, bin_data[8001:10000,9])
```

```
# ac_bagging <- mean(bagging_results$BAGGING_LABEL != bin_data[8001:10000,9])
```

```
#
```

```
# end_bagging - start_bagging
```

```
# pr_bag <- create_precisionRecallSummary(container1, bagging_results)
```

```
#Random Forests Model
```

```
set.seed(4)
```

```

start_rf <- Sys.time()
rf_model <- train_model(container1, "RF", ntree=10)
end_rf <- Sys.time()

rf_results <- classify_model(container1, rf_model)
table( rf_results$FORESTS_LABEL, bin_data[8001:10000,9])
ac_rf <- mean(rf_results$FORESTS_LABEL != bin_data[8001:10000,9])
end_rf-start_rf

pr_rf <- create_precisionRecallSummary(container1, rf_results)

```

#LDA Model

```

set.seed(5)
start_lda <- Sys.time()
lda_model <- train_model(container1, "SLDA")
end_lda <- Sys.time()

lda_results <- classify_model(container1, lda_model)
table(lda_results$SLDA_LABEL, bin_data[8001:10000,9])
ac_lda <- mean(lda_results$SLDA_LABEL != bin_data[8001:10000,9])
end_lda - start_lda

pr_lda <- create_precisionRecallSummary(container1, lda_results)

```

#Runtime table

```

time_svm <- end_svm - start_svm

# time_bagging <- end_bagging - start_bagging

time_rf <- end_rf - start_rf

time_multi <- end_multi - start_multi

time_lda <- end_lda - start_lda

```

#Decision Tree Confusion Matrix

```
table(tree_results$TREE_LABEL, bin_data[8001:10000,9])
```

#Multinomial Confusion Matrix

```
table(multi_results$GLMNET_LABEL, bin_data[8001:10000,9])
```


References

- Foster, D., et al. (2013). *Featurizing text: Converting text into predictors for regression analysis*. The Wharton School of the University of Pennsylvania.
- Hastie, T., et al. (2013). *An introduction to statistical learning with applications in R*. Springer.
- Joachims, T. (1994). *Text categorization with support vector machines: Learning with many relevant features* (pp. 1–6). University of Dortmund.
- Leopold, Edda, & Kinderman, J. (2002). Text categorization with support vector machines. how to represent texts in input space?, 1–20.
- Pang, Bo, & Lee, L. (2002). *Thumbs up? Sentiment classification using machine learning techniques* (pp. 1–20). Cornell University.
- Platt, et al, John. (1994). *Inductive learning algorithms and representations for text categorization*. Stanford University.
- Yogatama, D. (2015). *Sparse models of natural language text*. Carnegie Mellon University.