Examining the Effect of Word Embeddings and Preprocessing Methods on Fake News Detection

Jessica Hauschild
University of Nebraska-Lincoln, jessica.hauschild@huskers.unl.edu

EXAMINING THE EFFECT OF WORD EMBEDDINGS AND PREPROCESSING METHODS ON FAKE NEWS DETECTION

by

Jessica L. Hauschild

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Doctor of Philosophy

Major: Statistics

Under the Supervision of Professor Kent Eskridge

Lincoln, Nebraska

May, 2023

EXAMINING THE EFFECT OF WORD EMBEDDINGS AND PREPROCESSING METHODS ON FAKE NEWS DETECTION

Jessica L. Hauschild, Ph.D.

University of Nebraska, 2023

Advisor: Kent Eskridge

The words people choose to use hold a lot of power, whether that be in spreading truth or deception. As listeners and readers, we do our best to understand how words are being used. There are many current methods in computer science literature attempting to embed words into numerical information for statistical analyses. Some of these embedding methods, such as Bag of Words, treat words as independent, while others, such as Word2Vec, attempt to gain information about the context of words. It is of interest to compare how well these various methods of translating text into numerical data work specifically with detecting fake news. The term "fake news" can be quite divisive, but we define it as news that is hyper-partisan, filled with untruths, and written to cause anger and outrage, as defined in Potthast & Kiesel (2018). We hypothesize a person's word choice relates to the factualness of an article. In Chapter 5, we utilize this embedded information in several binary classification methods. We find that words are only marginally valuable in detecting fake news regardless of the embedding or classification method used. However, within natural language processing tasks, there are many preprocessing steps taken to get the text ready for analysis, which is explored in Chapter 6. The embedding methods are confounded with the preprocessing methods used. Preprocessing of text includes, but is not limited to, filtering out words that do not appear a minimum number of times, filtering out stop words, removing numbers, and translating all letters to lower case. We find filtering out stop words and removing words not appearing a minimum number of times have the most significant effect in combination with embedding and classification methods. Finally, in Chapter 7, we extend the classification to six categories ranging from true to pants-on-fire false and found these preprocessing methods are not as influential as they were with the binary outcome. Other predictors outside of the words and word embeddings themselves are necessary for improvement in the detection of fake news.

DEDICATION

Dedicated to Shirley Hauschild and Wilda McKee.

ACKNOWLEDGMENTS

I would like to start by thanking my parents, Craig and Dayna Hauschild. They had to deal with every low point of this whole process. They patiently listened to every phone call where I wanted to quit. They also celebrated with me every high, and I am eternally grateful for their love and support. Next, I would like to thank my advisor, Dr. Kent Eskridge. He has been incredibly patient and supportive throughout the entirety of my graduate career. I have learned much from him, and I hope to be a mentor like him in my future career. I also need to thank Dr. Reka Howard, not only as a member of my committee, but as my emotional support throughout this PhD program. She was always a listening ear to struggles I was having, both personally and academically. I have enjoyed having the opportunity to work with her as her teaching assistant for multiple semesters and have learned a lot from her. I would like to thank the rest of my committee members: Dr. Stephen Scott, Dr. Yuzhen Zhou, and Dr. Susan VanderPlas. They provided beneficial input into my dissertation, and I value the time they provided me in their busy schedules to serve on my committee. Dr. VanderPlas provided invaluable time and feedback on all of the plots you will see. I have so many friends to thank because I literally could not have done this without them: Elizabeth McFaddin, Kelsey Karnik, Miguel Fudolig, Emily Robinson, Alison Kleffner, Johnna Baller, Vamsi Manthena, Ashley Erceg, and Ella Burnham. I would list off all the ways these people have supported me and encouraged me over the years but that would take pages and pages, and I just don't have time (or space) for that. Lastly, I send my thanks to my nieces, Avery and Aubriella, and my nephew, Zackary (Bubba). They have no idea how much light they brought into my life, and I

hope they never lose it.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The analysis of text has become very popular over the years. From analyzing customer reviews left on products for sale, to clustering documents by topic, to detecting fake news, the analytical information held by text has become a valuable tool. Different methods have been developed to transform the information presented in words, either written or spoken, into numerical information. These methods tend to fall in one of two categories: treat all words as independent or try to capture the semantic relationships between the words.

1.1.1 Embedding Methods

The most basic method is known as the Bag of Words method. In the Bag of Words method, each word within a document is treated as independent from the others (Kwartler, 2017). For instance, within a sentence, let's say we focus on word 4. We ignore the word immediately prior and after word 4 in the bag of words method in regards to how they appear relative to each other. Order does not matter. Using this method, we create what is called the document-term matrix. Each document makes up a row of the matrix, and each unique word is a column of the matrix. A cell of the matrix contains the total number of appearances of each unique word per document. In this method, the relationship between words is lost - the relationship being the number of times word pairs appear together; see Section 2.1.

Utilizing the document-term matrix created with the Bag of Words method, we can apply the Term Frequency - Inverse Document Frequency (TF-IDF) measure to determine how relevant a word is to a document by weighting the term frequency in relation to how many documents the word appears in (Silge & Robinson, 2021). If we are interested in determining topics related to the documents, we can apply Principal Component Analysis to the document-term matrix created with the Bag of Words method. We can determine topics of documents by exploring what words are related to each principal component (Bikienga, 2018). Details about TF-IDF are found in Section 2.1.1, and details of the PCA method are in Section 2.1.2.

In writing, however, words appearing before and after a specific target word usually have some importance. These words can offer context of the target words, helping to give it meaning. Hence, the words surrounding the target word are called context words. Methods working to capture the relationship between context and target words have been heavily developed and researched in the computer science and artificial intelligence literature. The main idea of all these methods is one can derive the sense of a word based on the words that surround it - i.e. words are similar if they appear with similar context words (Harris, 1954; Torregrossa, Allesiardo, Claveau, Kooli, & Gravier, 2021). One method, Word2Vec, works to capture this semantic relationship between the words utilizing a neural network. This neural network contains an input layer, a single hidden layer, and an output layer. However, we are not interested in the output layer with this method; we care about the weights learned in the hidden layer (McCormick, 2016). There are two versions of Word2Vec developed by Mikolov, Chen, Corrado, & Dean (2013) called Continuous Bag of Words and Skip-Gram. These models are essentially reflections of one another. The Skip-Gram model utilizes the target word to predict the context words, and the Continuous Bag of Words model does the reverse. Details for the Skip-Gram model will be shown in Section 2.2, as well as a depiction of the network in Figure 2.1. Through the training of this neural network, the Skip-Gram model calculates the probability a context word appears with the target word in the output layer. Because we expect the probabilities of context words to be close together for similar target words, the weights found in the hidden layer should be similar (McCormick, 2016). Hence, this weight matrix in the hidden layer contains information about the semantic relationships between words. If a word embedding vector (i.e. a row in the weight matrix) is close to another word embedding vector, then those words are more likely to be semantically similar.

Another popular method, Global Vectors for Word Representation (GloVe), involves dimension reduction of the co-occurrence matrix. The co-occurrence matrix is a term-term matrix, where the number of rows and columns is the number of unique words in the entire collection of documents. A cell contains the frequency in which the two words appear together. Here, we define “together” as within the same window-size around a target word. A nearest-neighbors and weighted least squares approach is used to estimate a ratio comparing probabilities of a word given the presence of a target word (Pennington, Socher, & Manning, 2014). Details are found in Section 2.3.

A popular, more current method of creating word embeddings capturing context is Bidirectional Encoder Representations from Transformers (BERT). The foundations of BERT are found in development of transformers. Transformers can be considered as a type of language translator device comprised of an encoder and decoder stack of multiple neural networks. BERT does not contain all pieces of a transformer; it only contains the encoder stack. The encoder stack contains multiple attention head and neural networks, all of which are explained in detail in Section 2.4. This method, as compared to Word2Vec and GloVe, allows for the context of a word be addressed from the whole sequence instead of just a single direction (Horev, 2018).

All of these embedding methods, ranging from Bag of Words to BERT, are used in multiple types of natural language processing tasks. For this dissertation specifically, they will be used in the detection of fake news as described in Chapter 2. The detection of fake news at its essence is a document clustering and classification problem with the goal being able to identify real news from fake news.

1.1.2 Fake News Detection

With the exploding popularity of social media, news is able to travel faster than ever before, but this is not without consequences. The biggest consequence is the transmittal of fake news. The term “fake news” can be quite divisive. It has been used to describe any news source or story that does not agree with a person’s viewpoints on matters such as politics, vaccines, climate change, gun control, etc. However, according to Potthast & Kiesel (2018), fake news arose from the fact certain "news" spreads faster, and more successfully, than others. The "news" spreading quickly with the most success is hyper-partisan, filled with untruths, and written in such a way as to cause anger and outrage. In other words, the news that seems to spread the fastest is not always truthful.

If people are relying on social media as their source of news, they must take the time to check the credibility of the story or the site posting it. However, there have been many psychological studies showing once a misconception has been formed and then solidified by an article supporting the misconception, the presentation of true, factual information is unhelpful in reducing said misconceptions. In fact, presenting this true and factual information may even increase misconceptions (Shu, Sliva, Wang, Tang, & Liu, 2017).

Additionally, on social media, there is a psychological phenomenon known as the "echo chamber effect." A person tends to follow like-minded people on social media, and due to this, the psychological challenges mentioned previously of surmounting the impact of fake news are even more difficult to overcome. If a person sees another share an article, that person is likely to believe the article is credible since someone else already does. In addition, if a person continually hears certain information, whether it is true or false, the person is more likely to believe it. Both factors are heightened on social media because of the echo chamber effect since "segmented, homogeneous communities" have been created (Shu et al., 2017).

Generally, an entire news article is neither completely true nor completely false; it is usually a mixture of the two. This leads to looking specifically at statements made in these articles or made verbally to determine if statements are true or false. The non-profit organization PolitiFact does just this. The credibility of this organization will be described in Chapter 4, but briefly, the people who work at PolitiFact scour articles every day looking for statements to fact check. A portion of the statements they choose to fact check are submitted by every day people. After choosing statements to fact check and following an extensive process, the statement is given one of six labels that can be condensed into real or fake news.

1.2 Objectives

Using statements collected and labeled by PolitiFact, we will assess how treating words as independent compare to methods capturing context when classifying these statements. We will be focusing purely on how well the words by themselves classify fake news in order to investigate these methods at more of a pure level with regards to classification. In much of the literature, other features, such as number of paragraphs, are used to assist in the classification, and these other features will be ignored for this dissertation. It is of note these other features have been found to be useful with classification (Horne & Adali, 2017; Potthast & Kiesel, 2018; Shu et al., 2017).

The Bag of Words and TF-IDF methods allow for the document-term matrix to be used directly for classification of documents, where each variable - i.e. a unique word - is a column. With a word embedding method utilizing a neural network or dimension reduction, using this for classification is not as straight forward. Recall these methods output a unique vector for each word contained in the group of documents. These vectors can then be concatenated into a matrix where the number of rows is the number of unique words and the number of columns is the number of dimensions. From there, a few methods have been developed in order to translate this to a vector per document, including averaging over each dimension for every word contained in the document or using the minimum and maximum for each dimension (Boom, Canneyt, Demeester, & Dhoedt, 2016; Palachy, 2019). In this dissertation, we will be utilizing the average across the dimensions for the vectors of words contained in a statement. We begin by exploring the relationship between these embedding methods and different classification methods on fake news detection with only a binary response (real or fake news) in Chapter 5.

As with all natural language processes, there are multiple ways of preprocessing the text to get it ready to be used in an embedding method. The preprocessing methods utilized affect how well the embedding methods capture context as well as the complexity of the document-term matrix. Thus, classification of fake news is confounded with what preprocessing methods are utilized by the authors. As a contribution to current fake news detection literature, an exploratory analysis is completed in order to explore how certain preprocessing methods affect binary classification. We explore the effect of removing numbers, converting to lowercase, filtering stop words, and removing any word not appearing a minimum number of times, as explained in Chapter 6.

Lastly, we extend from the binary classification of real/fake news to the six labels used by PolitiFact in Chapter 7. We will explore how well the embedding methods and classification methods extend to a multi-class response. In addition, any preprocessing methods found to have an effect in the binary case will be explored further with the multi-class case.

There are three major objectives of this dissertation.

1. Compare Bag of Words, TF-IDF, PCA on the document-term matrix, Word2Vec Continuous Bag of Word, Word2Vec Skip-Gram, GloVe, and BERT with how well they classify fake news.

2. Investigate the effect of text preprocessing methods on classification and embedding methods with regard to classification of fake news.

3. Extend classification from the binary case to the multi-class case, and evaluate the classification of fake news.

Chapter 2

Review of Embedding Methods

The purpose of this chapter is to introduce methods used to translate words into quantitative values. Before we begin, let us define a few terms to assist us. There are target and context words within a sentence. A target word is the one of interest, and context words surround the target. We will denote the target word as w_t and denote a context word as w_c. To find a context word, a window of size k is used around w_t. This window would include the k words before w_t and the k words after w_t, as shown in the following example and as defined in Mikolov, Sutskever, Chen, Corrado, & Dean (2013) and Pennington et al. (2014).

$$\text{The } \overbrace{\underbrace{\text{dog}}_{w_c}\ \underbrace{\text{barked}}_{w_c}\ \underbrace{\text{loudly}}_{w_t}\ \underbrace{\text{at}}_{w_c}\ \underbrace{\text{the}}_{w_c}}^{\text{Window of size } k=2} \text{ mailman.}$$

2.1 Bag of Words

As mentioned in Chapter 1, we discussed the simplest form of transforming text to numerical information is the Bag of Words method, where all words are treated as independent from one another. *Independence* here means the context words have no connection with the target word. Thus, the words be-

Table 2.1: Document-Term Matrix Example

	at	barked	cat	dog	loudly	mailman	meowed	the
Statement 1	1	1	0	1	1	1	0	2
Statement 2	1	0	1	1	0	0	1	2

fore and after have no effect on the context of the target word. This is a major assumption. It allows us to treat each word individually without taking into account the context. Using this method, we can create the document-term matrix (DTM) (Kwartler, 2017). Each row in the matrix is a document/statement in our data set. Each column presents a unique word found in the collection of documents. The word count per statement appears in the cell of the matrix.

For a very basic example, suppose we have two statements (i.e. documents) as follows:

- The dog barked loudly at the mailman.
- The cat meowed at the dog.

These two statements would make the document-term matrix seen in Table 2.1. The columns are in alphabetical order to match the default settings in the `text2vec` package in `R` (Selivanov, Bickel, & Wang, 2022). The words "the," "at," and "dog" show up in both statements, but each word is only represented in a single column.

This document-term matrix can easily be used in classification methods. Each column (i.e. unique word) can be an individual variable used to help classify the document. Thus, the number of unique words contained in the collection of documents is the number of predictors used. In many applications, it is important to filter words such that only important words (i.e. distinguishing words) are used to classify documents. This filtering can be done a number of different ways and can likely change results of our classification. Too much filtering could lead to a significant word not being included, whereas too little filtering could lead to too much noise. Effects of filtering will be explored with fake news classification in Chapter 6. For instance, one of the common filtering techniques is to remove stop words from the documents. Stop words are common words used frequently in text, such as “the,” “but,” and “through.” Because these words are used frequently, they are considered to be additional noise. They tend to not be informative to the document itself.

A major drawback of the Bag of Words method is the loss of information resulting from the assumption of independence. We lose the ability to draw context about a word given the words that surround it. This led to the development of word embedding methods attempting to maintain this information.

2.1.1 Term Frequency - Inverse Document Frequency

As an extension of the Bag of Words method, a few values can be calculated from the DTM to capture the information available in text. We can calculate the term frequency (TF), the inverse document frequency (IDF), and the term frequency-inverse document frequency (TF-IDF), all of which are based on the counts located in the DTM. TF measures how often a term appears within a document (Silge & Robinson, 2021). This is calculated as follows:

$$TF(w_i) = \frac{w_{id}}{\Sigma_{j=1}^{V} w_{jd}}$$

where w_{id} is the total number of appearances of word i in document d. In the denominator, we have the length (i.e. total number of words) of document d, and where V is the total number of unique words in the collection.

This measurement does not take into account how often the word shows up across all documents. By using TF only, words that show up many, many times will have a large value, but those words actually do not tell us much about what the specific document is about. For instance, words such as “and” or “the” appear often in a single document, but they do not provide useful information. One way of handling this issue (other than filtering out these words from the data set as part of data preprocessing procedures) is to use the IDF.

The IDF measures how important a word actually is to the collection of documents, and it is calculated as follows:

$$IDF(w_i) = \log(\frac{D}{\Sigma_{k=1}^{D} I_{w_{ik}}})$$

where log is the natural logarithm and D is the total number of documents in the collection. In the denominator, we are counting the total number of documents in which word i appears (Silge & Robinson, 2021). By combining the IDF measure with the TF measure, we address the problem mentioned earlier by decreasing the weight of terms occurring frequently but not providing useful information. IDF also increases the weight of rare, informative words (Silge & Robinson, 2021). TF-IDF is formally calculated as

$$TF - IDF(w_i) = TF(w_i) \times IDF(w_i)$$

.

The issues discussed with the Bag of Words method are still present despite the additional information weighting TF by IDF provides. We still assume all words are independent and ignore the semantic relationship between words.

2.1.2 Principal Component Analysis on Bag of Words

We can apply Principal Component Analysis (PCA) to reduce the dimension of the DTM matrix from Bag of Words to determine the topics contained in the corpus of documents. We decompose the DTM such that

$$X_{D\times V} = Z_{D\times K} B_{K\times V}$$

where

- K is the number of dimensions explaining a high percentage of the variation,
- $Z_{D\times K}$ is the PC scores matrix with the principal component scores per document,
- $B_{K\times V}$ is the loadings matrix.

The PC loadings in $B_{K\times V}$ allow us to determine which words are associated with each principal component, providing an indication of what topic is represented by that PC. We can then explore the PC scores in $Z_{D\times K}$ to determine which PCs have higher weights per document, allowing us to determine topics of each document based on the associations found in the weight matrix (Bikienga, 2018; Landauer & Dumais, 1997). To utilize this as an embedding method, we run PCA on our training corpus of documents, and we use the transposed loadings matrix as our embedding matrix. Thus, to get our document embeddings, we take $X_{D\times V}B^T_{K\times V}$, where we multiply the word frequency by the associated PC loading and sum across the component.

2.2 Word2Vec

Word2Vec is a neural network approach to quantifying textual information created by Mikolov, Sutskever, et al. (2013). This method incorporates two different types of relationships between words: paradigmatic and syntagmatic. **Paradigmatic** relations are the individual concepts each word represents (i.e. the part of speech), and **syntagmatic** relations are the context of a word inferred by the surrounding words (Torregrossa et al., 2021). These relationships contain the bulk of the information excluded by the independence assumption of the Bag of Words method. We maintain two important aspects of a target word: the information provided by the context words and the information of the target word itself. For instance, if the target word is "barked," we are able to maintain that this word is a verb while also adding in information that "dog" shows up in the same window.

There are two Word2Vec models: Skip-Gram (SG) and Continuous Bag of Words (CBoW). In the SG model, the goal is to predict context words given the target word, while the CBoW model aims to do the opposite. Depending on the language of your collection of documents, the CBoW model has been shown to outperform the SG model in terms of capturing relationships, which makes the model choice a parameter needing to be optimized (Grave, Bojanowski, Gupta, Joulin, & Mikolov, 2018; Torregrossa et al., 2021). Here, we will focus on the SG model, as the CBoW is similar. In both cases, the neural network is shallow, and it contains only an input layer, a single hidden layer, and an output layer.

In the Word2Vec SG model, seen in Figure 2.1, the input to the neural network is known as a one-hot word vector representation. A one-hot vector representation is a row vector of length, V, which is the size of the vocabulary. The vocabulary is the set of unique words extracted from the collection of documents. The one-hot word vector contains a value of 1 at w_t and 0 everywhere else. When this input vector is entered into the neural network, it acts as a searching tool for a randomly initialized weight matrix. This weight matrix is size $V \times N$, where is N is a user-specified number of dimensions. This is inputted into the hidden layer. There is no activation at this hidden layer, so this layer is just the row of the input weight matrix. We calculate a score vector by using an output weight matrix of size $N \times V$. We can denote this as follows:

$$\underline{h}' W_I W_O = \underline{z} \tag{2.1}$$

where $\underline{h}'$ is the one-hot vector, W_I is the input weight matrix, W_O is the output weight matrix, and $\underline{z}$ is the final score vector of size $1 \times V$. Finally, the softmax activation is used to calculate a probability for every possible context word in the vocabulary. This is the output layer (Chaubard, Fang, Genthial, Mundra, & Socher, 2019; Karani, 2020). The softmax activation is $\sigma(\mathbf{z}_j) = \frac{e^{z_j}}{\sum_{i=1}^{V} e^{z_i}}$, where z_j is the j^{th} observation of the score vector.

Since the goal of the neural network in the SG model is to learn the proba-

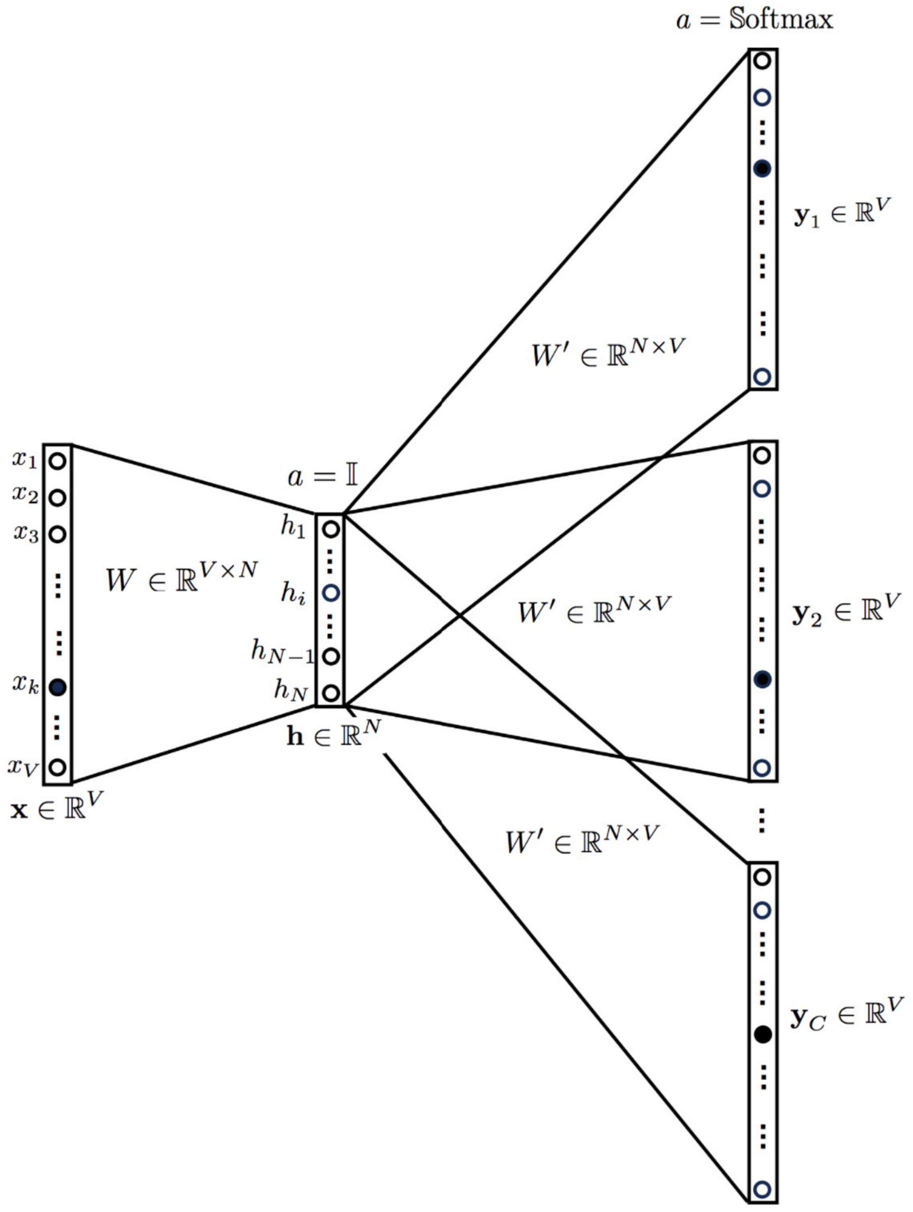
$a = \mathbb{S}\text{oftmax}$
$\mathbf{y}_1 \in \mathbb{R}^V$
$W' \in \mathbb{R}^{N\times V}$
$a = \mathbb{I}$
x_1
x_2
x_3
x_k
x_V
$\mathbf{x} \in \mathbb{R}^V$
$W \in \mathbb{R}^{V\times N}$
h_1
h_i
h_{N-1}
h_N
$\mathbf{h} \in \mathbb{R}^N$
$W' \in \mathbb{R}^{N\times V}$
$\mathbf{y}_2 \in \mathbb{R}^V$
$W' \in \mathbb{R}^{N\times V}$
$\mathbf{y}_C \in \mathbb{R}^V$

Figure 2.1: Skip-Gram Model diagram from Karani (2020)

bility of a context word appearing with the target word, once the output layer has been calculated, a loss function must be applied to determine how accurate the probability estimate is. If this loss function is above a specific criterion, we work backwards in the neural network to update the weights in order to more accurately capture the probability. This updating process is done using stochastic gradient descent (Chaubard et al., 2019). This updating will only effect the row of the input matrix associated with w_t. We repeat this process for each word in our vocabulary, meaning we have V total one-hot vectors. After the updating is complete and the loss function has been minimized for all words, we actually discard the output layer. We are interested in the learned input weight matrix, that is size $V \times N$, as our word embedding matrix.

Because the softmax activation function involves several dot products to find z_j, the function cannot easily be solved in polynomial time. In Mikolov, Sutskever, et al. (2013), they proposed a solution called Negative Sampling. The main idea of this method is to add random context words into the vocabulary and detect them. The method will classify target-context pairs that actually exist in our collection of documents and those randomly generated. At the end, word pairs that are real (called positives) are geometrically close together, whereas word pairs that were randomly generated (called negatives) are not (Torregrossa et al., 2021).

As previously stated, the output layer is not of interest, and it only serves as a means to an end. We want to use the output layer to help us update the input weight matrix. Once we have minimized loss for all V word input vectors, the final input weight matrix is our word embedding matrix we have been trying to retrieve. These word embeddings were optimized to capture the probability a context word appears with the target word. Thus, the embeddings have captured the relationships between pairs of words. Words appearing in similar contexts should have similar embeddings based on the optimization. The dimensions, which were user specified, show those relationships - similar values for the same dimensions indicate a relationship. However, what those dimensions represent are not typically interpretable. Despite this, Mikolov, Sutskever, et al. (2013) did address the drawback of the Bag of Words method.

Recall this matrix is size $V \times N$. Since V is the size of the unique words in our vocabulary and N is the number of dimensions in which we have embedded, we only get a single vector per word, i.e. our word embedding. We have no knowledge of the document/statement level information, which is what we are trying to classify. Therefore, some form of aggregation is needed in order to use these embeddings in a classification task. Aggregating based on the mean, minimum, or maximum of each dimension are popular and widely used methods to find document-level representations (Boom et al., 2016). This aggregation is completed over the rows of the embedding matrix that are associated with the unique words in each document.

An issue with the Word2Vec method is the focus on local information. By focusing on local information, we lose global information that allows us to generalize results better (Torregrossa et al., 2021). The focus is very narrow due to only one-hot vectors being the input into the neural network. It raises the question of what information we lose when we ignore global information. Additionally, the size of the embedding dimension is selected by the user and is a parameter typically optimized for performance. Typically, a value around

300 is selected. Of course, the more dimensions selected, the higher the quality of the information captured.

2.3 Global Vectors for Word Representation (GloVe)

To address the shallow focus of only local information of the Word2Vec method, Pennington et al. (2014) developed a method that incorporates global information. The GloVe method gathers word co-occurrence frequencies from the entire collection of documents, hence global information is captured. The method is based on a co-occurrence matrix, which is a square matrix where the number of rows and columns are equal to V (Pennington et al., 2014). The co-occurrence matrix, denoted as C, records the number of windows where w_c and w_t both occur. When creating the co-occurrence matrix, we will assume the row denotes the target word, and the column denotes the context word. Currently, we will be focusing on a window size of 5, where we are looking at the 5 words directly prior and directly after the target word. This is the default window size with the GloVe procedure in the `R` package `text2vec` (Selivanov et al., 2022). We also assume a symmetric window, which ensures the co-occurrence matrix is symmetric as well.

Let us look at a trivially small example with a window size of 1. Suppose we have the following two sentences:

- The dog barked loudly tonight.
- The cat meowed loudly tonight.

We begin by assigning each unique word a number. In `R`, by default, this assignment is done in alphabetical order. For our trivial example, the

Table 2.2: Simple Co-Occurrence Example

			Context						
		Word Number	barked	cat	dog	loudly	meowed	the	tonight
	barked	1	0	0	1	1	0	0	0
	cat	2	0	0	0	0	1	1	0
	dog	3	1	0	0	0	0	1	0
	loudly	4	1	0	0	0	1	0	2
Target	meowed	5	0	1	0	1	0	0	0
	the	6	0	1	1	0	0	0	0
	tonight	7	0	0	0	2	0	0	0

numbering would go as follows:

- $\overbrace{\text{The}}^{6}\ \overbrace{\text{dog}}^{3}\ \overbrace{\text{barked}}^{1}\ \overbrace{\text{loudly}}^{4}\ \overbrace{\text{tonight}}^{7}.$
- $\underbrace{\text{The}}_{6}\ \underbrace{\text{cat}}_{2}\ \underbrace{\text{meowed}}_{5}\ \underbrace{\text{loudly}}_{4}\ \underbrace{\text{tonight}}_{7}.$

By doing this, the rows and columns of the co-occurrence are in alphabetical order as well, making it fairly straight forward to make. Since each unique word is assigned a number, the number will correspond to the row and column it is assigned to. For instance, the word “loudly” would be in row 4 and column 4. Suppose “loudly” is our target word. The window around “loudly” would look like:

- The dog $\overbrace{\text{barked loudly tonight}}$.
- The cat $\underbrace{\text{meowed loudly tonight}}$.

In these window, both “barked” and “meowed” appear once, and “tonight” appears twice. The co-occurance matrix will reflect this with 1 in $C_{4,1}$ and $C_{4,5}$ and 2 in $C_{4,7}$. To see the complete co-occurrence matrix for this simple example, see Table 2.2.

The main idea behind GloVe is to model a ratio between probabilities. Let the cell of the co-occurrence matrix be denoted as x_{tc}, the number of times w_c appears in context of w_t. We can calculate the empirical probability of this relationship appearing by finding $P(w_c|w_t) = \frac{x_{tc}}{x_t}$, where $x_t = \Sigma_{i=1}^{V} x_{ti}$. The ratio of interest is between two target words, say w_{t_1} and w_{t_2}, and the probability of a specific w_c appearing with the respective target words. This looks like $\frac{P(w_c|w_{t_1})}{P(w_c|w_{t_2})}$. The closer this ratio is to one, the less of a distinguishing context word w_c is. The larger the ratio, the more w_c is related to w_{t_1}. A value closer to zero means w_c is related more to w_{t_2}. The ratio allows us to figure out which words are relevant or irrelevant to distinguishing the context of it (Pennington et al., 2014). For an example, see Table 2.3. Clearly, the word solid is more related to the word ice, hence the large ratio value. The word water is equally related to both ice and steam, which makes the ratio approximately 1.

Table 2.3: Co-occurrence probabilities and ratios from a 6 billion word corpus (Pennington et al., 2014; Sahil, 2021).

	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$\frac{P(k\|ice)}{P(k\|steam)}$	8.9	8.5×10^{-2}	1.36	0.96

Instead of utilizing these ratios directly, GloVe attempts to approximate the ratios by embedding into vectors of dimension N. Therefore, a function is needed for approximation that meets specific criteria. The input of this function is the embedded vectors for w_{t_1}, w_{t_2}, and w_c. Let us denote these embedded vectors as v_{t_1}, v_{t_2}, and v_c respectively. The first criteria of the function F is to restrict it to only consider the difference between v_{t_1} and v_{t_2}, such that

$$F(v_{t_1} - v_{t_2}, v_c).$$

Additionally, F needs to output a scalar. Since F is a function of vectors, the dot product is utilized. Now

$$F((v_{t_1} - v_{t_2})' v_c) = \frac{P(w_c|w_{t_1})}{P(w_c|w_{t_2})},$$

which can be rewritten as

$$F((v_{t_1} - v_{t_2})' v_c) = \frac{F(v'_{t_1} v_c)}{F(v'_{t_2} v_c)}.$$

Lastly, F needs to be interchangeable between context words and target words. A natural choice is the exponential function, which means

$$P(w_c|w_{t_1}) = F(v'_{t_1} v_c) \leftrightarrow v'_{t_1} v_c = \log(x_{t_1 c}) - \log(x_{t_1}).$$

In the previous equation, the $\log(x_{t_1})$ violates this ability to exchange the distinction between target and context word. Since it does not depend on the context word, it is absorbed into a bias vector, b_{t_1}, and an additional bias vector, b_c, is added in order to maintain the interchangeability between a target and context word (Pennington et al., 2014; Sahil, 2021).

To measure loss, GloVe utilizes weighted least squares regression. They use a weight function, $f(x_{tc})$, to address the issue that arises with $\log(x_{tc})$. Recall the formulation of the co-occurrence matrix described earlier. Most of the x_{tc} values will be 0, and the natural logarithm will not be well defined. The weight function should have the following properties (Torregrossa et al., 2021):

- $f(0) = 0$ and $\lim_{x \to 0} f(x) \log^2(x)$ is finite
- f should increase
- f should be small for large values

Pennington et al. (2014), the developers of the GloVe procedure, suggest using

$$f(x) = \begin{cases} (\frac{x}{x_{max}})^{\alpha} & \text{for } x < x_{max} \\ 1 & \text{Otherwise.} \end{cases}$$

The word embedding vectors are considered optimal when the following is minimized (Pennington et al., 2014; Torregrossa et al., 2021):

$$\sum_{t,c=1}^{V} f(x_{tc})(v_t' v_c + b_t + b_c - \log(x_{tc}))^2.$$

A unique feature of this method is we get two different embeddings for a word, v_t and v_c. We get an embedding for when the word was considered the target word and when it was a context word. To get a single word embedding vector per word, we find $v_t + v_c'$ per word. This can then be used similarly as Word2Vec for document classification by aggregating word vectors by the mean, minimum, or maximum. These word embeddings encapsulate local information by using a window around each target word and global information by tabulating the results for the entire document. While the main idea is similar to Word2Vec in terms of capturing relationships between words in a collection of documents, GloVe has two main advantages. It includes global information, and negative sampling does not need to be utilized in order to train the word embedding vectors (Torregrossa et al., 2021). As a result, GloVe is more computationally efficient. Nonetheless, the user still defines the number of dimensions, as discussed in Section 2.2. As with Word2Vec, 300 dimensions is a standard choice.

2.4 Bidirectional Encoder Representations from Transformers (BERT)

The previous methods of Word2Vec and GloVe are *unidirectional* methods, meaning they only look at context from a single direction. Text information is inputted sequentially into these models, restricting the direction of context (Horev, 2018). For instance, with Word2Vec, the objective of negative sampling is to determine a correct context word from incorrect context words, specifically assessing correctness based on position to the left or right of the target word (Devlin, Chang, Lee, & Toutanova, 2019; Mikolov, Sutskever, et al., 2013). To address this, Devlin et al. (2019) developed the Bidirectional Encoder Representations from Transformers (BERT), in which the input is completed for the whole sequence at once, allowing for the context to be generated from all surroundings (Horev, 2018). This allows words with multiple meanings, such as "bank," to have different embeddings related to the different meanings. Thus, a river bank and a bank that holds money would be embedded differently since the surroundings of these words would be different.

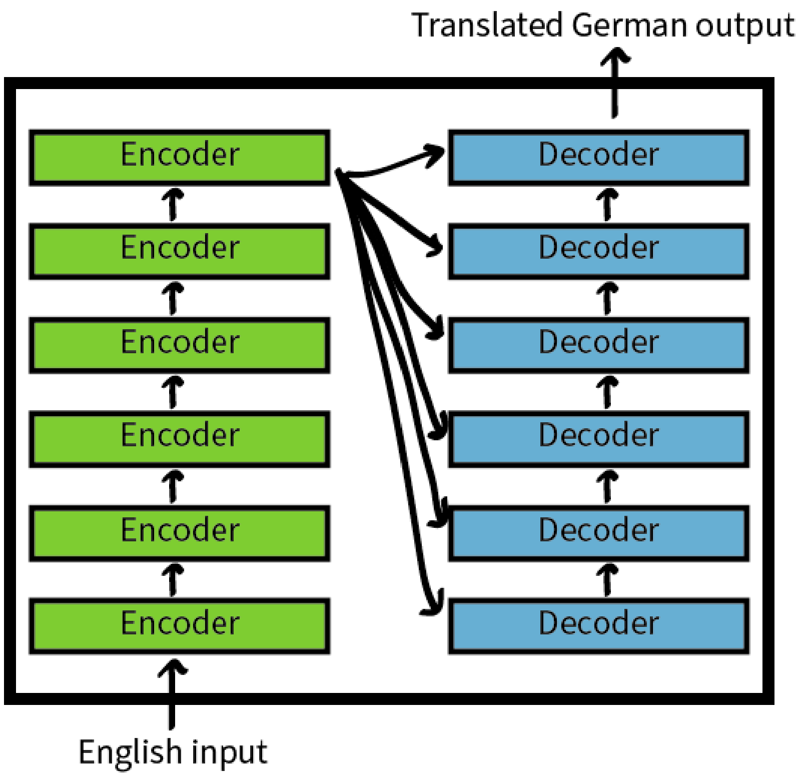

Figure 2.2: A simple transformer at a glance. BERT only contains the left hand side of a transformer (the encoders), but BERT-base contains 12 encoders instead of 6.

The foundation of BERT is a deep neural network known as a transformer. A transformer contains Encoder and Decoder components, which one can consider as translation device (see Figure 2.2). Consider the Encoder as person who speaks English and the Decoder as a person who speaks German. The Encoder takes in an input in English and translates it to a shared language with the Decoder. The Decoder then translates from that shared language into German. This shared language between the Encoder and Decoder must be learned between them. In the original development of the transformer, each component contained a stack of six encoders and decoders respectively

(Alammar, 2018b). BERT only contains the stack of encoders, and only the structure of the encoding stack will be discussed in this dissertation.

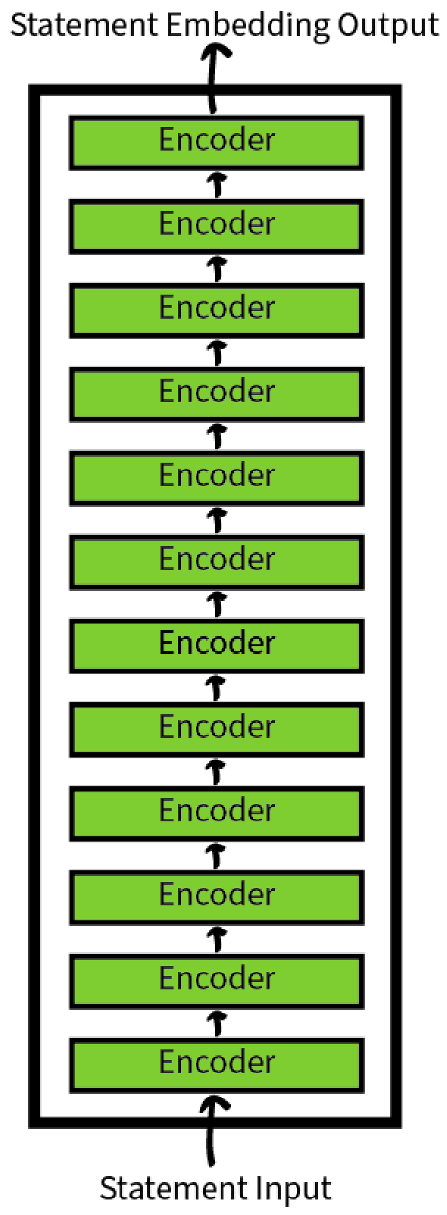

Figure 2.3: A glance at the structure of BERT-Base with a stack of 12 encoders. BERT-Large contains double this many encoders.

BERT is a pre-trained model available for users to access word embeddings trained on a large amount of data from books and Wikipedia entries. Two of the pretrained models are BERT-Base and BERT-Large. For each subsequent model, the creators of BERT (Devlin et al., 2019) extended the stacks from six encoders to 12 and 24 encoders respectively (see Figure 2.3). Each of the encoders in the stack are identical in structure, but each encoder contains separate weights. Within each encoder, there are two sub-layers as seen in Figure 2.4: self-attention and a feed forward neural network (Alammar, 2018a). At a high level, one can think of the self-attention layer as a method of having the Encoder component learn how to read context clues. When we read a statement containing the word “it,” we have the capability of using context clues from before or after “it” to determine what “it” is referencing.

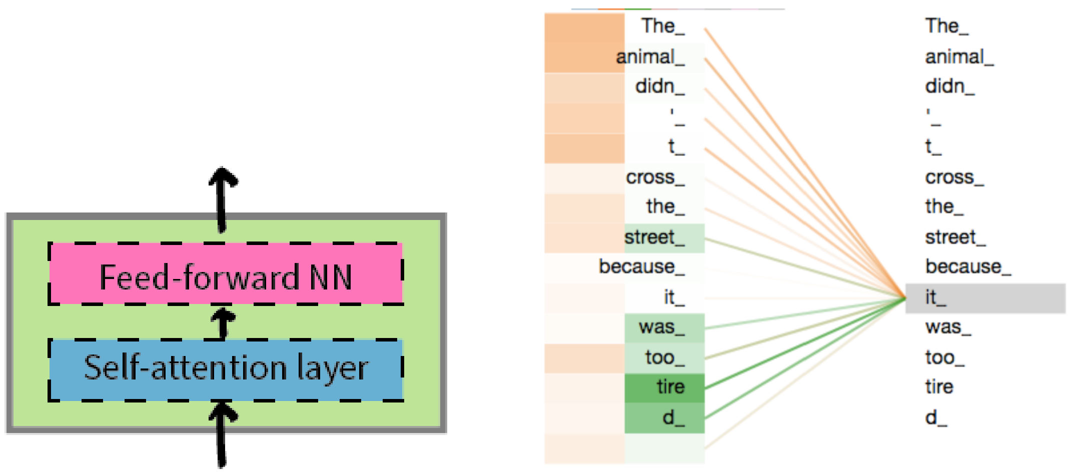

Figure 2.4: On the left, a glance at the sub-layers contained within each of the encoders. On the right, an idea of how the attention head works. The attention head represented by the orange color finds ‘it’ to be mostly related to ‘the animal’, where as the green attention head finds ‘it’ to be mostly related to ‘tired.’ Image from (Alammar, 2018b).

BERT utilizes positional encoding to help the model learn where the word is located in the sentence, which improves its learning of context clues (see Figure 2.5). BERT Base contains 12 attention heads and 768 hidden units per encoding element. Attention heads can be considered a cross-validation approach to learning context clues. Each attention head may find a different relationship, or context, but over all of them, the best representation of the context is likely to be identified. The idea can be seen in Figure 2.4 The 768 hidden units are the dimensions in which words are being embedded (Alammar, 2018a). Unlike Word2Vec and GloVe, the dimension size of the pre-trained BERT model is not user-defined and cannot be changed.

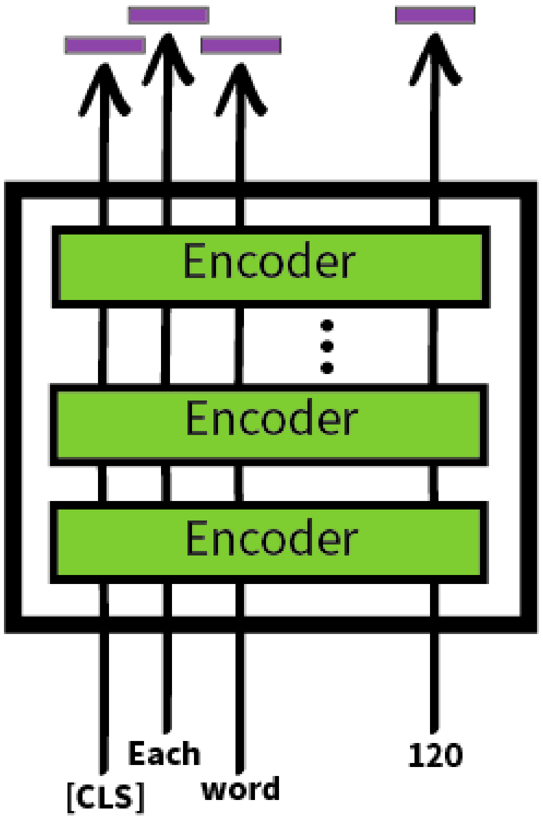

Figure 2.5: The [CLS] token (stands for classification) denotes the beginning of the sentence to be embedded. It helps BERT learn the order of the words to better learn context. BERT outputs an embedding vector per word of size 768, represented in this image with the purple rectangle. The '120' represents the max sequence length inputted into the model.

The training of the model includes two steps: Masked Language Model and Next Sentence Prediction. In the masked language model, 15% of the tokens in the input are masked as seen in Figure 2.6. The goal is to predict the missing word based on the output of the position of the missing word (Alammar, 2018a). This again helps improve the model's ability to gain information about a particular word given the words that surround it. The model will find the word with the embedding most closely related to the output of the missing token embedding. Within this same step, words will also randomly be replaced by a different word, and the model must learn which word has been replaced (Alammar, 2018a). This again helps improve the model's ability to capture the relationships between the words. In addition to the relationships between words, the model is interested in capturing relationships between sentences. Context from one sentence can be carried over to the next, therefore understanding the order of sentences can be helpful. The model will be trained in such a way it finds the probability Sentence B follows Sentence A (Alammar, 2018a).

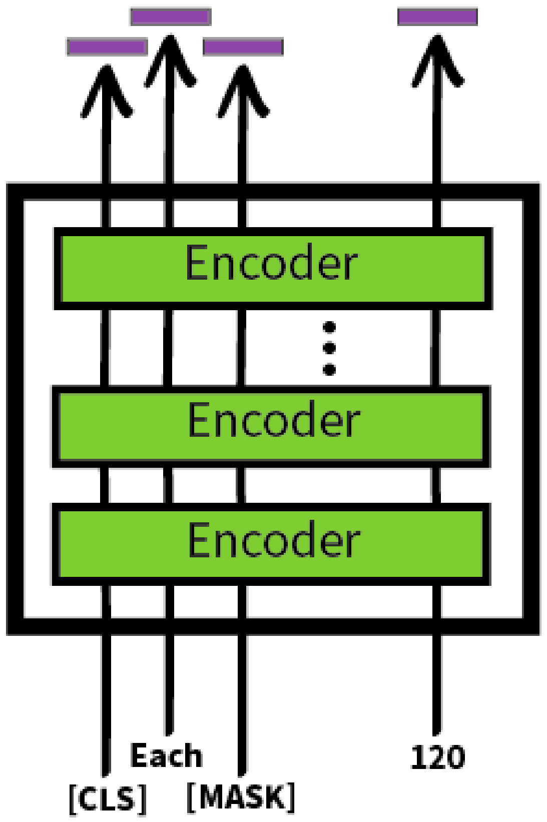

Figure 2.6: The [MASK] token masks a word from the model, and BERT attempts to fill in with the appropriate word given the outputted embedding. This is completed during training of the model.

For the purposes of this dissertation, the pre-trained embeddings are used via the BERT-Base model, a mean pooling layer, and the sentence transformer function developed by Reimers & Gurevych (2019). This will take the model, trained as described above, and average the word embeddings together in order to form sentence embeddings, which will then be used for classification.

Chapter 3

Review of Current Fake News Detection Methods

This chapter serves as a review of the current methods being used to classify news as either real or fake. In Chapter 1, we defined fake news as news that is untruthful and written in such a way it is intended to cause anger and outrage. We begin by looking at methods used in detecting opinion spam, which helped build some of the methods we use with fake news detection. Fake news detection and the success some researchers have had will then be explored. In this chapter, unless otherwise specified, assume the bag-of-words embedding method was used to translate text into numerical quantities.

3.1 Opinion Spam

Opinion spam is where users leave a review on an item, such as a hotel or a good bought on Amazon.com, meant to be perceived as real in order to deceive the reader. It has become common for companies to hire people to write positive reviews to outshine some of the negative ones; another tactic is to use automatic bots programmed to leave positive reviews. In other cases, these reviews may be used to promote or demote a different, but similar, product. Many human users are unable to determine whether a review is real or fake, but consumers would undoubtedly prefer to read true reviews when determining whether to purchase a product.

Within opinion spam detection, researchers have access to what is considered the *gold standard* data set created by Ott, Choi, Cardie, & Hancock (2011). This data set was constructed to contain 400 truthful and 400 gold standard deceptive positive reviews. To create these gold standard deceptive reviews, the authors enlisted the work to a marketplace of small tasks to be completed by people with basic programming skills. Each person was only allowed to submit one entry to ensure each review was done by a unique user. Each user was assigned a name and website of a hotel (one of the top 20 Chicago hotels). They were asked to pretend they worked for the marketing department of the hotel, and their boss asked them to write a review. For the truthful reviews, the authors collected reviews from the same top 20 Chicago hotels. After eliminating reviews that were not 5 stars, not English, less than 150 characters, and reviews written by first-time authors, the authors were left with 2,124 truthful reviews. To select 400, the authors made sure the document lengths were similarly distributed as the collected deceptive reviews. To model the distribution of document lengths, a log-normal distribution truncated at 150 characters was used. A log-normal has been shown to be an appropriate model for document lengths (Serrano, Flammini, & Menczer, 2009).

Ott et al. (2011) focused their research on comparing the use of n-grams to keyword based detection in classifying reviews. A n-gram is a set of features from the document, where the feature(s) could be a word or group of words. A unigram is a single word, where a bigram is a set of two words (Kwartler, 2017). In the work of Ott et al. (2011), a unigram and a bigram are used, and

Table 3.1: The accuracy results of the SVM classifier used by Ott et al. (2011).

	Accuracy
Keyword-based	0.768
Unigrams	0.884
Bigrams	0.896
Keyword-based + bigrams	0.898

in each case, the words are lowercase and left unstemmed. Unstemmed words, such as “like” and “liking,” are not treated as the same word because the “-ing” part is not dropped from the word. For their keyword based detection approach, the authors use a software, *Linguistic Inquiry and Word Count*, that is popular in social sciences. The software contains a list of 4500 keywords that have been grouped into 80 psychologically meaningful dimensions. Each of the 80 dimensions are used as a single feature of the review. A support vector machine was used as a classifier with both the n-gram approach and keyword-based approach to classifying reviews. Their results on the gold-standard data set they collected are summarized in Table 3.1. Unigrams and bigrams on their own improved the accuracy by over 10% when compared to the keyword-based approach. Combining bigrams with the keyword-based approach only offered minor improvement over bigrams alone. They determined using the actual words, or n-grams, improves the ability to accurately determine if a review is real or fake.

Ott et al. (2011) examined only positive leaning reviews. In a follow-up study, Ott, Cardie, & Hancock (2013) created a *gold-standard* data set for negative reviews, employing a similar methodology as the positive reviews. The n-gram based classification of reviews produce similar results as the positive reviews of Ott et al. (2011). Subsequent studies (Ahmed, Traore, & Saad, 2017; Shojaee, Murad, Azman, Sharef, & Nadali, 2013) utilize a combined data set of both positive and negative reviews.

Shojaee et al. (2013) adopted 234 stylometeric features, broadly categorized into character-based lexical, word-based lexical, and syntatic features. Character-based lexical features consist of character count, ratio of digits to character count, ratio of letters to characters, and many more; there are a total of 52 of these features. Word-based features include total number of tokens, average sentence length in terms of characters, average token length, and an additional 22 features. For the syntatic features, they count the occurrence of punctations (7 features) and occurrences of 150 function words defined in Zheng, Li, Chen, & Huang (2006). For a full list of features explored, see Shojaee et al. (2013). Naive Bayes and Support Vector Machine with Sequential Minimal Optimization with a polynomial kernel (SVM with SMO) are utilized in order to classify the reviews of the hotels. SMO is an algorithm to efficiently solve the optimization problem when training a SVM. Shojaee et al. (2013) use these methods in three feature spaces: lexical (both character-based and word-based), syntatic, and then the full set. For all three of these different feature sets, SVM with SMO outperformed Naive Bayes, with the use of all 234 stylometeric features performing the best at an accuracy of 84%. Syntactic features alone performed the worst for both classification methods (Shojaee et al., 2013). Thus, even with 234 stylometeric features, the accuracy was lower than the 89.8% reported by Ott et al. (2011). This seems to imply n-grams perform the best at distinguishing between real and fake reviews, though this difference could be due to the combined positive and negative review data set.

Ahmed et al. (2017) worked on a method that built from the basic n-

gram and hopefully improved accuracy using the data set containing both positive and negative reviews from Ott et al. (2011) and Ott et al. (2013). Using n-grams varying from $n = 1$ to 4, the term frequency (TF) and the term frequency-inverted document frequency (TF-IDF) are calculated. The TF is a measure of the observed probability of a n-gram appearing within a certain document. They count how many times a n-gram appears in the document and divide by the total number of n-grams within the document. The TF-IDF metric is discussed in Chapter 2.1.1. Using the Python natural language toolkit, Ahmed et al. (2017) compare classification using stochastic gradient descent, support vector machine, linear support vector machine, K-nearest neighbor, logistic regression, and decision trees. Only the top features were used and were selected using the TF and TF-IDF criteria, varying the threshold of inclusion from 1000 to 50,000 n-grams. The highest accuracy (90%) in distinguishing real and fake reviews was achieved using linear support vector machines and ten thousand bigrams ($n = 2$). Thus, the Ahmed et al. (2017) were able to improve on the 84% accuracy reported in Shojaee et al. (2013). Thus, it appears using the TF and TF-IDF based on the n-grams provides improvement.

3.2 Fake News Detection

There are two main approaches researchers have used to facilitate the classification of fake news utilizing the news contents: knowledge-based and style-based. In Shu et al. (2017), both approaches are reviewed. Knowledge-based approaches utilize fact checking of the major claims in a news article. There are three major categories for fact checking: expertise, crowd sourcing, and computational. Expert-oriented methods use the expertise of a person in the related field to the news article. This method is extremely time consuming and demands a lot of contacts and pooled, intellectual resources. It is therefore not feasible for large data sets. The second method is the use of crowd sourcing information from regular people who may not be experts in a specific area. Commonly, this data is acquired through the reporting of suspicious articles on social media or through comments made on a particular section of a news article on websites dedicated to detecting fake news. The crowd sourcing method combines these reports and comments to gauge the truthfulness of an article. Unfortunately, there is no guarantee the crowd is being objective. This method can only offer an overall assessment. The last method of fact checking is computational-oriented. This method attempts to automatically detect false claims by comparing statements from an article of interest to already established statements of truth. One way to gather statements of truth is through references made on open web sources already verified by an expert (Shu et al., 2017).

In style-based approaches, researchers look at stylistic differences in writing between fake and real news. Since fake news spreads misleading, distorted information, the writing style used by authors of fake news will be different than the writing style used by authors of real news. Authors of fake news want to manipulate information to deceive the reader. Therefore, researchers have explored the possibility of detecting these stylistic differences automatically in news articles. Two methods have been proposed: deception-oriented or objectivity-oriented methods. A deception-oriented stylometric method is one that attempts to capture the untruthful (deceptive) statements in an article.

One way this method approaches deception detection is by looking for structural differences between deceptive and truthful sentences. An objectivity-oriented method looks for stylistic differences between real and fake news that indicate a decreased neutrality of the news content. Real news is meant to present the facts and not spin them to mislead readers. Therefore, the further an article is from neutrality, the more likely it is fake news. These stylistic differences can be measured utilizing linguistic features (Shu et al., 2017).

Knowledge-based approaches tend to heavily rely on experts as well as a lot of trust in the overall mass of readers of the articles, and are limited to a specific skill-set of the experts. Taking a learning method and applying it to articles outside of the specific news content would not work since the knowledge (experts) needed for different contents are needed. For instance, news about politics would require different experts then news about medicine. Additionally, knowledge-based approaches can **only** be applied after publication and are typically slow. Many of the automatic approaches already developed require a data set labeled by an expert. It would be useful to develop methods solely based on the writing style of an article as a writing style can be learned for any genre article. Style-based approaches could be applied before an article goes viral (i.e. has been shared among social media members many, many times). Between the two approaches, style-based seems to be more robust in terms of classifying fake news (Shu et al., 2017). Researchers such as Horne & Adali (2017) and Potthast & Kiesel (2018) have applied style-based approaches to data sets containing political news content. However, they disagree on the ability of style-based approaches in predicting fake news. We should note writing style can differ based on the topic as well as the outlet of the news content. Exploring how those writing styles differ between topics, such as politics and medicine, or news outlets, such as *NY Times* and *Wall Street Journal*, could be a potential route for future studies but is not explored in this dissertation.

Both Horne & Adali (2017) and Potthast & Kiesel (2018) utilized data sets collected by BuzzFeed in 2016, but the data sets were different in terms of collection and ground truth. The BuzzFeed-Webis Fake News Corpus 2016, employed by Potthast & Kiesel (2018), contains a total of nine publishers, of which six are prolific hyper-partisan sources (equally split between right-wing and left-wing politics) and three are mainstream. All publishers selected earned a blue check-mark on Facebook; this blue check-mark indicates to a social media user the publisher is an authentic source, elevating the status of the publisher. The articles were published on seven workdays close to the United States Presidential Election in 2016. These seven dates are specifically September 19-23, 26, and 27.

The articles contained in the BuzzFeed-Webis Fake News Corpus 2016 were all fact-checked by BuzzFeed journalists. Two journalists would review an article, and then they would rate it as either "mostly true," "mostly false," "mixture of true and false," and "no factual content." Journalists were given the following guidelines:

- A rating of "mostly true" is given when the article and any included links or images with the article are based on fact. This information is presented accurately. Authors are able to interpret the factual information as they please, just as long as they do not misrepresent the information. This means they cannot make unsupported claims and speculations.

- A rating of “mixture of true and false” contains factually accurate information, but there are elements of unsupported claims and speculation. This label is used when the title of the article contains a false claim, and the associated article contains accurate presentation of information. This rating is only used when the false information is approximately equal to the true information, and when authors’ use unconfirmed information.
- A rating of “mostly false” consists of articles where most or all of the information presented is inaccurate.
- A rating of “no factual content” consists of mostly satirical and pure opinion articles. This category is also used for comics, videos, and social media posts of the “Like this if you think…” variety.

If a reviewer gives a rating of “mostly false” or “mixture of true or false,” the reviewer has to justify the rating. If there was doubt regarding a rating or a disagreement between the two reviewers, a third was consulted. An article being given a “mostly false” rating was given one final check to make sure the rating was justified in case a publisher were to argue. In the news industry, long-term publishers update their websites frequently. Therefore, in order to maintain access to the corpus created by BuzzFeed, Potthast & Kiesel (2018) archived posts, linked articles, and other data. Of the original data set, 1627 articles were recovered: 826 mainstream (*ABC News*, *CNN*, *Politico*), 256 left-wing (*Addicting Info*, *Occupy Democrats*, *The Other 98%*), and 545 right-wing (*Eagle Rising*, *Freedom Daily*, *Right Wing News*). Figure 3.1 contains the breakdown of the labels per source.

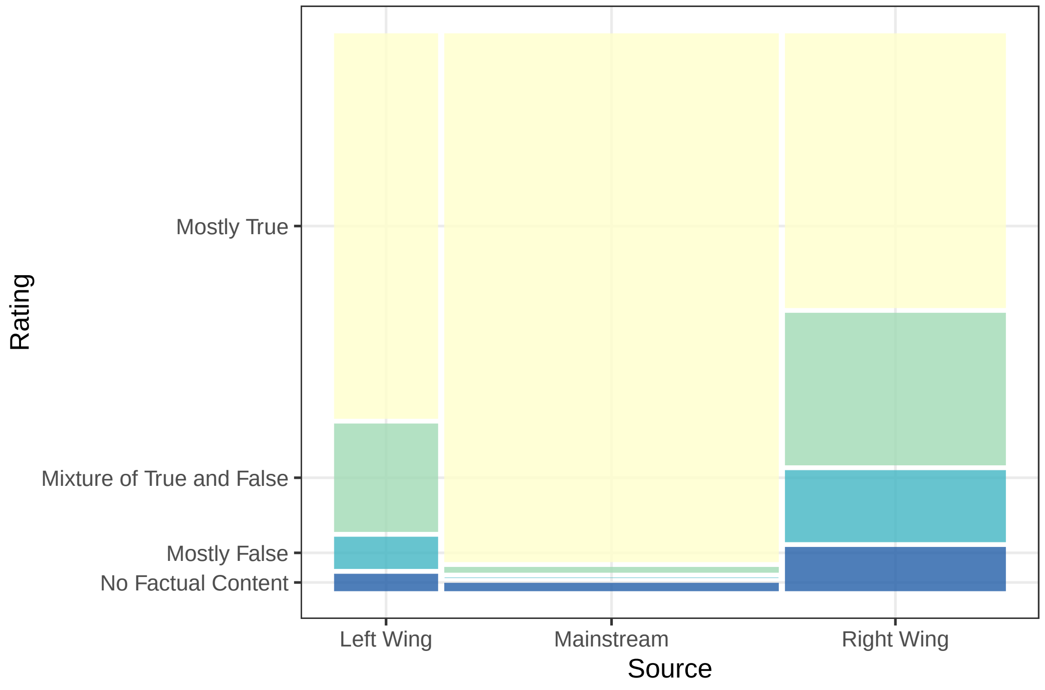

Figure 3.1: The breakdown of labels given to the articles in the BuzzFeed corpus (Potthast & Kiesel, 2018). There were no articles from mainstream sources given a rating of mostly false. We can see majority of the false news is coming from hyper-partisan sources.

Based on the corpus construction, it is clear there are imbalances between the ratings. There are multiple ways of handling this imbalance, but one way is to combine the rating of mixture of true and false with the rating of mostly false (Potthast & Kiesel, 2018). In practice, fake news is hardly ever presented without some kernel of truth. Typically, all articles given a rating of n/a were disregarded, unless they were deemed satirical writing. After this, Potthast & Kiesel (2018) gathered different stylistic features. These features include *n*-grams of size 1 to 3 (of characters, stop words, and parts-of-speech), readability scores, dictionary features (frequency of words), ratio of quoted words and external links, number of paragraphs, and average length. A random forest was implemented for classification.

When attempting to classify news as either fake or real, all articles written by mainstream portals were disregarded as none of those articles had been labelled as mostly false and very few were labelled as a mixture. Thus, only hyper-partisan news was used to develop the classifier. Predictions were made across the orientation of an article as well as based on orientation-specific information. However, the probability of correctly classifying a statement based on stylistic features was only 55% for both sets of predictions. Thus, Potthast & Kiesel (2018) concluded style-based fake news classification is not very useful in general.

By combining the ratings of “mixture of true and false” with “mostly false,” an aspect of “mixture of true and false” has been ignored: articles with a false/misleading headline but largely accurate content. If the bulk of the article itself is true, then this could affect the accuracy of the resulting classifier. The classifier will be trained to understand an actually mostly true article under a classification label of mostly false. Therefore, it seems logical to include the information from titles in the classifier in hopes there is improvement since the misleading information will now be included, which was done by Horne & Adali (2017). The data used by Horne & Adali (2017) was from BuzzFeed, but it was at a smaller scale of the BuzzFeed-Webis Fake News Corpus 2016 and collected slightly differently. The data set contains only articles receiving high engagement on social media (i.e. lots of comments, shares, likes, etc.), and they were collected from known fake and real news sources. They collected 60 real and 60 fake articles, but they filtered out opinion based and satirical articles. After filtering, there were only 36 real and 35 fake articles (Horne & Adali, 2017). Taking ground truth as the portal it came from was clearly not accurate, based on the breakdown of the article classifications seen earlier by Potthast & Kiesel (2018).

The stylistic features used in this study, measured for both articles and titles, consist of the number of part of speech tags, stop words, punctuation, quotes, negations, informal/swear words, interrogatives, and words in all capital letters. They also included "complexity" features such as number of words per sentence, readability scores, and ratio of unique words to total words (Horne & Adali, 2017). Because there are a lot of features to use to build a classifier, the authors conducted multiple one-way ANOVA tests (or a Wilcoxon rank sum when the normality assumption failed) to determine which features were significantly different between real and fake news. After determining the top 4 features that were significantly different between real and fake news articles and titles, a Support Vector Machine with a linear kernel model was fit to identify real and fake news. The top 4 features found for articles were the number of nouns, ratio of unique words to total words, word count, and number of quotes; the SVM model achieved 77% accuracy. The top 4 features found for titles were the number of nouns, percent of stop words, average word length, and the readability score measured by the Flesh-Kincaid grade level index; the SVM model achieved 71% accuracy (Horne & Adali, 2017). The features Horne & Adali (2017) explore are easily manipulated. Thus, developing classifiers based on these features may not be useful for very long because authors can change their writing styles (Potthast & Kiesel, 2018).

In Section 3.1, Ahmed et al. (2017) was discussed in regards to detection of opinion spam. In the paper, they also explored their methods and how they would apply to fake news detection. For their news data set, they combine a data set containing only fake news from websites that PolitiFact deemed unreliable (available at kaggle.com) and a data set collected from Reuters.com, which is considered to be reliable and true. All articles were related to political news topics. They achieved results indicating TF-IDF performs better than TF and found unigrams with a linear-based classifier provided the best performance. With 50,000 of the top features, TF-IDF, and unigrams, they achieved a 92% classification accuracy. In contrast to the opinion spam detection, not much accuracy is lost between bigrams and trigrams. Only fourgrams caused a drastic drop in accuracy. On the BuzzFeed data set used in Horne & Adali (2017), Ahmed et al. (2017) achieved a classification accuracy of 87% using linear SVM and unigrams. This greatly outperforms the results seen in Horne & Adali (2017) and demonstrates n-grams may outperform text-based features.

Based on the success of n-grams in the classification of fake news, it is of interest to explore how capturing the context of a word effects classification. With n-grams, we can assume some sort of context is captured in the window. Through multiple natural language process methods, such as Word2Vec, GloVe, and BERT, the context of word is captured in the word embeddings they create. Thus, it seems to be a reasonable hypothesis these methods should outperform the traditional bag-of-words methods. We hypothesize the individual word embedding methods will distinguish between real and fake news, indicating word choice alone is different between them.

Chapter 4

PolitiFact Data

The methods explained previously will be compared based on their standalone ability to classify fake news. We are only interested in how well words and the relationships between words can be used to classify real and fake news. To complete this task, we will be using a data set from PolitiFact.

In order to combat fake news related to or spread by politicians, the non-profit, nonpartisan company PolitiFact was formed. PolitiFact is owned by the non-profit Poynter Institute for Media Studies, which is the parent company to *Tampa Bay Times*. PolitiFact developed the Truth-O-Meter rating system to evaluate statements made by politicians. This rating system contains six different levels: true, mostly true, half true, mostly false, false, and pants on fire. A rating of true indicates the statement is accurate with nothing significant missing. Mostly true signals the statement is accurate, but it requires clarification and additional information. If the statement is partially accurate, it receives a rating of half true. It is either missing an important detail or takes something out of context. A mostly false statement has an element of truth in it, but it ignores a critical fact that gives a different impression. An inaccurate statement is given a rating of false. Lastly, a rating of pants on fire is also not accurate but in turn makes a ridiculous claim (Holan, 2020). Note, in the data set we are using, mostly false is instead labeled “barely true.”

Each statement goes through the same, extensive review process. The assigning editor gives a reporter a statement flagged for fact checking. Based on the information available at the time the statement was made, the reporter conducts research to support their rating. This review is conducted using primary sources and original documentation, requiring direct access to government reports, academic studies, etc. The reporter must only rely on information they can verify independently. After the reporter makes a decision on what they believe would be the correct rating, they return to the assigning editor, and the two of them come to an agreement on the rating (Holan, 2020).

Once an agreement has been reached between the assigning editor and the reporter, the assigning editor takes their agreed upon rating to two additional editors. The three editors and the reporter discuss four major questions:

- Is the statement literally true?
- Is there another way to interpret the statement? Is it open to interpretation?
- Does the speaker of the statement provide any evidence and prove the statement to be true?
- How has PolitiFact handled previous, similar statements?

After discussion, the three editors vote on a rating, which is either the original rating the reporter gave or a vote to change it. Only 2 editors are needed to pass a rating on a statement (Holan, 2020). Thus not all ratings are agreed upon unanimously before getting published on PolitiFact’s website.

As one can see, a statement is thoroughly vetted before the report is published for public viewing. PolitiFact decides which statements they will review based on submissions from the public in addition to any statements journalists feel need to be fact checked. They check statements they feel are the most significant and newsworthy. As with the vetting of statements, there is a checklist a statement must meet in order to even be considered (Holan, 2020):

1. The statement is verifiable and rooted in fact. If the statement is opinion, it is not reviewed. As these are statements by politicians, PolitiFact editors do allow for some hyperbole, as long as the hyperbole is not misleading.
2. Is the statement misleading or does it sound wrong?
3. If the statement is significant, would a typical person wonder if the statement is true?
4. Does the statement appear to be one that will be passed on and repeated?

This is the critical idea behind fake news as defined earlier. There are specific types of statements that will spread faster and further than others. If a statement appears to likely be spread, it is important to fact check it as soon as possible.

PolitiFact is a non-profit organization, which operates solely from donations. This can lead to a conflict of interest when choosing statements, but PolitiFact has a very strict code of conduct for their journalists in addition to disclosing donations of a certain size. PolitiFact journalists are to avoid public expression of political opinion and public involvement in public processes such

	Real			**Fake**			
FNN	8767 (50.6%)			8557 (49.4%)			17324
LIAR	2403 (13.9%)	3096 (17.9%)	3268 (18.9%)	2897 (16.7%)	3648 (21.1%)	2012 (11.6%)	17324
	True	**Mostly True**	**Half True**	**Barely True**	**False**	**Pants on Fire**	

Table 4.1: Breakdown of label frequency. 259 statements in LIAR but not FNN were exclued from the analysis.

as voting. This applies to all full-time staffers, correspondents, and interns. To remain nonpartisan, they try to select equal number of statements made by Democrats and Republicans, but the political party in power does tend to be fact checked more often (Holan, 2020). People and companies who make large donations are named on their website.

Based on the credibility of PolitiFact and their rating process, Sadeghi, Jalaly Bidgoly, & Amirkhani (2020) collected statements and ratings available on the website from 2007 to April 26, 2020. They created two data sets; the first maintains the original labels given by PolitiFact, which was denoted as the LIAR data set. The second places the labels into two groups: real and fake. "Real" is given to statements receiving a true, mostly true, or half true rating. "Fake" is given to statements that received a mostly false, false, and pants on fire rating. This data set was labeled as the Fake News Net (FNN). We will use the FNN data for Chapters 5 and 6, where we evaluate performance of a binary classifier. The LIAR data containing original PolitiFact ratings will be explored in Chapter 7. The researchers who collected the data already randomly split the set into training, validation, and evaluation data sets. In order to avoid potential bias from the researchers, all three sets were combined, and we created our own split for training and evaluation based on a preset and recorded seed.

The breakdown of the data set is found in Table 4.1. Using their data set, we extract the statements PolitiFact placed under investigation and the classification label given to it. We ignore the research done by PolitiFact journalists which supports why a given rating was given. We remove all punctuation, new line spacing syntax, and all non-alphanumeric symbols for all parts of this dissertation. Other data preprocessing methods are explicitly explained in future chapters.

Chapter 5

Comparing Methods for Classifying Fake News

In this chapter, we compare the word embedding methods' ability to capture textual information by comparing their stand alone ability to classify fake news utilizing the PolitiFact FNN data set. We will additionally be comparing how well these different classification methods work using multiple performance metrics. We ignore other stylistic features, such as number of nouns, punctuation, etc. We want to know purely how treating words as independent versus capturing relationships between words affects this classification. Note we are assuming each dimension of PCA, both Continuous Bag of Words (CBoW) and Skip-Gram (SG) Word2Vec models, GloVe, and BERT are not highly correlated as this is an assumption of the classification methods used. The use of embedding methods with respect to the classification models are described later in this chapter. We will first review the classification methods utilized in this dissertation and then explore the results of the embedding and classification methods.

5.1 Classification Methods

Multiple classification analyses are applied to compare each embedding methods to determine if a specific classification analysis and embedding method combination works better at classifying fake news. We treat the response as binary as follows:

$$Y = \begin{cases} 1 & D \in \text{Real} \\ 0 & \text{Otherwise} \end{cases}$$

where D is the document. Methods include logistic regression, linear discriminant analysis, quadratic discriminant analysis, mixture discriminant analysis, flexible discriminant analysis, classification tree, and random forest. We explore the performance of the common classification methods with classifying our binary response. As stated in Chapter 4, a classification of "Fake" actually consists of multiple labels from PolitiFact, as does "Real." Thus, extending past a binary response for some of these classification methods is of interest, and this will be explored in Chapter 7.

Predictor variables for each classification method are defined as follows. With Bag of Words and TF-IDF, the predictors are the frequency value or the TF-IDF value of each unique word in the vocabulary of the collection, respectively. For PCA, CBoW, SG, GloVe, and BERT, we find individual vectors for each unique word in the vocabulary, which are the length of the dimension embedded. To get statement level embeddings, for each word in the statement, we use element-wise addition to sum together the individual word vectors. Each embedded dimension is used as a predictor variable.

5.1.1 Logistic Regression

Logistic regression is an approach for a categorical response variable. The purpose of logistic regression is to predict the probability of an observation being in group 1, or in our case, the probability of a document being real news. We do this by fitting the logit to our typical regression model as follows:

$$\log\left(\frac{p_i}{1-p_i}\right) = \underline{x}_i'\underline{b}$$

where p_i is the probability of being in group i, $\underline{x}_i$ are the predictors used to estimate the logit, and $\underline{b}$ are the parameters associated with each predictor. Maximum likelihood is used to estimate $\underline{b}$.

To train the regression model, we utilize observations where group membership is known to get estimates of $\underline{b}$. We test how well the logistic regression model is working by using observations of "unknown" group membership and predicting the probability of being in group 1.

Unlike other classification methods, logistic regression does not minimize the expected cost of misclassification. Sometimes classifying an observation as being in group 1 when it is really in group 2 can be costly. In our case, this is equivalent to classifying a statement as real news when it really is fake news. We need to decide which direction of misclassification is more costly (Johnson & Wichern, 2008). Currently, we are assuming the cost of misclassification is the same. Is the cost of classifying real news as fake worse than classifying fake news as real? Both directions have different, but significant, consequences. With both types of misclassification, people are going to have a hard time trusting real, actual facts. It is believed treating the cost of misclassification as the same is reasonable for this dissertation.

5.1.2 Discriminant Analyses

Multiple discriminant analyses are executed in the attempt to classify fake news. There are two goals of a discriminant analysis. The first is to develop a discriminant rule to separate the groups as much as possible, and the second is to use this rule to classify observations with an "unknown" group membership (Johnson & Wichern, 2008). This is different from logistic regression as it has a deterministic prediction of group classification. Logistic regression provides a probability of group 1 membership, and the user must determine a cutoff threshold.

Linear discriminant analysis (LDA) creates a linear function from all predictor variables to classify the response as best as possible. This linear function is found by maximizing the difference between the means of group 1 and group 2. A new observation is classified based on the closest group mean. In LDA, we assume the observation vectors are $N(\underline{\mu}_i, \mathbf{\Sigma})$, where $\underline{\mu}_i$ is the mean vector of the i^{th} group. It is also assumed $\mathbf{\Sigma} = \mathbf{\Sigma}_1 = \mathbf{\Sigma}_2$. In other words, we have homogeneous covariance matrices across group membership. Additional assumptions include equal cost of misclassification, and the prior probability of group membership is equally likely. If we believe the last two assumptions are violated, the discriminant rule can be updated accordingly (Johnson & Wichern, 2008). As discussed earlier, the assumption about the cost of misclassification being equal seems to make sense when trying to classify fake news. Based on the PolitiFact FNN data set, where we have approximately 50/50 real and fake labels, the prior chance of group membership is also reasonable. However, it is important to note this assumption may not extend outside of this data set since it is unclear how much fake news is actually in the population of political statements. Generalization is not possible.

As noted, covariance matrices are assumed to be homogeneous between the two groups, but there are times this assumption might be in question. In those cases, quadratic discriminant analysis (QDA) can be utilized. We now assume the observation vectors are $N(\underline{\mu}_i, \mathbf{\Sigma}_i)$, where where $\underline{\mu}_i$ is the mean vector of the i^{th} group and $\mathbf{\Sigma}_i$ is the covariance matrix of the i^{th} group. We now have the ability to assume $\mathbf{\Sigma}_1 \neq \mathbf{\Sigma}_2$. We maintain the assumption of the prior probability of group membership being equal, and the cost of misclassification being equal. If the covariance matrices are in fact equal, QDA reduces to LDA (Johnson & Wichern, 2008).

Additionally, in LDA, we assume observations from each class are from a single Gaussian distribution. However, Hastie & Tibshirani (1996) proposed this might be too restrictive in some cases, and some observations actually come from a mixture of Gaussian distributions. The class/group is actually made up of subclasses that are $N(\underline{\mu}_{ji}, \mathbf{\Sigma})$, where $\underline{\mu}_{ji}$ is the mean vector for the j^{th} subclass of the i^{ih} group. We assume each subclass has the same covariance structure. This type of discriminant analysis is called Mixture Discriminant Analysis (MDA). In Figure 5.1, this idea of Gaussian mixtures is presented visually for two groups. Group 1, presented in blue, is a mixture of two Gaussians, and Group 2 is a mixture of three. Each Gaussian has a common variance of 3 (Li, 2017).

Finally, Flexible Discriminant Analysis (FDA) is also applied, as this form of discriminant analysis allows for a non-parametric approach to finding the

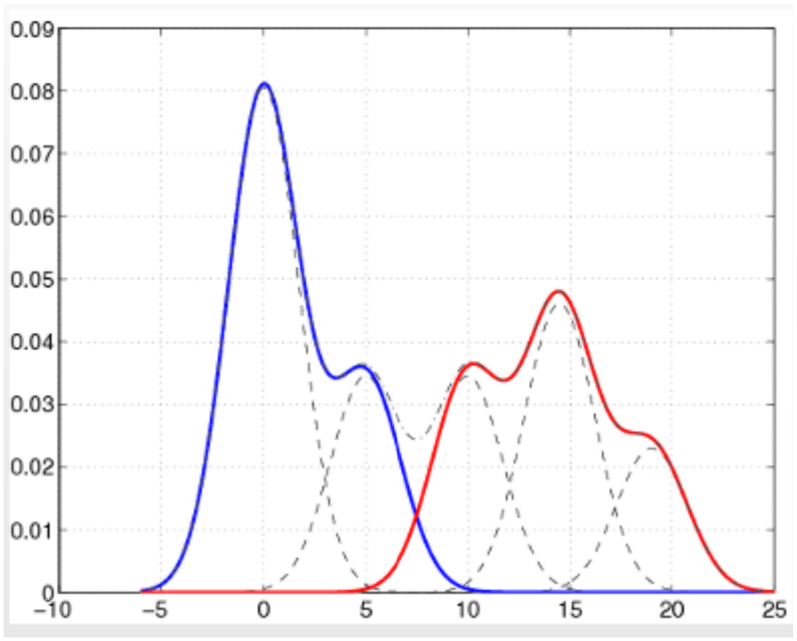

Figure 5.1: A two-class example of Gaussian mixtures from Li (2017)

boundary between the two groups (Hastie, Tibshirani, & Buja, 1994). This method applies non-parametric regression approaches, such as splines, in order to estimate non-linear boundaries between the groups. This allows for more flexibility with the assumptions of LDA, QDA, and MDA as we no longer rely on the Gaussian distribution. Essentially, FDA performs a version of LDA in an extended space, and then projects it back down into two dimensions, which is the archetype of Support Vector Machines (Hastie, Tibshirani, & Friedman, 2009).

5.1.3 Classification Trees and Random Forests

One of the final classification methods exploited is a classification tree. A classification tree is a set of if/then logic statements to determine group membership of an observation. These are a useful tool to assist in classification as the approach is non-parametric and non-linear. A classification tree consists of multiple nodes, where each node consists of a logical if/then statement. Since they just consist of these logic statements, we do not have the underlying assumption of a linear relationship between the predictors and response. Usually, a classification tree only results in a few nodes, and those nodes show which predictor variables seem to relate most with the response. Classification trees are sometimes simpler to interpret than some of the other methods discussed above (Sungur, 2011).

Classification trees are "grown" through binary recursive partitioning. We start at the base node, and if we meet the condition, we follow branch one. If we do not meet the condition, we follow branch 2. Within both of these branches, a new logical statement occurs. In other words, the if/then statement at the node on branch one may not be the same statement as on branch two. The tree continues to branch until one reaches a good stopping point. One common issue with classification trees is over-fitting. If you let the tree continue to "grow" as tall as possible, it will potentially create a rule to fit every single observation perfectly. Thus, typically you grow the tree to that point and then "prune" the tree by collapsing nodes to avoid over-fitting the data (Sungur, 2011).

With a classification tree, there is no guarantee the tree you grow is the "best" tree. To combat this issue, random forests (RF) were developed. A random forest consists of many classification trees, which then vote on an outcome. These trees are fit using subsets of variables and data, leading to more robust outcomes during the tree growing process (Breiman & Cutler, 2004). This addresses the issue of a single classification tree by creating an ensemble method resulting from multiple trees.

		Actual Rating		
		Real	**Fake**	
Predicted Rating	**Real**	True Positive (TP)	False Positive (FP)	Precision/Positive Predictive Value
	Fake	False Negative (FN)	True Negative (TN)	
		Sensitivity/True Positive Rate	Specificity/True Negative Rate	Accuracy

Table 5.1: Confusion matrix for a binary response.

5.2 Applying the PolitiFact Data Set

For each classification and embedding method combination, four performance metrics are utilized to measure the efficiency of classifying fake news: accuracy, precision/positive predictive value, sensitivity/true positive rate, and specificity/true negative rate. In Table 5.1, we see an outline of a confusion matrix generated from a binary classification method. The four metrics are calculated directly from this matrix as follows.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \tag{5.1}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.2}$$

$$\text{Sensitivity} = \frac{TP}{TP + FN} \tag{5.3}$$

$$\text{Specificity} = \frac{TN}{TN + FP} \tag{5.4}$$

Equation 5.1 measures the probability of correct classifications out of all classifications done by the method. Equation 5.2 is the probability of correctly predicting a true real statement out of all statements predicted as real. If precision is large, the more real news is being accurately predicted as real (true positive). If precision is small, then more fake news is being predicted incorrectly as real (false positive). Equation 5.3 is the probability of correctly classifying a true real statement. If sensitivity is large, real news has been correctly classified as real. If it is small, then real news has been incorrectly classified as fake. Lastly, Equation 5.4 is the probability of correctly predicting a fake label (DataTechNotes, 2019). If specificity is high, fake news has been correctly identified as fake, whereas if it is small, fake news has been incorrectly identified as real.

For this chapter, the PolitiFact FNN data set was cleaned with the following details. As a reminder, all punctuation, new line spacing syntax, and all non-alphanumeric symbols are removed. Beyond this, all characters are transformed to lower case letters, and any numbers are removed, such as year. Common English words, called stop words, are removed as well. The list of stop words within the `tidytext` package in `R` contains 1,149 words, such as "she," "her," "problem," and "member" as a few examples (Silge & Robinson, 2016). Lastly, any word appearing less than 5 times in the entire data set is removed. The effect of some of these preprocessing measures will be explored in Chapter 6. After preprocessing and randomly splitting the data set, we are left with 13,852 documents in the training set and 3,463 documents in the test set. Each classification model is trained with 10-fold cross-validation.

After these preprocessing methods, we are left with a total of 4,298 unique words across both training and test sets. This means for Bag of Words and TF-

IDF methods, classification methods have 4,298 predictor variables. However, the matrices associated with both of these methods are highly sparse, even with the data preprocessing steps taken, and many of the classification models did not be converge due to the amount of zeros in the data. The only methods which produced valid models were logistic regression and LDA, and even those should be examined with caution.

With the PCA method, we found 1900 PCs were sufficient in explaining 90% of the document term matrix. Thus, we reduce the dimension to 1900, and we use the method described in Section 2.1.2 to use this in our classification methods. BERT by default embeds into 768 dimensions. As discussed in Chapter 2, Word2Vec CBow, Word2Vec SG, and GloVe require the user to specify the number of dimensions to embed. For these methods, 768 dimensions was selected to compare to BERT in addition to popular choices of 100, 200, and 300 dimensions. Since all these methods involve reducing the number of dimensions, we are left with a dense matrix with these embedding methods, and all classification models can be fit with these embedding methods.

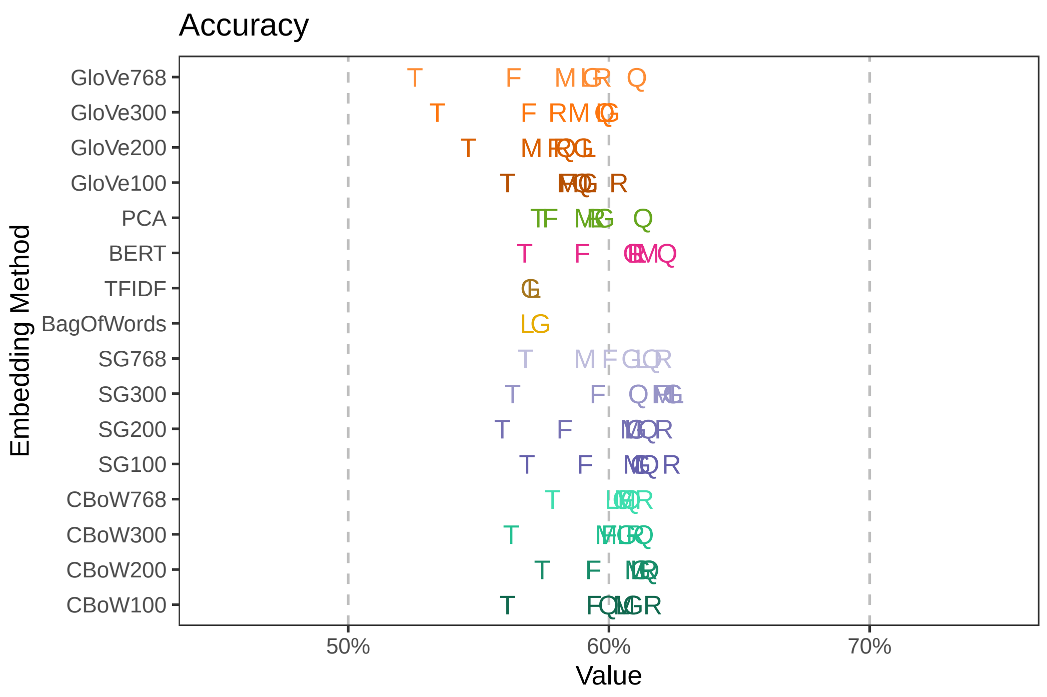

Figure 5.2: Accuracy metric for each classification and embedding method combination. Each classification method is represented by a letter. T is for classification tree, and G is for logistic regression. All others are based on the first letter of the method name.

Plots depicting accuracy, precision, sensitivity, and specificity for each embedding and classification method are seen in Figures 5.2, 5.3, 5.4, and 5.5 respectively. Let's first focus on accuracy. All methods perform similarly at a little over 60%, except for the classification tree method. For the classification models Bag of Words and TF-IDF work with, they consistently under-perform compared to the other embedding methods. In Table 5.2, we see SG300 performs the best when averaged over all the classification models at 0.6089 (0.0227). We see, according to Table 5.3, QDA performs the best averaged over all embedding methods at 0.6081 (0.0111). A more interesting tidbit would be which combination produces the highest accuracy, which is SG300 and LDA with 0.626. This can be seen Tables 5.4 and 5.5. Thus, no method significantly outperforms guessing whether a statement is real or fake in overall accuracy.

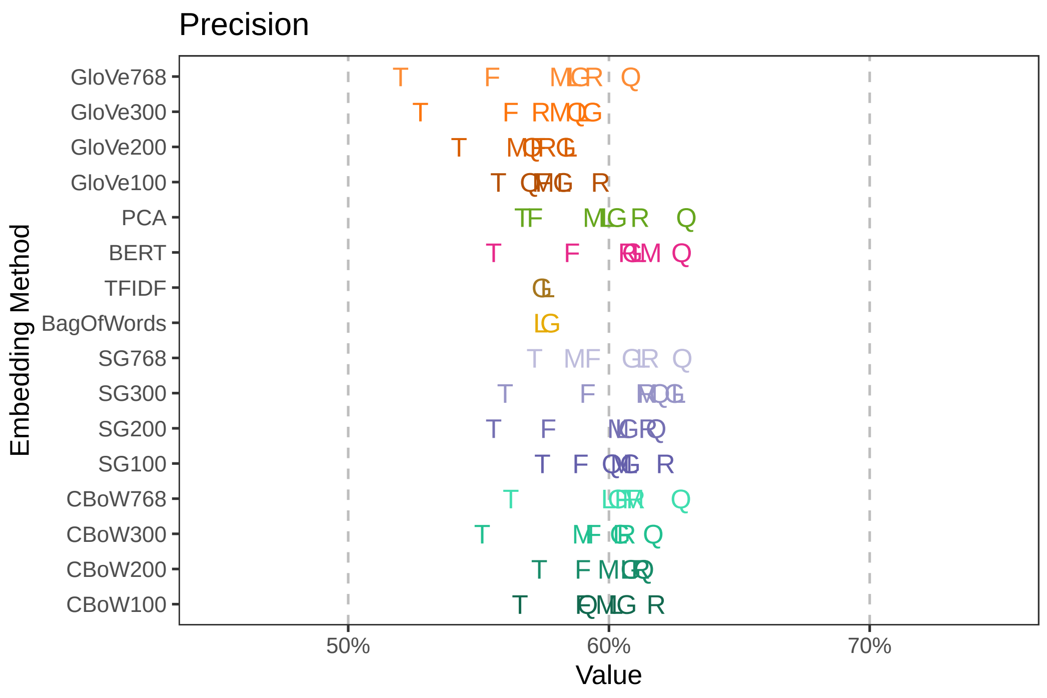

Figure 5.3: Precision metric for each classification and embedding method combination.

As for how precise our classification models are performing per embedding method, we see much the same as we did with accuracy. All method combinations perform right around 60%. SG300 performs the best at 0.6075 (0.0238) when averaged over all the classification models. QDA performs the best at 0.6079 (0.0207) averaged over all embedding methods. The highest precision occurs for combination PCA and QDA with 0.6297. The probability of correctly classifying a true real statement out of all predicted real statements still seems like the method is not doing much better than guessing if the statement is real or fake.

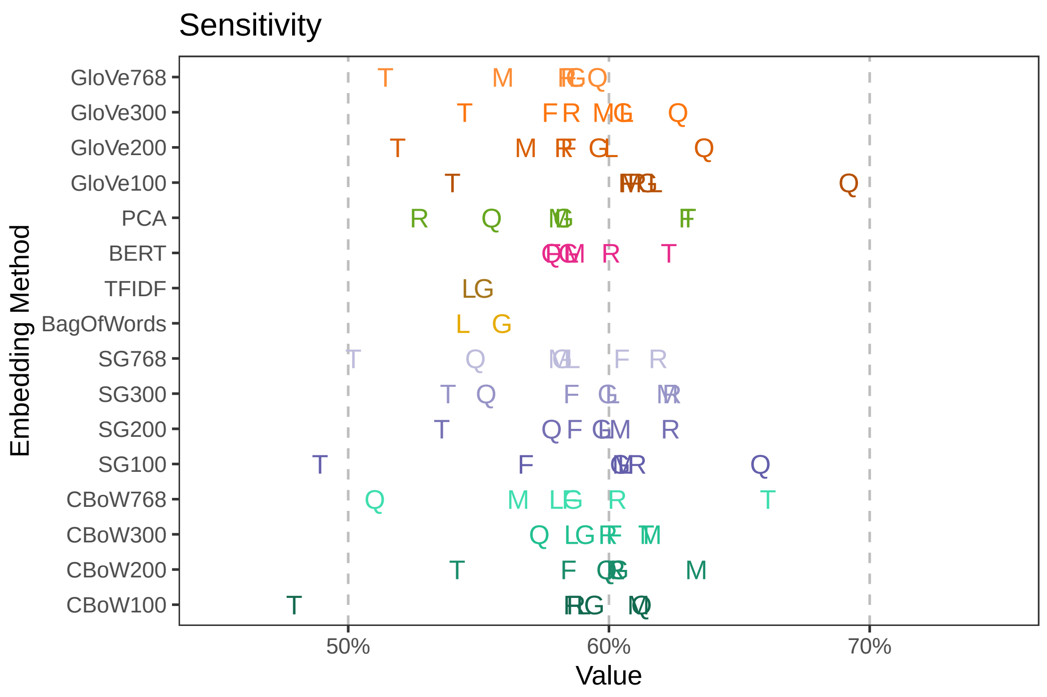

Figure 5.4: Sensitivity metric for each classification and embedding method combination.

With the metric sensitivity, we start to see some interesting things. Classification trees produce sensitivity values over 60% for some methods, which is somewhat surprising given its accuracy and precision results. One can also see QDA performs very well. In fact, the maximum sensitivity value occurs with the combination GloVe100 and QDA with 0.692. If you are interested in predicting real news correctly out of all actual real news, the previous combination would perform the best for this objective. On average, GloVe100, with a value of 0.6129 (0.0441), is the highest across all classification methods. Across all the embedding methods, RF performs the best on average at 0.5972 (0.0246). However, it is not expected these averages are different from the other methods for both embedding and classification methods.

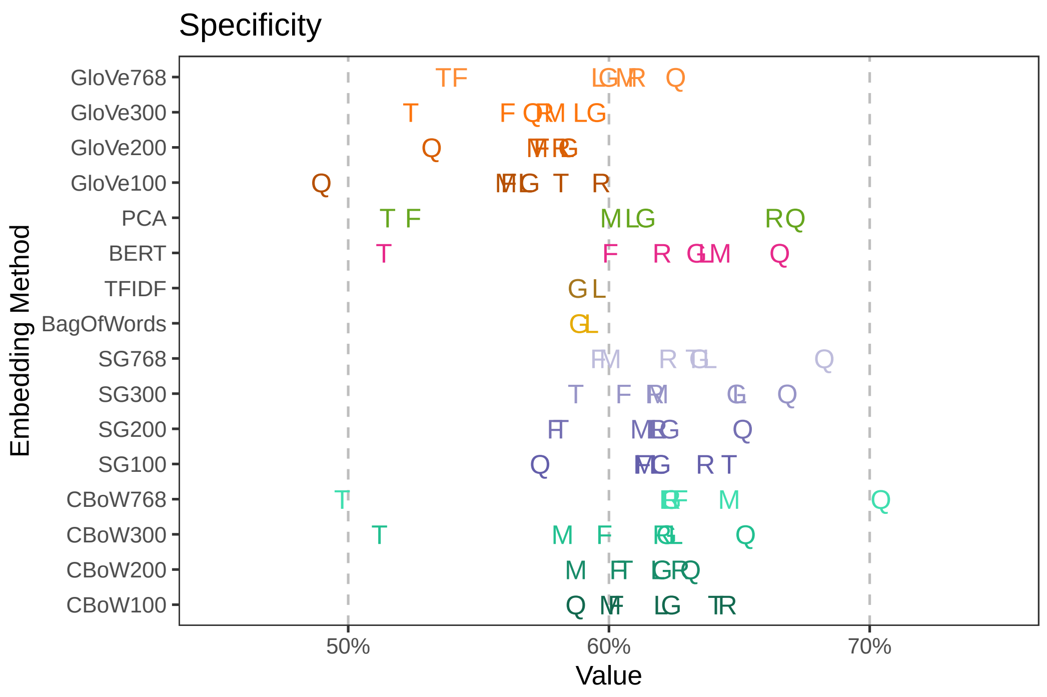

Figure 5.5: Specificity metric for each classification and embedding method combination.

Lastly, with the specificity metric, we can find the combination that detected fake news the best. This combination was CBoW768 and QDA with 0.7043. We have decent success in correctly classifying fake news. We can also look at the averages for the embedding methods and classification methods. Across the classification methods, SG768 was best with a value of 0.6296 on average (0.0287). Then, on average, QDA performs the best across the embedding methods at 0.6219 (0.0624). Yet again, it is not expected, however, for these averages to be different from one another for both the embedding and classification methods.

If we take into account the sensitivity and specificity metrics, it appears we may have some good combinations of embedding methods and classification methods if one is interested in predicting real news or fake news well respectively. However, we need to look at these results *together*. With the QDA classification method and GloVe in 100 dimensions, we had a pretty high sensitivity rate compared to other classification and embedding methods. When we look at specificity, the rate for QDA and GloVe in 100 dimensions is the smallest, with a value below 50%. We see a similar reciprocal relationship when looking at QDA and Word2Vec Continuous Bag of Words in 768 dimensions, except this time sensitivity is one of the lowest and specificity is the highest. Classification trees also show a similar pattern of methods with high sensitivity rates having low specificity. This indicates that for some embedding methods, the classification method is not actually learning what the difference is between real and fake news. The classification method is just predicting majority of statements as being real for those high with sensitivity and low specificity, or vice versa. This is something to keep in mind when viewing these results.

From all these metrics, we see the most success in classifying fake news accurately or real news accurately but only marginally for both. QDA appears as part of the "best" combination for 3 out of the 4 metrics, leading us to think the covariance matrix of each group is not equal. As was pointed out, however, these maximum values we are seeing for sensitivity and specificity do not tell the whole story. When looking at those metrics together, we start to see a pattern evolving of the classification method just classifying every statement one way or the other depending on the embedding method. Remember we were wanting to see how well the different embedding methods performed to know if capturing relationships between words improves these classification performance metrics. We see there really is only marginal differences between the embedding methods themselves, as seen in Figures 5.2, 5.3, 5.4, and 5.5. This leads one to believe that capturing the relationships between words may slightly have an effect on the ability to classify news as real or fake as compared to Bag of Words and TF-IDF but not so much to make a significant difference. It is worth to note though that this comparison is only able to be made with logistic regression and LDA. Capturing relationships does allow for more efficiency in the fact we do not have to worry about sparsity and a classification method being able to calculate all important metrics. Within the classification methods where all the embedding methods capturing relationships, GloVe appears to be performing the worst across the board on average. The type of classification method does not seem to make much of a difference, except not to use classification trees on their own.

We must keep in mind these results are depend on the data preprocessing methods utilized. Recall we filtered out words appearing fewer than 5 times across the data set, converted words to lower case, removed numbers, and filtered out stop words. What if the stop words are used differently between real and fake news? What about numbers? Capital letters? Is there too much or too little filtering? These data preprocessing methods could potentially have important ramifications on the results. With embedding methods capturing relationships, every part of the preprocessing process impacts how the models define relationships between words, which in turn affects the classification method. We explore the effects of these data preprocessing decisions in the next chapter.

Table 5.2: Average metric value and standard deviation by word embedding method

	Accuracy		Precision		Sensitivity		Specificity	
Embedding Method	Average	SD	Average	SD	Average	SD	Average	SD
CBoW100	0.5989	0.0181	0.5963	0.0164	0.5801	0.0458	0.6173	0.0216
CBoW200	0.6046	0.0152	0.6006	0.0145	0.5955	0.0278	0.6135	0.0154
CBoW300	0.5996	0.0172	0.5954	0.0213	0.5974	0.0154	0.6018	0.0454
CBoW768	0.6028	0.0114	0.6026	0.0198	0.5844	0.0449	0.6208	0.0617
SG100	0.6047	0.0188	0.6011	0.0152	0.5917	0.0523	0.6174	0.0230
SG200	0.6008	0.0219	0.5974	0.0227	0.5892	0.0275	0.6122	0.0250
SG300	0.6089	0.0227	0.6075	0.0238	0.5892	0.0329	0.6281	0.0286
SG768	0.6026	0.0183	0.6025	0.0194	0.5748	0.0388	0.6296	0.0287
BagOfWords	0.5712	0.0037	0.5757	0.0027	0.5515	0.0106	0.5910	0.0033
TFIDF	0.5706	0.0008	0.5754	0.0016	0.5492	0.0041	0.5922	0.0057
BERT	0.6038	0.0188	0.6019	0.0239	0.5910	0.0160	0.6162	0.0494
PCA	0.5919	0.0134	0.5967	0.0220	0.5840	0.0372	0.5998	0.0609
GloVe100	0.5867	0.0131	0.5770	0.0122	0.6129	0.0441	0.5611	0.0340
GloVe200	0.5777	0.0157	0.5708	0.0144	0.5840	0.0357	0.5716	0.0182
GloVe300	0.5812	0.0236	0.5739	0.0230	0.5920	0.0262	0.5707	0.0235
GloVe768	0.5808	0.0283	0.5765	0.0296	0.5732	0.0283	0.5883	0.0346

Table 5.3: Average metric value and standard deviation by classification method

	Accuracy		Precision		Sensitivity		Specificity	
Classification Method	Average	SD	Average	SD	Average	SD	Average	SD
LogReg	0.6011	0.0144	0.5991	0.0141	0.5900	0.0164	0.6119	0.0218
LDA	0.6003	0.0151	0.5983	0.0146	0.5889	0.0201	0.6115	0.0222
QDA	0.6081	0.0111	0.6079	0.0207	0.5940	0.0482	0.6219	0.0624
MDA	0.5988	0.0144	0.5943	0.0152	0.5958	0.0226	0.6017	0.0246
FDA	0.5877	0.0123	0.5824	0.0138	0.5903	0.0156	0.5850	0.0289
Tree	0.5602	0.0151	0.5561	0.0163	0.5524	0.0568	0.5677	0.0516
RF	0.6080	0.0143	0.6053	0.0148	0.5972	0.0246	0.6187	0.0232

Table 5.4: Maximum metric value by word embedding method

Embedding Method	Accuracy	Precision	Sensitivity	Specificity
CBoW100	0.6168	0.6181	0.6125	0.6455
CBoW200	0.6154	0.6134	0.6335	0.6313
CBoW300	0.6133	0.6170	0.6160	0.6524
CBoW768	0.6136	0.6276	0.6610	0.7043
SG100	0.6240	0.6217	0.6581	0.6461
SG200	0.6211	0.6181	0.6236	0.6513
SG300	0.6260	0.6267	0.6242	0.6684
SG768	0.6208	0.6281	0.6189	0.6826
BagOfWords	0.5738	0.5776	0.5590	0.5933
TFIDF	0.5712	0.5765	0.5521	0.5962
BERT	0.6223	0.6279	0.6230	0.6655
PCA	0.6131	0.6297	0.6304	0.6715
GloVe100	0.6038	0.5968	0.6920	0.5970
GloVe200	0.5923	0.5851	0.6365	0.5845
GloVe300	0.6003	0.5937	0.6265	0.5953
GloVe768	0.6107	0.6084	0.5956	0.6256

Table 5.5: Maximum metric value by classification method

Classification Method	Accuracy	Precision	Sensitivity	Specificity
LogReg	0.6246	0.6252	0.6148	0.6490
LDA	0.6260	0.6267	0.6178	0.6501
QDA	0.6223	0.6297	0.6920	0.7043
MDA	0.6206	0.6160	0.6335	0.6461
FDA	0.6064	0.6052	0.6298	0.6273
Tree	0.5784	0.5745	0.6610	0.6461
RF	0.6240	0.6217	0.6242	0.6634

Chapter 6

Effect of Text Preprocessing Methods

During natural language processes, there are many preprocessing methods researchers use to make text easier to analyze. In this work, we have removed all punctuation, new line spacing syntax, all non-alphanumeric symbols, stop words, numbers, converted to lowercase, and filtering minimum number of appearances. There are various rationales for using these preprocessing methods, but majority of these reasons boils down to reducing the complexity of the text (Silge & Robinson, 2016). However, these preprocessing methods may affect the ability to capture context of words as well as the ability to classify fake news. In this chapter, we will explore the effects of removing stop words and numbers, converting to lowercase, and filtering minimum number of appearances.

6.1 Previous Literature on the Effect of Text Preprocessing

The effect of preprocessing textual data has been briefly explored in areas of text classification similar to fake news. In 2014, the effect of tokenization, stop word removal, lowercase conversion, and stemming were explored on emails (spam/not) and news (10 categories) in both the Turkish and English languages. Tokenization is the process of splitting the text into words or phrases based on a specified dictionary, or in this case, the tokenization process involves alphanumeric characters or just alphabetic characters. Stemming is the process of obtaining the root of a word. For instance, with stemming, the word "talk" and "talking" would be shortened to the root. Thus, "talking" becomes "talk" and are counted as the same word. Tokenization, stop word removal, and stemming algorithms are all dependent on the language of the text (Uysal & Gunal, 2014).

Uysal & Gunal (2014) explore all 16 possible combinations of the text preprocessing methods. Feature selection was completed using the chi-squared method and only the top (most useful) features are maintained. They only utilize a Support Vector Machine classifier, and compare the different combinations based on the micro-F1 score (Uysal & Gunal, 2014). The F1 score considers both precision and sensitivity measures. We can think of the F1 score as an average of precision and sensitivity, which allows us to get a more well-rounded understanding of the model performance by taking these classification metrics into consideration at the same time. The micro-F1 score is a global average of the F1 score (Leung, 2022). Thus, for the news data set, Uysal & Gunal (2014) utilize the true positives, false positives, and false negatives across all 10 classification groups globally. There does not appear to be much difference in the minimum and maximum micro-F1 scores for the email data sets, both in Turkish and English. However, for smaller feature sizes (10 and 20), the score appears much smaller than with larger feature sizes (1000 and 2000) with the news data set in both Turkish and English.

With both the email and news data sets, Uysal & Gunal (2014) report different combinations of preprocessing methods resulting in the minimum and maximum of the micro-F1 score for each feature size. For the feature size that produces the overall maximum score, they did perform a two-tailed paired t-test and find the minimum and maximum scores are significantly different for Turkish emails, Turkish news, English emails, and English news. However, no other formal hypothesis testing was done. The authors conclude different preprocessing methods significantly improve classification methods, and it varies based on language, type of text classification (email or news), and feature size (Uysal & Gunal, 2014). This conclusion is based primarily on empirical evidence with no formal statistics. For instance, there were no formal tests that the feature size makes a difference or for interaction effects of the preprocessing methods. They also only consider a single classification algorithm.

In another study regarding the effect of text preprocessing, the role of n-grams, stop word removal, stemming, top feature selection based on two different weighting schemes, and text normalization were all explored in relation to their effect on classifying reviews (Barushka & Hajek, 2019). The data they used related to hotel reviews. They had a data set containing positive reviews, both legitimate and fake and another containing negative reviews, both legitimate and fake, related to multiple hotels. They compared the effects of the mentioned text preprocessing methods using three classification methods: Naive Bayes (NB), Support Vector Machine (SVM), and a Neural Network (NN). They create a baseline setting, consisting of the top 2000 trigrams found via the TF-IDF weighting scheme, stop word removal, stemming, and document normalization. Document normalization is the process of adjusting the term frequencies such that the length of the document is taken into account. In this review, we will only focus on the accuracy statistic reported by Barushka & Hajek (2019). For positive and negative reviews, the baseline preprocessing methods produced accuracies higher than 80% for all classification methods. All but keeping stop words led to an increase in accuracy of less than 1% (Barushka & Hajek, 2019). Interactions of these preprocessing methods were not explored, and formal testing was not conducted.

6.2 Preprocessing Methods Explored

Stop words are typically removed during natural language processes due to the assumption of their lack of information. In works discussed in Chapter 3, there were mixed results about the importance of stop words. Potthast & Kiesel (2018) found stop words, as part of different stylistic features, may not provide any informative data to detecting fake news, but Horne & Adali (2017) discovered stop words might be informative, especially in the article title. Logic dictates words such as “the” and “as” do not add to the overall meaning of a sentence. However, it is worth exploring how words contained within the 1,149 words in the `tidytext` package in `R` affect the meaning and context of sentences (Silge & Robinson, 2016). For instance, some of the words removed is “help,” “member,” and “problem.” While in general these words may not provide much information, in political statements, it might. Additionally, removing stop words may put words into the same window, which would change how they are treated in Word2Vec, GloVe, and BERT. Filtering of stop words is the first factor we will be exploring to examine its effect on fake news classification.

The second factor we will be exploring is the filtering of numbers. In a recent study, there has been evidence people tend to remember numerical information that fit their views of the world, even if those numbers are false. Additionally, in the same study, as this numerical information is passed person-to-person, the distortion of these numbers becomes worse (Coronel, Poulsen, & Sweitzer, 2019). This holds the potential for dramatic effects in political statements. Therefore, including numbers in our study on the effect of data preprocessing on text seems to be important.

Another preprocessing method commonly used in natural language processes is converting all characters to lowercase. Converting to lowercase makes sense with words at the start of the sentence. For instance, take the first sentence of this paragraph; the first word is “Another.” It does not make sense to treat “Another” and “another” as different words. However, when it comes to proper nouns, this distinction might be significant. As an example, if a possessive, proper noun such as “President Trump’s” is used, based on the fact we are removing punctuation, converting to lowercase would look like “president trumps.” Now, suppose there is another statement using “trumps” in a manner such as “this policy trumps that one.” A proper noun and a verb are going to be treated as the same word in majority of the embedding methods, with the exception of BERT (see Section 2.4). Thus, it is of interest to explore the effect of converting all characters to lowercase on classification and embedding.

Lastly, another common preprocessing method is filtering out words that do not appear a minimum number of times across the collection of documents. In this work, we filter out words at a minimum appearance of 5, 30, and

60 times across all statements. We also include when no words are filtered at all; this would include situations where a word only appears in a single statement in our collection of 17,324 statements. In theory, filtering words out not meeting a specified number of appearances removes uninformative (or misspelled) words. However, we could be filtering out words that would help us distinguish between real and fake news. We explore if we are actually filtering out informative words aiding in classification.

6.3 An Exploratory Analysis

6.3.1 Methods

Following the set-up used in Chapter 5, logistic regression, LDA, QDA, MDA, FDA, classification trees, and random forest are all classification methods explored, and 10-fold cross-validation is used. We compare the effect of classification on multiple embedding methods: PCA, Word2Vec CBoW (100, 200, 300, and 768 dimensions), Word2Vec SG (100, 200, 300, and 768 dimensions), GloVe (100, 200, 300, and 768 dimensions), and BERT. Bag of Words and TF-IDF are ignored in this chapter due to the computational issues related to the sparsity of these matrices. For each combination of classification and embedding method, the following effects are investigated:

- stop word removal (yes/no)
- converting to lowercase (yes/no)
- filtering numbers (yes/no)
- filtering minimum number of appearances (0/5/30/60 words minimum).

We are interested in only some subsets of the full $7 \times 14 \times 2 \times 2 \times 2 \times 4$ factorial. We will be focusing our analysis on the following effects:

- main effects
  - classification method (C)
  - embedding method (E)
  - stop word removal (S)
  - converting to lowercase (L)
  - filtering numbers (N)
  - filtering minimum number of appearances (F)
- two-way interactions
  - $C \times E$
  - $C \times S$
  - $C \times L$
  - $C \times N$
  - $C \times F$
  - $E \times S$
  - $E \times L$
  - $E \times N$
  - $E \times F$
- three-way interactions
  - $C \times E \times S$
  - $C \times E \times L$
  - $C \times E \times N$
  - $C \times E \times F$

We will compare each of these based on their effect on accuracy, precision, sensitivity, and specificity.

It is important to note we lack replication in this study; we only have one observation per $C \times E \times S \times L \times N \times F$ combination. Thus, all other interactions will be contained within the error of the model. In addition, due to the large degrees of freedom for many of these interactions (error degrees of freedom = 2,387), p-values will produce misleading results. The F-ratio will provide more reliable understanding of the effects on our classification metrics and training time. We will be focusing only on F-ratios greater than 2. We will also be relying on visual exploration of the effects, which can be found in Appendix A. In all plots, the black triangle is the median, and the grey circle is the mean. We have used violin plots to explore our data. A violin plot allows us to visualize the distribution of the observations. Since we see the distribution of observations, we get to see how the variability is affected along with the means and medians. This is more insight than a typical mean plot. It is worth to note that for GloVe, Continuous Bag of Words, and Skip-Gram in 768 dimensions, there were computational issues with classification methods QDA, MDA, and FDA when a minimum appearances of 60 was required. This comes from trying to embed into more dimensions than unique words. As a result, these results are exploratory only.

6.3.2 Accuracy

Let us first look at the effect of the classification method, embedding method, and preprocessing methods on the accuracy of our model. Recall accuracy is the probability of correctly classifying real and fake news overall

Table 6.1: Effects of Accuracy

Effect	Degrees of Freedom	F-ratio
C	6	1425.005
E	13	86.977
S	1	1389.398
L	1	0.110
N	1	4.074
F	3	196.127
C x E	78	11.359
C x S	6	4.065
E x S	13	5.061
C x L	6	1.099
E x L	13	1.244
C x N	6	1.497
E x N	13	1.092
C x F	18	8.586
E x F	39	13.043
C x E x S	78	2.459
C x E x L	78	0.790
C x E x N	78	0.580
C x E x F	225	1.688

(see Equation 5.1). Results are in Table 6.1. Among the three-way interactions, classification method, embedding method, and filtering of stop words appears to have some impact with a F-ratio of 2.459. In Figure A.1, we can see for every classification method, almost every embedding method is improved in overall accuracy by not removing stop words. The only combination not improved when stop words remain is a classification tree and PCA. Most of the improvements are small and do not appear practical. Because almost all methods are improved when stop words are kept and the differences are so similar, this indicates visually no interaction between the classification method, embedding method, and filtering of stop words.

Examining Figure A.2, we see including stop words (FilterStopwords = No) does increase the average accuracy rate across all classification methods. In general, however, there does not actually appear to be an interaction, at least visually, despite a F-ratio of 4.065 for the $C \times S$ interaction since keeping stop words improves accuracy across all classification methods. The differences also do not appear to be very large. We also see logistic regression and classification trees are more skewed than other classification methods with their long tails but not significantly as the median and the mean are approximately the same. However, the long tails may be indicative of those methods not doing well (or at least as consistent) in learning the difference between real and fake news.

In Figure A.3, we can see all embedding methods benefit with stop words not being removed. The embedding methods BERT, PCA, and GloVe in all dimensions seem to be aided the most when stop words are maintained in terms of accuracy. Visually, we can only see the slightest of interactions, with the difference in accuracy between keeping and removing stop words for the Word2Vec embedding methods being smaller than the others (F-ratio = 5.061). We see visual evidence of PCA being slightly more skewed when stop words are maintained, indicating potential reliability issues. There is more variability, which a firm statement about PCA and maintaining stop words is difficult. If we combine this with what we saw in the three-way interaction discussed before, this makes sense because PCA and classification trees actually did better when stop words were removed. We can also see many of the distributions of the observations are slightly skewed and have more than a single mode. More than a single mode indicates there is inconsistency in how well methods are classifying real and fake news overall. For a unimodal, skewed distribution, majority of the methods are doing well, but there are a few outliers.

Classification method interacts with filtering threshold (F-ratio = 8.586). In Figure A.4, we see filtering out words appearing less than 30 times is slightly better in terms of accuracy per classification method, except for QDA. For QDA, removing words not appearing a minimum of 5 times improves accuracy slightly, and for random forests, filtering of a minimum of 30 performs the same as with 60. However, there does not appear to be any practical significance between any of the methods in regards to how accurate they are for the number of words filtered out. Classification trees perform the worst regardless of the filtering level. Filtering out words not appearing a minimum of 60 times has quite a few outliers when using logistic regression. We believe this represents the fact when removing words not appearing at least 60 times, we have fewer than 768 words left in the corpus of statements. Thus, embedding into 768 dimensions is actually making dimensionality larger in this case instead of smaller, and logistic regression is more sensitive to this issue than the other methods.

The interaction between embedding method and removing words not appearing a minimum number of times has a F-ratio of 13.043. In Figure A.5, BERT appears to do worse the more words we remove. There does not appear to be much difference between no filtering, filtering a minimum of 5, and filtering a minimum of 30 with BERT. Conversely, GloVe performs better the more words we filter out. In fact, across all dimensions of GloVe and with PCA, there appears to be a significant increase in mean and median accuracy the more words are filtered out. The Word2Vec methods are also increasing in accuracy but it is not as visually noticeable.

There is an effect of classification method by embedding method (F-ratio = 11.359) . There does not appear to be one classification method and embedding method greatly outperforming the other combinations (Figure A.6). One can say for certain not to use classification trees. With PCA in particular, the classification tree method is highly variable has quite of few outliers. Logistic regression with both Word2Vec methods in 768 dimension is performing inconsistently. Majority of the observations are performing similarly to the other classification methods, but there are several outliers affecting the mean.

Lastly, when we look at main effects, classification method, embedding method, filtering stop words, and filtering words out appearing less than a minimum number of times have very high F-ratios at 1425.005, 86.977, 1389.398, and 196.127 respectively. Filtering out numbers appears to be significant with a F-ratio of 4.074. For the classification methods, random forest seems to perform the best, though this method does not appear to be practically different from LDA, logistic regression, MDA, and QDA (Figure A.7). This is still seen in the medians. The methods of FDA and classification tree do not perform as well. As for embedding methods, we do not see many practical differences between the methods in terms of the means. BERT appears has the highest mean accuracy, followed by PCA (Figure A.8). The Word2Vec methods seem to be performing similarly, with SG300 being the best, but not practically different from BERT or PCA. GloVe under performs across all embedding dimensions.

As we saw earlier, maintaining stop words improves accuracy (Figure A.9). However, the difference is 0.0105918, which in the practical sense is not much of an improvement unless consumers are interested in a 1% increase. We can see the distributions of observations between filtering stop words and not are approximately the same, just shifted down when stop words are removed. In terms of filtering words not appearing a specified amount of times, a minimum appearance of 30 seems to perform best with an average of 0.6017579, though this is only 0.0094509 higher than when we filter out no words (Figure A.10). This does not seem very practical. We can see as more words are filtered out, the longer the left tails of the distributions get. This indicates as more words are removed, the more unreliable the classification gets (i.e. the more outliers we observe). Lastly, while the F-ratio seems to indicate there is an effect of filtering out numbers, the difference is only 5.7357702×10^{-4} between maintaining numbers and removing them (Figure A.11). This is not a practical difference. When we are comparing the distributions resulting from filtering numbers (or not), we can see more outliers occur when numbers are removed.

Overall, it appears the interaction of classification method, embedding method, and filtering stop words is the most influential on the accuracy rate. Maintaining stop words does improve the accuracy of the classification and embedding method combinations. Unlike what we saw in the previous chapter, there do appear to be some differences between the embedding methods as a main effect. BERT appears visually to be outperforming the other embedding methods when averaged across all other effects. As for classification methods, FDA and classification trees appear to perform the worst in terms of average accuracy. If a consumer or company is interested in maximizing the probability of correctly classifying real and fake news overall, they should maintain stop words, filter words out not appearing a minimum of 30 times, use

Table 6.2: Effects of Precision

Effect	Degrees of Freedom	F-ratio
C	6	1126.108
E	13	156.542
S	1	835.244
L	1	12.680
N	1	0.020
F	3	17.048
C x E	78	13.202
C x S	6	11.891
E x S	13	6.576
C x L	6	0.887
E x L	13	1.199
C x N	6	2.814
E x N	13	2.089
C x F	18	71.954
E x F	39	15.850
C x E x S	78	2.170
C x E x L	78	0.726
C x E x N	78	1.077
C x E x F	225	3.334

the BERT embedding method, or use a random forest classification method. Out of all $C \times E \times S \times L \times N \times F$ combinations, logistic regression, PCA on the document-term matrix, including stop words, maintaining capitalization, keeping numbers, and filtering words not appearing a minimum of 60 times has a maximum accuracy of 0.6395. However, as we observed, logistic regression and PCA are both highly variable methods.

6.3.3 Precision

Precision is the probability of correctly predicting real news; see Equation 5.2. This value gives an indication of the precision of the predictions. If the probability is high, then more real news is being predicted as real (true positives), whereas if the probability is small, then more fake news is being predicted as real (false positives). The effects on precision can be found in Table 6.2. With the interaction between classification method, embedding method, and filtering words not appearing a minimum number of appearances, there is an effect on precision with a F-ratio of 3.334. QDA noticeably improves precision for all Word2Vec methods when no words are removed from the analysis as seen in Figure A.12. As more words are filtered out, PCA and Random Forest appear to be performing better. GloVe across all classification methods and dimensions seems to improve the more words are filtered out.

Similarly to the accuracy metric, we see there is possibly an effect on precision from the interaction of classification, embedding, and filtering stop words with a F-ratio of 2.17. In Figure A.13, precision is slightly improved when stop words are not removed from the analysis for almost classification methods. PCA and classification trees have improved average precision when stop words are removed. There is no difference between the presence of stop words and no stop words for all Word2Vec methods and random forest. We see more outliers with the Word2Vec methods in 768 dimensions.

For the two-way interactions, filtering of stop words again is interacting with classification method with a F-ratio of 11.891 and affecting precision. We can see in Figure A.14, maintaining stop words improves precision, but it is only slightly. Both the means and the medians appear to visually be equal across classification methods between the levels of filtering stop words. Logistic regression seems to have the most outliers.

Filtering stop words also appears to interact with embedding method (F-ratio = 6.576). All embedding methods, except Continuous Bag of Words in

100 dimensions, appears visually to have significant differences in the medians between removing or keeping stop words (Figure A.15). The largest improvements in precision visually appear with BERT and PCA, where both benefit from keeping stop words.

Unlike with accuracy, there is the slightest evidence of an interaction with classification method and filtering numbers with a F-ratio of 2.814 as well as embedding method and filtering numbers with a F-ratio of 2.089. However, Figures A.16 and A.17 shows the mean and median precision at each classification method and each embedding method is approximately the same for when numbers are maintained versus removed.

Filtering words not appearing a minimum number of times is interacting with classification method and embedding method individually, with F-ratios of 71.954 and 15.85 respectively. The primary driver of the interaction between filtering and classification method appears to be QDA and random forest seen in Figure A.18. For QDA, the best mean (and median) precision comes with no words filtered, whereas with random forest, filtering out words appearing less than 30 times or less than 60 times has the best mean (and median) precision. Filtering words appearing less than 30 times appears to produce the highest mean precision for each classification method other than QDA. The interaction between embedding method and filtering is visually explored in Figure A.19. For BERT, one achieves similar mean (and median) precision for all levels of filtering except with filtering out a minimum number of 60 appearances. GloVe across all dimensions appears to have a decent increase in mean (and median) precision when more filtering occurs. PCA and Word2Vec methods do not appear to have as "large" of differences as with BERT and

GloVe, but the fewer words removed with these methods, the better.

We also see evidence (F-ratio = 13.202) of an interaction between embedding method and classification method. Unsurprisingly, for all embedding methods, classification trees provide worse precision (Figure A.20). For QDA, embedding methods of BERT, Continuous Bag of Words 768, and Skip-Gram 768 produce the largest average precision rate. This is interesting because recall 768 dimensions was picked for Word2Vec methods to see how well they compare to BERT, which has 768 dimensions. However, QDA across the all Word2Vec methods and dimensions has high variability, indicating it is not a reliable method with this embedding method. For all other embedding methods, random forest classification performs the best in terms of precision. Logistic regression also contains multiple outliers with the Word2Vec methods in 768 dimensions, which is not seen with other methods.

Lastly, similar to accuracy, classification method, embedding method, and filtering stop words have large F-ratios (1126.108, 156.542, and 835.244 respectively). With classification method (Figure A.21), random forest is performing the best, but it does not seem practically different from LDA, logistic Regression, MDA, and QDA. Classification trees again perform worst in terms of precision. QDA has high variability, indicating it does not appear to actually be learning the difference between real and fake news. Logistic regression has a fairly symmetric distribution of observations, but it contains much longer tails than the rest of the classification methods due to many extreme outliers. For the embedding methods, PCA actually produces the highest precision, but this embedding method has high variability (Figure A.22). BERT does not seem practically different from PCA and also has large variability. As the number of dimensions increase, the Word2Vec methods begin to have more extreme outliers. GloVe appears to be performing the worse in terms of precision. With filtering stop words, 0.0098283 is the difference in means for when stop words are maintained and when they are removed, which is again probably not practical (Figure A.23).

Filtering words not appearing a minimum number of times and converting to lowercase also appear to have an effect on precision with F-ratios 17.048 and 12.68 respectively. The maximum precision occurs with filtering out words appearing less than 30 times, but this maximum is only 0.0039094 higher than the minimum precision (Figure A.24). When converting to lowercase, the difference in precision is even smaller at 0.0012109, with maintaining capitalization producing the highest precision (Figure A.25). For both of these effects, these differences are so small that they do not appear to make any practical difference to precision.

On the whole, the classification method, embedding method, filtering stop words, and filtering minimum appearances look to be the driving effects on precision. Including stop words seems to improve precision on average for both classification method and embedding method. The number of words to filter out based on the minimum number of appearances greatly depends on the classification method or embedding method used. QDA, Continuous Bag of Words, Skip-Gram, BERT, and PCA perform better respectively when fewer words are removed. As more words are filtered out, Random Forest and GloVe perform better. Out of all $C \times E \times S \times L \times N \times F$ combinations, QDA, Word2Vec Skip-Gram 768, including stop words, maintaining capitalization, keeping numbers, and filtering out no words has a maximum precision

Table 6.3: Effects of Sensitivity

Effect	Degrees of Freedom	F-ratio
C	6	82.573
E	13	23.438
S	1	295.017
L	1	2.701
N	1	7.061
F	3	219.469
C x E	78	10.811
C x S	6	15.512
E x S	13	1.165
C x L	6	0.795
E x L	13	1.345
C x N	6	3.149
E x N	13	1.643
C x F	18	65.520
E x F	39	10.259
C x E x S	78	1.193
C x E x L	78	1.231
C x E x N	78	0.959
C x E x F	225	3.598

of 0.6829. This indicates this specific combination was creating more true positives than false positives. However, Skip-Gram in 768 dimensions and QDA are both respectively highly variable, as well as variable in combination as seen in Figure A.20.

6.3.4 Sensitivity

Found in Table 6.3 are the results exploring the effect on sensitivity. Sensitivity is the probability of correctly identifying real news; see Equation 5.3. If this number is high, we have more true positives, where real news is being accurately classified as real. If it is low, we have more false negatives, where more real news is being classified as fake. The interaction of classification method, embedding method, and removing words not appearing a specified minimum number times looks to be impacting sensitivity (F-ratio = 3.598). In Figure A.26, we notice there is a lot of variability with QDA, with average sensitivity generally improving as the number of words removed increases. The Word2Vec methods in particular do poorly when no words are removed, and QDA is used. PCA and random forest is also not performing well in terms of sensitivity.

The interaction between classification method and filtering of stop words is still creating an impact of sensitivity just as it did with accuracy and precision (F-ratio = 15.512). Figure A.27 shows for majority of the classification methods, the difference in the mean (and median) sensitivity for when stop words are present and when they are removed is about the same is largest for random forest and classification trees. Maintaining stop words appears to help random forest and classification trees predict real news correctly. All other classification methods visually do not show much difference at all between keeping stop words and removing them. Logistic regression has a symmetric distribution but has long tails. QDA also seems pretty skewed given the differences in mean and median. Both of these do not seem as reliable.

Classification method and filtering numbers also appear to interact in terms of sensitivity (F-ratio = 3.149). However, it does not appear there is a difference between the mean (and median) sensitivity rates for when we include numbers versus when we do not for all classification methods, as seen in Figure A.28. The "largest" difference is with QDA, where keeping numbers seems to improve the probability of correctly identifying real news. Logistic regression has a symmetric distribution with long tails for when numbers are removed. When numbers are kept, the distribution for logistic regression is no longer symmetric and has outliers performing worse than the bulk of the observations. QDA again seems pretty skewed both with and without numbers with the differences in means and medians.

With F-ratios 65.52 and 10.2588 respectively, both classification method and embedding method interact individually with filtering out words not appearing a minimum number of times. For the discriminant analyses, there are similar average sensitivities for filtering minimum of 30 and 60 appearances (Figure A.29). For random forest, filtering out fewer words improves mean (and median) sensitivity. Classification trees do not appear to have much difference between the numbers of words removed. We again see quite a bit of variation when using QDA, and this classification method seems to perform much better when words appearing less than 30 or less than 60 times are removed. QDA has high variability when no words are removed. Logistic regression when removing words not appearing a minimum of 60 times also has many extreme outliers. When exploring the effect of embedding method and filtering words (Figure A.30), removing no words from the analysis has the lowest sensitivity rate across the board, except again with BERT. In general, filtering out words that do not appear a minimum of 30 times appears to help each embedding method outside of BERT. When removing no words, the Word2Vec methods do much worse across the board. Multiple combinations appear to have outliers, creating long tails.

The last interaction creating an impact on sensitivity is between classification method and embedding method (F-ratio = 10.811), as we have seen with both accuracy and precision. Figure A.31 shows large variability with

QDA across the Word2Vec methods. Random forest and PCA are producing the lowest average sensitivity rate. The distributions associated with classification trees have high variability. The distribution for GloVe 100/200/300 with QDA is noticeably higher than other combinations.

Finally, we explore the main effects. In terms of sensitivity, all main effects appear to be influential. Classification method has a F-ratio of 82.573, and in Figure A.32, we notice a decrease in sensitivity with classification trees. We see how variable QDA and classification trees are in comparison to other methods, and logistic regression creates many extreme outliers. The median sensitivity for QDA does visually appear to be the highest. In Figure A.33, we gain insight into the embedding methods (F-ratio = 23.438). GloVe in 100 dimensions produces the highest mean (and median) sensitivity, where as Skip-Gram in 768 dimensions is the lowest. In general, fewer dimensions produce higher sensitivity rates.

The filtering of stop words again has a large F-ratio at 295.017. Not removing stop words improves sensitivity by 0.0215402. If we are wanting to detect real news correctly, we are better off not removing stop words. While the F-ratio of 2.701 implies there could possibly be an effect of converting to lowercase, Figure A.35 indicates there is no difference between maintaining capitalization and converting to lowercase. Converting letters to lowercase only increases sensitivity by 0.0020611 compared to maintaining capitalization, which is not practically significant. The F-ratio of 7.061 for removing numbers also indicates a significant effect on sensitivity. However, in Figure A.36, there does not appear to be a difference, and maintaining numbers only increases sensitivity by 0.0033325.

Lastly, filtering out words not appearing a minimum number of times has a F-ratio of 219.469, indicating very strong evidence of an effect on sensitivity. Figure A.37 shows filtering words not appearing a minimum of 30 times performs the best, and a minimum of 60 appearances does not seem that different from 30. Filtering out no words performs much worse than the other levels. The average sensitivity when removing words not appearing 30 times is 0.0420397 higher than when no words are removed. This indicates the more words one filters out, the better the probability of correctly classifying real news. There are, however, many extreme outliers for all levels of minimum number of appearances, which would affect the means. However, the medians seem to follow the same pattern as we see with the means.

In general, we see many of the same patterns we saw with accuracy and precision. Classification method, embedding method, filtering of stop words, and filtering out words not appearing a minimum number of times play a role in how well we are detecting real news correctly. In general, random forest and classification trees do not perform as well as other methods. To improve sensitivity, we want to keep stop words as well as increase the minimum number of appearances a word must have in order to remain in the analysis. GloVe in 100 dimensions seems to produce the highest sensitivity, but much like we saw in Chapter 5, sensitivity is much more variable than accuracy and precision, as seen with many of the extreme outliers in many of the distributions. We really need to consider these results in conjunction with specificity. Out of all $C \times E \times S \times L \times N \times F$ combinations, Logistic Regression, Word2Vec Skip-Gram 768, including stop words, maintaining capitalization, removing numbers, and filtering out words not appearing a minimum of 60 times has a maximum

Table 6.4: Effects on Specificity

Effect	Degrees of Freedom	F-ratio
C	6	136.591
E	13	66.197
S	1	0.015
L	1	3.506
N	1	3.310
F	3	79.932
C x E	78	11.752
C x S	6	16.452
E x S	13	2.347
C x L	6	0.479
E x L	13	1.300
C x N	6	3.670
E x N	13	2.173
C x F	18	76.906
E x F	39	11.126
C x E x S	78	1.280
C x E x L	78	1.155
C x E x N	78	1.197
C x E x F	225	3.492

sensitivity of 0.9087. Thus, we are able to classify real news alone extremely well with this combination. However, the minimum sensitivity value is 0.17719 is for the exact same combination as the maximum except numbers are kept in the analysis. This further supports the large variability we see in sensitivity. We also need to keep in mind both logistic regression and Skip-Gram in 768 dimensions were the combination that produced the most extreme outliers.

6.3.5 Specificity

The effects on specificity are presented in Table 6.4. Specificity is the probability of correctly classifying fake news. If this number is large, then we have more true negatives, where we have correctly found fake news. If the number is small, then we have more false positives, where we have incorrectly identified fake news as real. See Equation 5.4. We see much of the same as we saw before with the three-way interactions with sensitivity. The only interaction seemingly significant is between classification method, embedding method, and removing words not appearing a minimum number of times (F-ratio = 3.492). If we look at Figure A.38, we see QDA has large variability again, especially when no words are removed. Skip-Gram in 300 dimensions with QDA clearly performs the best when no words are removed. However, as more words are removed, specificity is significantly affected for that combination. In general, random forest appears to be performing better as more words are removed, especially with PCA. If we were to put this plot side-by-side with the associated plot with sensitivity, we see an almost inverse relationship. The methods performing the worst in regards to sensitivity are performing the best in terms of specificity. This is what we saw in Chapter 5. This inverse relationship is particularly seen in classification methods of QDA and random forest, but for other classification methods, the number of words removed performing the best across the embedding methods does flip from sensitivity.

Classification method and filtering of stop words again seem to have an effect on specificity (F-ratio = 16.452). In Figure A.39, random forest and classification tree methods are actually aided by removing stop words. There does not appear to be much difference between keeping stop words and removing them for all other classification methods except for FDA. FDA appears to be aided the most by maintaining stop words. This plot does provide indication of an interaction. We do see visual evidence of the inverse relationship with sensitivity, specifically in QDA (Figure A.27). For the other classification methods, the inverse relationship with sensitivity is only seen in the extreme outliers, which seemed to represent unreliable combinations anyways.

Embedding method and filtering of stop words has a F-ratio of 2.347, but Figure A.40 reveals there is not much difference between removing or keeping stop words across all embedding methods. We see multiple outliers with the Word2Vec methods in all dimensions.

Figure A.41 shows the possible interaction between filtering numbers and classification method (F-ratio = 3.67). There does not appear to be any differences in the mean (and median) in specificity between keeping and removing numbers for all classification methods. This graph also visually appears to be a complete mirror of the associated graph with sensitivity (Figure A.28). The primary drivers of this inverse relationship appears to be QDA and logistic regression. When numbers are not filtered, the outliers are now performing better than the bulk of the observations. The interaction between filtering numbers and embedding methods has a F-ratio of 2.173, but Figure A.42 shows there is not much difference between keeping or removing numbers across the different embedding methods.

Filtering threshold is interacting with classification method and embedding method respectively (F-ratios 76.906 and 11.126). We can see, in Figure A.43, QDA again has the most variability, with no words being filtered performing the best for specificity. For random forest, it appears the more words one filters out, the better the predictive performance of fake news. The same is said with classification trees, but the difference is not as large as with random forests. For all other methods, when fewer words are filtered out, the better specificity is attained. When compared to the associated interaction plot with sensitivity (Figure A.29), it is a mirrored image again. In general, as more words are filtered out, the centers of the distributions are decreasing except with random forest. This is opposite of the distributions seen with sensitivity. For the embedding methods, filtering out no words appears to significantly help improve specificity for both Word2Vec methods across all dimensions (Figure A.44). With GloVe, the more words removed, the better specificity is achieved. The Word2Vec methods and PCA have a mirrored relationship compared to what we saw with sensitivity (Figure A.30). With sensitivity, the more words we removed, the better the performance on fake news with Word2Vec and PCA.

The last two-way interaction, classification method and embedding method, is also significant with a F-ratio of 11.752. We again see large variability with QDA across the embedding methods in Figure A.45, except for BERT and GloVe in all dimensions. Random forest performs the best across all embedding methods in general. QDA performs better in terms of the mean (and the median) for certain embedding methods, but due to the variability, this does not seem like a reliable method. This again appears to be the opposite of sensitivity, primarily seen with QDA and the random forest/PCA combination (see Figure A.31).

Classification method (F-ratio = 136.591), embedding method (F-ratio = 66.197), converting to lowercase (F-ratio = 3.506), filtering numbers (F-ratio = 3.31), and filtering threshold (F-ratio = 79.932) are all significant. In Figure A.46, random forest appears to be improving the specificity rate. We also see high variability with both QDA and classification tree as we did with sensitivity. Logistic regression is a symmetric distribution, but it seems to produce outliers. For embedding methods, BERT is performing the best, but it does not appear significantly different from Skip-Gram in 768 dimensions. GloVe in 100 dimensions is noticeably the worst (Figure A.47). In this case, as the number of dimensions increases, so does the probability of correctly classifying fake news, which is the opposite of what we saw with sensitivity. There are again many extreme outliers with the embedding methods and specificity.

Converting to lowercase and filtering out numbers have somewhat high F-ratios, as mentioned. However, in Figures A.48 and A.49, we see the F-ratio is slightly misleading. There does not visually appear to be a difference, let alone a practical difference. Maintaining capital letters only improves specificity by 0.0022164 over converting to lowercase. Removing numbers only improves specificity by 0.0021533 over keeping numbers. For both of these preprocessing methods, the extreme outliers seem to be the driving influence on the inverse relationship with sensitivity. When filtering out words based on number of appearances, filtering out no words seems to offer the most improvement, and it is 0.0180034 above a cutoff of a minimum of 5 appearances (Figure A.50). The trend we see with specificity is again an approximate inverse of what we see with sensitivity, but the primary driver looks to be the extreme outliers.

Overall, specificity seems most influenced by classification method, embedding method, and filtering out words not appearing a specified minimum number of times. QDA appears to be, again, highly variable across the different embedding methods and preprocessing methods. In general, random forest seems to be able to classify fake news correctly the best. For embedding methods, BERT and Skip-Gram in 768 dimensions appears to be the best in terms of specificity. Removing no words appears to also be helping classify fake news correctly. Unlike with other metrics, stop words only appear to be interacting with classification method, where removing stop words aids random forests and classification trees. Out of all $C \times E \times S \times L \times N \times F$ combinations, Logistic Regression, Word2Vec Skip-Gram 768, including stop words, maintaining capitalization, including numbers, and filtering out words not appearing a minimum of 60 times has a maximum specificity of 0.91838. The minimum (0.1975) is again for the same combination except for with removing numbers. These combinations are exactly reversed from what we saw with sensitivity! The maximum sensitivity value occurs for the combination that has the minimum specificity value. This leads us to question if the methods are actually learning the difference between real and fake news or just picking one to place all observations, and as we saw, the extreme outliers were the primary influences on this flipped relationship. These combinations definitely fall as extreme outliers and need to be carefully considered.

6.4 Conclusions

Across all of the metrics calculated, classification method, embedding method, filtering of stop words, and filtering out words not appearing a minimum number of times appear to be the most influential, as well as various interactions involving these four factors. In general, classification trees do not perform well for any of the metrics, and for all but sensitivity, random forest outperforms the other classification methods. This pattern emerges across all interactions involving classification method. There are occasions when QDA outperforms random forest, but QDA does have higher variability. As for the embedding methods, no method really seemed to be outperforming the others for all four metrics. For accuracy and precision, BERT and PCA appeared to be the best. For sensitivity and specificity, GloVe and Skip-Gram appear to be the best methods. Again, this pattern recurs in the interactions involving embedding method.

Filtering out stop words and filtering out words not appearing a minimum number of times are not individually impactful on the four metrics. The differences between levels of the respective individual factors are quite small. However, when each factor interacts with classification method and/or embedding method, we start to see how those preprocessing method affect our metrics. In general, keeping stop words in the analysis improves all four metrics. The only instance maintaining stop words was not beneficial was with random forest and classification trees in terms of the probability of correctly classifying fake news. The effect of filtering out words not appearing a minimum number of times depended quite a bit on which embedding method was used. For BERT in particular, it was aided by keeping more words in the analysis, whereas with GloVe, the more words we filtered out the better.

In a majority of the figures related to specificity, the patterns we see are the inverse of what we saw with sensitivity, similar to what we saw in Chapter 5. This raises the question: are the classification methods actually learning the difference between real and fake news? It looks as if the classification methods are choosing to classify as either all real or all fake and not really learning. However, this does depend on which embedding method and preprocessing methods we use. The embedding method and preprocessing methods are playing some kind of role in whether or not the classification method over predicts real news or over predicts fake news. Embedding methods and preprocessing methods correctly identifying real news do poorly when correctly identifying fake news. One of the possible explanations is the fact the original six labels in the PolitiFact data set have been condensed down to two, which could muddle the classification methods. We will explore this in the next chapter. Additionally, the inverse relationship seems to be primarily influenced by the extreme outliers. More exploration on what these outliers have in common is needed. If the outliers are primarily related to embedding methods, this could indicate the method of averaging across dimensions used to combine word embeddings into statement embeddings is inadequate. Further exploration of methods of embedding whole statements instead of just words are needed and is left for future work.

Lastly, the preprocessing method of stemming was not considered in this work. There are many stemming methods available for consideration, adding in another layer of complexity. Stemming would have some effect on all of the metrics explored here and will also be considered in a future work.

Chapter 7

Extending to Multiple Class Classification

In Chapter 4, we discussed how PolitiFact uses a six label system, where a statement can be given a label of true, mostly true, half true, barely true, false, or pants-on-fire. For the previous chapters, we condensed these six labels into two categories: real and fake. We are now going to extend the classification methods used in the previous chapters from the binary class problem to the multi-class problem. We will briefly explore how the previous classification methods extend to the non-binary class problem as well as how the interpretation of the classification metrics change. We will focus primarily on the preprocessing methods of filtering stop words and the filtering of words not appearing a minimum number of times as these were the effects found in Chapter 6 to have the biggest effects. For all combinations, numbers will be removed, and all words will be converted to lowercase.

Additionally for this chapter, we explore how adding information regarding the year a statement was made affects classification. In Figure 7.1, we can see how the six labels are distributed over our observed time frame from 2007 to 2020. Most notable are the classifications of true, false, and pants-on-fire. Since 2007, the proportion of statements labeled true is decreases, whereas the proportion of labels of false and pants-on-fire gets larger. This indicates including year as a predictor of the class may possibly improve the classification.

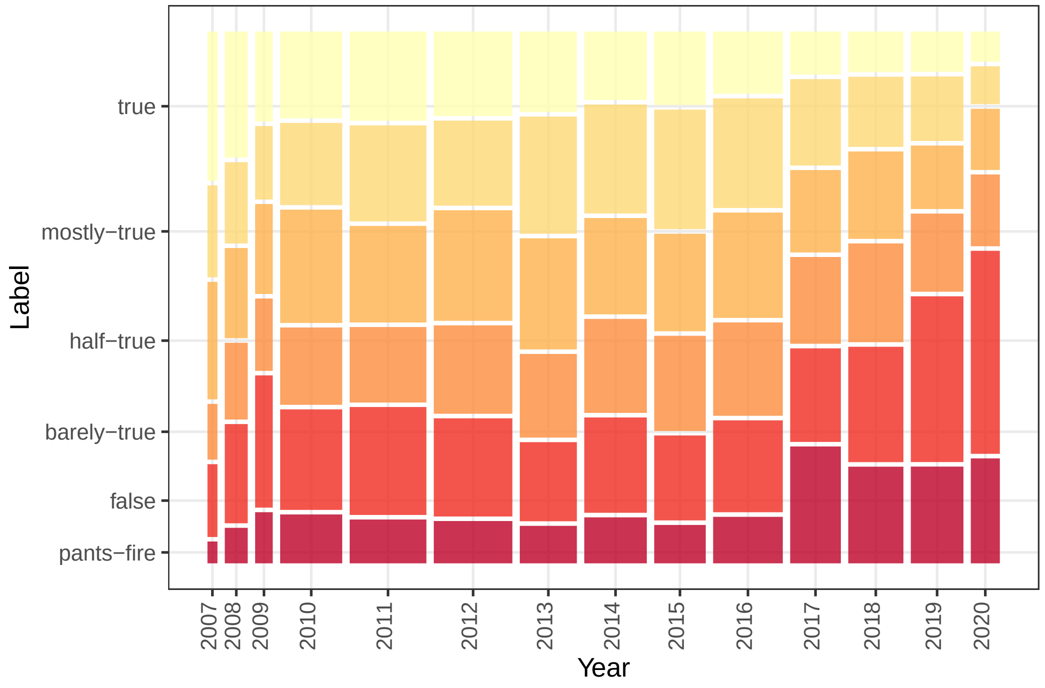

Figure 7.1: The breakdown of each label per year observed in our data set.

7.1 Multi-Class Classification Methods

We extend the classification methods used in Chapters 5 and 6 to a multinomial response $\boldsymbol{Y} = (Y_1, Y_2, \ldots, Y_6)$, where

$$Y_i = \begin{cases} 1 & D \in i^{th} \text{ Label} \\ 0 & \text{Otherwise} \end{cases}$$

The i^{th} label corresponds to true, mostly true, half true, barely true, false, or pants-on-fire respectively. With this case of a multinomial response, we use ordinal regression as the extension of the binary logistic regression. In ordinal regression, the response categories are ordered, and we can incorporate this ordering by utilizing the cumulative probability. We consider not only the probability of an outcome but also the probability of categories before it. There are five different link functions to be utilized with ordinal regression:

$$\text{Logit} = \ln \frac{p_i}{1 - p_i} \tag{7.1}$$

$$\text{Complementary log-log} = \ln(-\ln(1 - p_i)) \tag{7.2}$$

$$\text{Negative log-log} = -\ln(-\ln(p_i)) \tag{7.3}$$

$$\text{Probit} = \Phi^{-1}(p_i) \tag{7.4}$$

$$\text{Inverse Cauchy} = \tan(\pi(p_i - 0.5)) \tag{7.5}$$

In all of these, $p_i = P(Y \leq i)$, where $P(Y \leq i)$ is the probability of being in class i or below. Equations 7.1 and 7.4 are useful with evenly distributed categories or gradual changes in p_i. Equation 7.2 is useful when higher categories are more probable, whereas Equation 7.3 is better when lower categories have higher probabilities. Lastly, Equation 7.5 is useful when there are extreme values (Hua, Choi, & Shi, 2021). In `R` with the `polr` function and the `caret` package, determining which of these link functions to use is a tuning

Table 7.1: Multiple Classes Confusion Matrix

	Actual					
	True	Mostly True	Half True	Barely True	False	Pants-on-Fire
Predicted						
True	TN	TN	TN	TN	FN	TN
Mostly True	TN	TN	TN	TN	FN	TN
Half True	TN	TN	TN	TN	FN	TN
Barely True	TN	TN	TN	TN	FN	TN
False	FP	FP	FP	FP	TP	FP
Pants-on-Fire	TN	TN	TN	TN	FN	TN

parameter; the link function with the highest training accuracy is used. All other classification methods used extend naturally to the multiple classes.

7.2 Applying the PolitiFact Dataset

Extending to multiple classes extends the confusion table, as seen in Table 7.1. In this table, the columns indicate the actual label assigned by the PolitiFact team, and the rows indicate the label predicted by the classification method. Let us assume we have a statement given a rating of "false" by PolitiFact. If one recalls from Chapter 4, a rating of false is given to a statement that is inaccurate, whereas a statement given a rating of pants-on-fire is inaccurate plus makes a ridiculous claim. These two categories, while both indicate fake news, are different severities of false claims. If the classification method correctly identifies this statement as false, this is a *true positive* (TP). If the false statement is incorrectly predicted as one of the other five classes, this is a *false negative* (FN). If a statement given any of the other ratings by the PolitiFact team is predicted as false, this is a *false positive* (FP). Any other predication not involving a label of false, either actual or predicted, is considered a *true negative* (TN).

From this confusion matrix, we can calculate the overall accuracy of the classification method similar to Equation 5.1 in the binary case. Assume the confusion matrix for multiple classes is called C. The overall accuracy for a classification method can be found as follows:

$$\text{Accuracy} = \frac{\text{tr}C}{\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}} \tag{7.6}$$

where n is the number of classes and c_{ij} is an element of C. Unlike with binary classification, we can no longer look at sensitivity (true positive rate) and specificity (true negative rate) for the overall method; we explore these metrics *per class*. The calculations remain the same as we saw in Equations 5.3 and 5.4, but we will now have these values for all six classes. For an example, let us stick with the illustration used in Table 7.1. Sensitivity would be the probability of correctly classifying a statement given the rating of false. Thus, with a high sensitivity value, the method correctly identifies false news from the other five classes. Specificity would be the probability of correctly identifying the statement as **not** false. However, with the specificity value, we are not given any indication of how well the other groups are being classified. If specificity is large, the method correctly identifies the statement is something other than false, but we still do not know if this particular statement was predicted to the correct class. For this chapter, we will not explore the effect on precision due to the fact some classification methods produce 0 true positives and 0 false positives across multiple groups, leading to division by 0. In other words, some methods do not make predictions in all categories (i.e. a row of the confusion matrix sums to 0).

As previously noted in Chapter 4, the six labels in the PolitiFact data

Table 7.2: Breakdown of the Six Labels in the PolitiFact Data Set

Label	Percent
True	0.138
Mostly True	0.179
Half True	0.189
Barely True	0.167
False	0.211
Pants-on-Fire	0.116

set are not equally distributed. This breakdown can be found in Table 7.2. We determined that while the labels are not balanced among the groups, the imbalance is not severe. We proceed with splitting the data set into training and test sets without considering sampling strategies to address the imbalance.

7.3 Exploratory Analysis

7.3.1 Methods

We follow the set-up of the exploratory analysis used in Chapter 6. However, we no longer include the effects of converting to lowercase and removing numbers. For all combinations, all letters are converted to lowercase, and all numbers are removed. We only focus on the effect of filtering stop words and removing words not appearing a minimum number of times. We also include a new factor, whether or not year was included as a predictor. We will focus on the following effects:

- main effects
    - classification method (C)
    - embedding method (E)

  - stop word removal (S)
  - filtering minimum number of appearances (F)
  - year included (R)

- two-way interactions
  - $C \times E$
  - $C \times S$
  - $C \times F$
  - $C \times R$
  - $E \times S$
  - $E \times F$
  - $E \times R$

- three-way interactions
  - $C \times E \times S$
  - $C \times E \times F$
  - $C \times E \times R$

We will compare each of these based on their effect on overall accuracy. We will also compare based on their effect of sensitivity and specificity for the two extreme classes: true and pants-on-fire.

It is important to note we still lack replication in this study; we only have one observation per $C \times E \times S \times F \times R$ combination. As before, all other interactions will be contained within the error of the model, where there are currently 934 error degrees of freedom. P-values will again produce misleading results, so we will focus only on the F-ratios greater than 2. The visual explorations of these effects are found in Appendix B. We again use violin plots,

Table 7.3: Effects on Overall Accuracy

Effect	Degrees of Freedom	F-ratio
C	6	1369.341
E	13	31.290
S	1	249.807
F	3	29.811
R	1	325.709
C x E	78	11.719
C x S	6	8.694
E x S	13	0.895
C x F	18	5.213
E x F	39	2.805
C x R	6	36.490
E x R	13	1.152
C x E x S	78	2.401
C x E x F	221	1.502
C x E x R	78	1.064

and we denote the median with a black triangle and mean with a grey circle. Kindly note that for the Word2Vec methods and GloVe, when embedding is done in 768 dimensions for a filtering of 60 minimum appearances, there are several classification methods which result in errors of missing metrics for the analysis: ordinal regression, QDA, MDA, and FDA. This is a result of embedding into a dimension larger than the number of unique words in the data set, resulting in sparse embeddings. The same issue occurs with PCA with the classification methods of ordinal regression and QDA when the level of filtering out words not appearing a minimum number of times is 0 or 5. These results are exploratory only.

7.3.2 Overall Accuracy

We begin by exploring the effects on the overall accuracy of the model. Based on the label distribution, if a method was randomly selecting a class to predict all labels, then the classification method would achieve at most a 21% accuracy rate. Thus, the graphs depicting the effect on the overall accuracy display a dashed line at 0.21, indicating this best rate possible if randomly guessing.

The only three-way interaction with a F-ratio greater than 2 is between classification method, embedding method, and the filtering of stop words, seen in Table 7.3 (F-ratio = 2.401). In Figure B.1, we can see that for LDA and QDA, not removing stop words improves accuracy for all embedding methods. For the other classification methods, there is no consistency on the embedding method where filtering stop words does improve overall accuracy. For instance, with MDA, GloVe in 768 dimensions is improved when stop words are removed, but that is not seen elsewhere with MDA. In particular, multiple classification methods with PCA do better when stop words are filtered. Also with PCA, all discriminant analyses, except with FDA, are highly variable. In general, there are not many practical differences between when stop words are kept and when they are removed.

We now explore the two-way interactions. The interaction between classification method and year is significant (F-ratio = 36.49). Looking at Figure B.2, we see the discriminant analyses are greatly aided by including year as a predictor. Ordinal regression and random forest are also aided by including year, but the difference is not as large. The tree based methods do not appear to have any difference in overall accuracy when year is included. We see ordinal regression and classification trees are the only methods with observations appearing below our “best” accuracy of 21%, but the bulk of the observations are above the random chance line.

Filtering threshold and classification method are also interacting (F-ratio = 5.213). For QDA, removing no words produces a better overall accuracy, which is the opposite of what was seen with the binary case where removing no words did the worst (Figure B.3). For ordinal regression and classification trees, there does not appear to be much difference between the levels of minimum appearances. Removing words not appearing at least 5 times looks to be performing the best for all other methods, but not significantly.

There is moderate evidence of a possible interaction between the filtering threshold and embedding method (F-ratio = 2.805). In Figure B.4, the more words kept in the analysis, the better overall accuracy BERT provides. For the Word2Vec methods, removing words not appearing a minimum of 5 times greatly improves overall accuracy, except with Skip-Gram in 300 and 768 dimensions. We also see more skewed distributions; there are noticeable differences between the means and medians for multiple combinations. This is an indication of the methods struggling to differentiate between the six classes.

We have evidence of an interaction between classification method and the filtering of stop words (F-ratio = 8.694; Figure B.5). The associated graphs reveal not removing stop words improves overall accuracy across all classification methods, as we saw in the binary case. The difference between keeping stop words and removing them is not as large for the random forest method as it is with the other methods. The largest improvements in overall accuracy with keeping stop words occurred with LDA, QDA, and MDA.

Classification method and embedding method also interact (F-ratio = 11.719). Figure B.6 reveals ordinal regression and classification trees perform the worst across all embedding methods. Random forest produces the highest average overall accuracy for almost all embedding methods. LDA works best with PCA and Skip-Gram in 100 dimensions. GloVe in 768 dimensions works best with QDA. We see there is a lot of variability with many of the classification method and embedding method combinations, seen with the very long and flat distributions. We also see many skewed distributions with noticeable differences between the mean and the median. As before, this implies multiple classification methods are struggling distinguishing between the six classes.

All main effects have high F-ratios. Overall accuracy improved when year was included. However, the difference in the mean overall accuracy between including year and not including it is only 0.0056553 (F-ratio = 325.709; Figure B.7). There seems to be a more significant difference between the medians, where including year improves overall accuracy. Filtering out words not appearing a minimum of 5 times improves the accuracy the most, but there does not visually appear to be a difference between 0 and 30 minimum appearances (F-ratio = 29.811; Figure B.8). When stop words are kept, the overall accuracy is again improved but only by 0.0050824 as seen in Figure B.9 (F-ratio = 249.807). The medians seem to show more of a significant difference between keeping stop words and removing them.

For the main effect of embedding method (F-ratio = 31.29), the plot in Figure B.10 reveals the Word2Vec methods improve accuracy more than the other embedding methods. They do not appear significantly different from BERT, GloVe in 100 dimensions, and GloVe in 300 dimensions however. In addition, there does not appear to be any practical differences. The embedding methods with the largest average overall accuracy (Skip-Gram in 100 and 200 dimensions) is only 0.0069354 larger than the smallest (GloVe in 768 dimensions). However, we do see significant differences between the mean and median for many of the embedding methods. Thus, the medians are a better representation. BERT has the highest median overall accuracy, followed by Word2Vec Continuous Bag of Words in 300 dimensions.

Lastly, classification method is significant with a F-ratio of 1369.341. We can see ordinal regression and classification trees clearly perform the worst in terms of mean overall accuracy, whereas random forest performs the best (Figure B.11). Practically, random forest performs similarly to LDA with a difference of only 0.0036214. However, random forest outperforms classification trees by 0.0368441, indicating the choice of classification method does matter. None of the classification methods are doing significantly better than random chance despite the improvement seen with random forest.

As a whole, we do not see as drastic of an effect of the embedding method and filtering words not appearing a minimum number of times as we did in the binary case. We do continue to see not removing stop words improves the overall accuracy. We also see evidence that the inclusion of the year a statement was made improves accuracy, especially for the discriminant analyses. Out of all $C \times E \times S \times F \times R$ combinations, random forest, Continuous Bag of Words in 768 dimensions, removing stop words, filtering a minimum of 5 appearances, and including year produces the highest overall accuracy with 0.2834.

Table 7.4: Effects on Sensitivity of True Label

Effect	Degrees of Freedom	F-ratio
C	6	1443.107
E	13	10.451
S	1	0.753
F	3	9.608
R	1	1.800
C x E	78	7.095
C x S	6	7.727
E x S	13	1.897
C x F	18	5.371
E x F	39	0.816
C x R	6	49.388
E x R	13	1.213
C x E x S	78	1.427
C x E x F	221	0.800
C x E x R	78	1.313

7.3.3 Sensitivity - True Label

We now shift our focus to the class of true statements. Namely, we will explore the effect on sensitivity, the probability of correctly classifying a true statement as true. Only about 14% of all classes in the PolitiFact data set are true. Therefore, if a classification method is merely guessing, the best it can do with the true class specifically is 14%.

None of the three-way interactions have a F-ratio greater than 2 (Table 7.4), so we begin with the two-way interactions. We start with the interaction between classification method and the inclusion of the year a statement was made (F-ratio = 49.388). As with the overall accuracy, the inclusion of year improves the correct classification of true news with the use of discriminant analyses, seen in Figure B.12. Ordinal regression is greatly aided with year not included as a predictor. One will notice classification trees have a sensitivity rate of 0 for both with and without year. Classification trees consistently classify all statements to the middle classes. We see FDA is consistently under performing when compared to random guessing. We also see ordinal regression is the most variable compared to the other classification methods.

Classification method and the filtering threshold are interacting (F-ratio = 5.371). Random forest and QDA do better at classifying true news the more words are filtered out, seen in Figure B.13. Removing words not appearing at least 60 times does worse for ordinal regression, but there does not appear to be differences with the other levels of filtering. There appears to be no differences between the levels of filtering thresholds for LDA and MDA.

With a F-ratio of 7.727, classification method and filtering of stop words interact. Ordinal regression slightly benefit from keeping stop words (Figure B.14). QDA and random forest are improved with the removal of stop words. There does not appear to be a difference between keeping and removing stop words for LDA, MDA, and FDA.

We also find classification method and embedding method interact (F-ratio = 7.095). On average, we find the discriminant analyses, other than FDA, out perform random guessing across all embedding methods (Figure B.15). BERT and MDA seem to perform best when evaluated on the detecting of true statements correctly. We have already discussed classification trees never classifying any statement as true, but FDA does not do much better on average. Random forest also performs worse than random guessing on average. Ordinal regression has large variability regardless of the classification method, except with Skip-Gram in 200 dimensions. We also see noticeable differences between the means and medians, particularly with ordinal regression.

Filtering words not appearing a minimum number of times (F-ratio = 9.608), embedding method (F-ratio = 10.451), and classification method (F-ratio = 1443.107) are all significant. A minimum of 30 appearances has the highest average sensitivity rate, but it only appears to be significantly different than removing no words (Figure B.16). This difference is only 0.0106534. The differences are more apparent in the medians.

With the embedding methods, Figure B.17 reveals high variability with all embedding methods. Continuous Bag of Words in 768 dimensions performs the best, but it does not appear to be significantly different from CBoW in 100, 200, and 300 dimensions, GloVe in 300 dimensions, and Skip-Gram in 300 and 768 dimensions. GloVe in 100 dimension performs the worst and has an average 0.0150202 smaller than CBoW in 768 dimensions. Similar trends are seen with the medians.

Lastly, Figure B.18 reveals the effect of classification method on the probability of classifying true statements correctly. Other than FDA, the discriminant analyses greatly out perform the other classification methods. There do not appear to be any significant differences among LDA, QDA, and MDA, with QDA having the lowest mean sensitivity rate at 0.1742734. The next highest average sensitivity rate is with ordinal regression and is 0.0450147 lower than QDA. Ordinal regression has the most variability out of all classification methods.

Overall, if a person is primarily interested in detecting true news correctly, using LDA, QDA, or MDA are the better options. They still do not perform well, but they do at least a bit better than random guessing on average. With all discriminant analyses, including year improves sensitivity. Using any min-

Table 7.5: Effects on Specificity of True Label

Effect	Degrees of Freedom	F-ratio
C	6	1507.220
E	13	15.516
S	1	4.427
F	3	17.417
R	1	2.606
C x E	78	9.182
C x S	6	4.565
E x S	13	1.221
C x F	18	8.112
E x F	39	0.868
C x R	6	47.170
E x R	13	1.549
C x E x S	78	1.333
C x E x F	221	0.771
C x E x R	78	1.497

imum appearances cutoff other than 60 and including stop words improves sensitivity for LDA and MDA. However, for QDA, not including year and filtering words not appearing at least 30 times improves sensitivity. The highest sensitivity occurs with the combination of ordinal regression, PCA, not removing stop words, not including year, and filtering words not appearing a minimum of 5 times at a value of 0.30205. However, as noted, ordinal regression has the most variability among all the classification methods.

7.3.4 Specificity - True Label

We now explore the effect on specificity relative to the true class of statements. Results are located in Table 7.5. Specificity measures the probability of classifying a not true statement as not true. In other words, the probability of correctly identifying a statement as not true, but this does not mean the statement was correctly identified. It was only correctly identified as not true. If this number is large, then the method is correctly identifying statements as not true. If this number is small, then the method is incorrectly classifying statements as true.

Starting again with the two-way interactions, classification method and the inclusion of year are interacting with a F-ratio of 47.17. In Figure B.19, there is not much of a difference between including year and not for LDA, QDA, MDA, and random forest. FDA is improved when year is not included, whereas ordinal regression is improved when it is included. Ordinal regression has more variability both with and without year as a predictor when compared to the other methods, which was expected based on what we saw with sensitivity of a true class. Again, as we noted before, classification trees never classify any statement as true. Thus, this method appears to do well at correctly identifying as not true, but in combination with what we saw with how classification trees correctly identified true statements, this method is actually performing extremely poorly.

Classification method and removing words not appearing a minimum number of times interact as well (F-ratio = 8.112). The biggest differences appear to be occurring within QDA and random forest classification methods, where the fewer words one removes, the better (Figure B.20). The other methods show very little, if any, differences between the levels of filtering. We again see large variability with the ordinal regression method.

We have moderate evidence of an interaction between classification method and filtering of stop words (F-ratio = 4.565). However, after examining Figure B.21 there does not appear to be much of a difference between keeping and removing stop words for the classification methods. Ordinal regression is the only classification method showing a possible significant difference in the medians.

In Figure B.22, we see the interaction effect of classification method and embedding method on the probability of correctly identifying a statement as not true is the reverse of what we saw with sensitivity (F-ratio = 9.182). The discriminant analyses, except for FDA, are performing the worst across almost all embedding methods. We believe this is an indication the discriminant analyses are actually attempting to learn the difference between the classes and not merely picking a subset of the classes to classify all statements. Ordinal regression shows quite a bit of variability, as expected based on previous results. This variability is not showing up, however, for ordinal regression and Skip-Gram 200 dimensions.

The main effect of including year as a predictor appears significant with a F-ratio of 2.606. However, Figure B.23 reveals this F-ratio is misleading. In addition, the difference in the mean specificity between including year and not including year is only 0.0014752.

With the filtering threshold, the F-ratio for the main effect is 17.417, but again, Figure B.24 reveals this is might be misleading with the mean specificity. The highest specificity occurs when no words are removed, and this is only slightly different when filtering words not appearing at least 30 or 60 times. Not removing any words has an average specificity that is only 0.0063299 larger than the average specificity when a minimum of 30 appearances is used. There is visual evidence of significant differences in the medians however. Filtering no words or words appearing less than 5 times significantly outperform a threshold of 60 in terms of the medians.

We also have evidence of a main effect of filtering stop words (F-ratio = 4.427). However, Figure B.25 reveals this is misleading in terms of the means. The difference between the mean specificity when stop words are not removed is only 0.0021737 bigger than when stop words are filtered. Visually, there does appear to be a significant difference between the medians, but the practicality of this difference does not seem relevant.

Embedding method is also significant (F-ratio = 15.516). The mean specificity in Figure B.26 is almost a mirror image of what we saw with sensitivity of the true class, where Skip-Gram in 100 dimensions is performing the best and Continuous Bag of Words in 768 dimensions is performing the worst. However, the distributions are skewed, so the median may be a better representation of the data. With the median, Skip-Gram in 100 dimensions and BERT appear to be performing similarly, which is not seen with the means.

Figure B.27 reveals the main effect of classification method (F-ratio = 1507.22). LDA, QDA, and MDA appear to be classifying not true news correctly with a lower probability than the other methods. However, in combination with the results from sensitivity relative to the true label, it appears these methods are actually finding reasonable discriminant rules between the groups instead of just guessing.

Overall, if a person is wanting to make sure a method does not misclassify statements as true, the use of the FDA classification method is suggested. With FDA, using GloVe in 768 dimensions, including stop words, filtering out no words, and not including year improves the average specificity in individual combinations. Clearly, among all $C \times E \times S \times F \times R$, any combination using

Table 7.6: Effects on Sensitivity of Pants-on-Fire Label

Effect	Degrees of Freedom	F-ratio
C	6	2080.956
E	13	9.524
S	1	19.809
F	3	9.366
R	1	355.736
C x E	78	32.143
C x S	6	20.903
E x S	13	1.719
C x F	18	16.803
E x F	39	1.909
C x R	6	44.062
E x R	13	4.217
C x E x S	78	0.982
C x E x F	221	2.840
C x E x R	78	2.353

a classification tree is going to be the best since specificity is always 1 relative to the true class, though this is not the method we recommend to use.

7.3.5 Sensitivity - Pants-on-Fire Label

With the pant-on-fire class, we explore the effect on sensitivity (Table 7.6), the probability of correctly identifying this specific class. Only 11.6% of all statements received a rating of pants-on-fire. In this case, there are two significant three-way interactions. The first is between classification method, embedding method, and the inclusion of year with a F-ratio of 2.353. Figure B.28 reveals that for each classification method, including year improves the probability of classifying pants-on-fire statements correctly. There does not appear to be any differences within random forest and FDA across all embedding dimensions. As with the classification of true labels, classification trees are not classifying any statement into the pants-on-fire category with FDA having nearly the same result.

We have moderate evidence the filtering threshold, classification method, and embedding method interact (F-ratio = 2.84). In Figure B.29, FDA and random forest do not perform much better than classification trees. While these methods do sometimes classify pants-on-fire statements correctly, it is not often. LDA and MDA perform the most consistently above random guessing. For most classification and embedding method combinations, there does not appear to be any differences between the levels of filtering. The exception is QDA with any embedding method of dimensions 100, 200, or 300 (except GloVe in 300) where filtering out no words does better.

The first two-way interaction we will explore is embedding method and the inclusion of year as a predictor (F-ratio = 4.217). Observing from the means in Figure B.30, sensitivity is improved for all embedding methods when year is included as a predictor. The smallest difference in means occurs with Skip-Gram in 768 dimensions. When looking at the medians, we still see an increase in sensitivity for almost all embedding methods when year is included. The medians show Skip-Gram in 768 is actually slightly improved in detecting pants-on-fire statements when year is not included. BERT and PCA are the only embedding methods with median sensitivity below random chance for both the inclusion and exclusion of year. Because this is different from what we see in the means, this indicates skewness in the distributions, and analyzing the medians may be more appropriate.

Classification method and the inclusion of year also interact (F-ratio = 44.062). Figure B.31 reveals for all classification methods, including year improves the probability of detecting a pants-on-fire statement correctly, though not significantly for FDA and random forest. We see high variability within the QDA and ordinal regression classification methods, both when year is included as a predictor and when it is not. Additionally, outside one outlier with FDA and including year, both FDA and random forest perform consistently below random guessing on average.

Filtering words not appearing a minimum number of times and classification method interact as well (F-ratio = 16.803). QDA shows the most differences in the levels of minimum appearances (Figure B.32), where removing no words better classify pants-on-fire statements. For random forest, removing words not appearing at least 60 times performs best. There is no difference between minimum appearances with the LDA, MDA, and FDA classification method. With QDA, most of the variability is seen when no words are removed and when words not appearing a minimum of 5 times are removed. The other levels with QDA still have high variability, but it is not as extreme. As before, FDA and random forest consistently perform below random chance, except for an outlier with FDA and no words removed.

For the interaction of classification method and filtering of stop words (F-ratio = 20.903), there are no differences in the inclusion or removal of stop words with any classification method, except ordinal regression seen in Figure B.33. Ordinal regression has improved sensitivity relative to the pants-on-fire label when we keep stop words, but it has high variability. Again, random forest and FDA perform worse than random chance, except for the outlier with FDA and not filtering stop words.

With a F-ratio of 32.143, classification method and embedding method interact. In Figure B.34, we see discriminant analyses consistently perform better than random chance, outside of FDA and the QDA/BERT combination. Random forest and FDA consistently under perform compared to random chance. Ordinal regression varies between being better and worse than random chance, but we also see that this method is essentially a line for almost all embedding methods because of the variability. Ordinal regression does not have the same variability with the Skip-Gram embedding method. QDA primarily has high variability with PCA and Skip-Gram in 100 dimensions. Interestingly, PCA and QDA have two very distinct modes with no observations in between.

All main effects appear to be significant. The inclusion of year (F-ratio = 355.736) is greatly improved when year is included as a predictor. Without year, the average sensitivity is below random guessing but is higher with year (Figure B.35). In fact, including year is 0.0288157 larger than not including it. We see the same improvement with the medians.

The main effect of filtering words not appearing a minimum number of times has a F-ratio of 9.366. In Figure B.36, we see a minimum appearance of at least 30 is the only level with a mean and median slightly above random chance. The mean does not appear to be significantly different from removing no words. The average sensitivity at a minimum of 30 appearances is only 0.0108069 higher than minimum appearances of at least 5 times, however. There seems to be more significant differences, at least visually, between the medians. In terms of the distributions, we see the most variability in the probability of detecting pants-on-fire statements correctly when no words are removed.

Including stop words is significantly different from not including stop words (F-ratio = 19.809). When stop words are not removed, we see the average sensitivity improve by 0.0061235, which is not much of a practical improvement (Figure B.37). The medians show more of a significant difference but still not a practical significant difference.

With the embedding methods (F-ratio = 9.524), the violin plots reveal very skewed distributions (Figure B.38), and the interpretation of the results is remarkably different between the means and the medians. Because of this, we will focus on the medians. Only Continuous Bag of Words in 300 and 768 dimensions as well as Skip-Gram in dimensions in 200, 300, and 768 dimensions have medians above random chance. They seem to be performing significantly better than Continuous Bag of Words in 100 dimensions, Skip-Gram in 100 dimensions, GloVe in 100 dimensions, and BERT. For classification method (F-ratio = 2080.956), LDA, QDA, MDA, and ordinal regression are all performing better than random chance, as seen in Figure B.39. This is seen in both the means and the medians. However, QDA and ordinal regression appear to be performing very inconsistently due to the high variability.

Overall, if a person is mostly interested in classifying pants-on-fire statements correctly, using QDA performs the best on average. However, due to the high variability, it does not seem very reliable. The next method would be MDA. With the true statements, we did not see the variability with QDA, but the discriminant analyses were also performing the best. If they are not concerned about the variability seen with QDA, they could use Skip-Gram in 100 dimensions, filtering no words, and QDA for best results overall. They could also include year or filter stop words in combination with QDA. If a

Table 7.7: Effects on Specificity of Pants-on-Fire Label

Effect	Degrees of Freedom	F-ratio
C	6	1695.592
E	13	8.300
S	1	9.214
F	3	1.590
R	1	122.450
C x E	78	35.297
C x S	6	32.135
E x S	13	1.208
C x F	18	17.852
E x F	39	2.828
C x R	6	27.138
E x R	13	2.166
C x E x S	78	0.913
C x E x F	221	3.754
C x E x R	78	2.029

person does want something a bit more reliable, BERT and filtering no words with MDA is the best combination. Including year or not removing stop words also improves average sensitivity with the pants-on-fire class in combination with MDA. From all combinations of $C \times E \times S \times F \times R$, QDA, Skip-Gram in 100 dimensions, removing stop words, not including year as a predictor, and a minimum of appearances of 0 with a sensitivity of 0.64375. This is fairly high, so the combination seems very promising. However, as we have seen, QDA is considerably variable and not the most reliable of methods.

7.3.6 Specificity - Pants-on-Fire Label

Lastly, we explore the effect on the specificity relative to the pants-on-fire class. As a reminder, this is the probability of correctly identifying a statement as not part of the pants-on-fire class. Between classification method, embedding method, and the inclusion of year as a predictor, we have evidence of an interaction with a F-ratio of 2.029. In Figure B.40, we see the only classification method with the largest differences in specificity between including year or not is for ordinal regression with Continuous Bag of Words in 100/200/768 dimensions, BERT, and PCA. LDA and MDA as show slight improvement when year is not included as well for multiple embedding methods. Most combinations show little to no difference between the use of year as a predictor in relation to the specificity of pants-on-fire labels. We also note classification trees never classify any statement as pants-on-fire, leading to every combination having a probability of correctly identifying a statement as not pants-on-fire of 1.

The three-way interaction between filtering words not appearing a minimum number of times, classification method, and embedding method is significant (F-ratio = 3.754). We see a lot of variability in the embedding method and minimum appearances interaction by classification method for QDA and ordinal regression (Figure B.41). The graph appears to be a mirror image to what we saw with sensitivity relative to the pants-on-fire class. The combinations that did well in sensitivity are doing the worst in specificity. However, "worst" is still doing pretty well, usually above an average specificity of 0.8. The exceptions are with Continuous Bag of Words in 100 dimensions and Skip-Gram in 100, 200, and 300 dimensions, removing no words, and QDA. In addition, removing words not appearing a minimum of 5 times, Skip-Gram in 100 dimensions, and QDA is below 0.8 as is BERT, ordinal regression, and a minimum of 60 appearances.

With a F-ratio of 2.166, we have evidence the interaction of including year as a predictor and the embedding method are interacting. In Figure B.42, we see that across all embedding methods, not including year improves specificity relative to the pants-on-fire label. The largest increases occurs with BERT, GloVe in 100 dimensions, and Continuous Bag of Words in both 100 and 200 dimensions. The smallest increase happens with Skip-Gram in 768 dimensions. These differences are seen in both the means and the medians.

Classification method and the inclusion of year as a predictor is also affecting specificity (F-ratio = 27.138). Figure B.43 shows all classification methods have visually no difference between the including and excluding year. Large variability is found with QDA and ordinal regression similar to the results with sensitivity relative to the pants-on-fire label.

Embedding method and filtering threshold interact (F-ratio = 2.828). Figure B.44 shows there is really no consistent level of minimum appearances that increases the mean specificity across all embedding methods. Large differences exist with GloVe in 768 dimensions, Continuous Bag of Words in 768 dimensions, and Skip-Gram in 100 and 768 dimensions. For GloVe, Continuous Bag of Words, and Skip-Gram methods, all in 768 dimensions, a minimum of 30 appearances performs vastly lower than the other levels. Removing no words performs the worst for Skip-Gram 100, but this is not seen with medians; removing no words actually does best for Skip-Gram 100 based on the medians. With Skip-Gram 768, the best level is filtering out words not appearing a minimum of 60 times, which is also the best average specificity combination overall. This trend is partially seen in the medians. In terms of the medians, we also see a minimum of 30 appearances is much better than other levels for GloVe in 300. We see high variability across all minimum appearances for

BERT. We also see high variability with minimum appearances of 30 for PCA and GloVe in 768 dimensions. With the Skip-Gram and Continuous Bag of Words embedding methods, we see variability when no words are removed and in small dimensions.

The same preprocessing method also interacts with classification method (F-ratio = 17.852). Across the classification methods, only QDA appears to have differences, at least visually, between the levels of minimum appearances (Figure B.45). Not filtering any words performs the worst for QDA. We see the same variability among observations from the QDA and ordinal regression classification methods as we did with sensitivity relative to the pants-on-fire label.

Classification method and the preprocessing method of filtering stop words interact in their effect on specificity (F-ratio = 32.135). Only for ordinal regression does removing stop words improve the mean (and median) specificity (Figure B.46). Not removing stop words only slightly aids QDA in improving specificity. All other classification methods do not show visually a difference in correctly identifying statements as not pants-on-fire.

The last significant two-way interaction is classification method and embedding method with a F-ratio of 35.297. The plot in Figure B.47 is a mirror image of sensitivity relative to the pants-on-fire class. In this case, discriminant analyses, except for FDA, perform poorly compared to other methods. However, they are still performing relatively well with high specificity values. The methods with the best specificity (FDA and random forest) have the worst sensitivity values relative to the pants-on-fire label, indicating these methods are mostly guessing the group membership. We see ordinal regression is primarily unreliable with BERT, PCA, and GloVe due to the distributions that appear to be a line. QDA seems most variable with PCA, Continuous Bag of Words in 100 dimensions, and Skip-Gram in 100 and 200 dimensions.

The main effect of including year is again significant (F-ratio = 122.45), where not including year as a predictor improves mean (and median) specificity seen in B.48. The difference in the means is only 0.0105783, which is not much. However, the difference in the medians is 0.03152, which does seem relevant. Not including year seems to lead to large variability.

The main effect of filtering stop words may also be significant (F-ratio = 9.214). The plot again reveals the F-ratio may be misleading, and there are possibly no differences in the mean specificity between the presence and removal of stop words (Figure B.49). The medians also appear similar between keeping and removing stop words.

As with sensitivity, embedding method is significant (F-ratio = 8.3). We again see large differences between the means and the medians (Figure B.50). Based on the means, BERT is one of the embedding methods performing the worst on average. However, based on the median, BERT is the best embedding method. Skip-Gram in 100 dimensions appears to be the most unreliable embedding method as it has the highest variability.

Lastly, we look at the main effect of classification method, which has an F-ratio of 1695.592. Outside of classification trees, FDA has the highest probability of correctly identifying statements as not pants-on-fire on average, whereas QDA is performing the worst. QDA has several outliers and the most variability out of all classification methods (Figure B.51).

In conclusion, if a person wants to make sure statements are correctly identified as not pants-on-fire, the use of FDA is recommended. One could also argue to use any combination including classification trees if they really do not want to classify any statement as pants-on-fire. With FDA, using GloVe or either Word2Vec method in combination improves the average specificity. For those combinations, it does not seem to make a difference the level of minimum appearance. Additionally, if a person uses FDA, not including year as a predictor slightly improves average specificity but not by much.

7.4 Conclusions

On average, all classification methods perform better than random chance in terms of overall accuracy. However, only LDA and MDA seem to be classifying the extreme classes of true and pants-on-fire properly. Neither overall accuracy or sensitivity for the extreme classes do much better than random chance. With LDA and MDA, including year as a predictor improves average sensitivity for the extreme cases as well as overall accuracy. Not filtering stop words, removing words not appearing a minimum of 5 times, and Skip-Gram in 100 dimensions combine individually with LDA and MDA to improve the average overall accuracy. BERT with LDA or MDA produce the highest average sensitivity relative to the true label as do these combinations for sensitivity relative to the pants-on-fire label. Additionally, filtering words not appearing a minimum of 30 times does best with LDA and MDA for correctly identifying pants-on-fire statements.

Unlike with the binary classification in the previous chapter, we do not see as strong of an effect of embedding method. The interactions with embedding methods and the preprocessing methods are not as prominent. The preprocessing methods seem to predominantly affect the classification methods. There does not seem to be consistency among the preprocessing methods and maximizing the classification metrics we explored in the multi-class scenario.

The inclusion of year as a predictor does not appear to be as impactful as predicted based on Figure 7.1 led us to believe. While including year as a predictor does improve the average overall accuracy, average sensitivity relative to the true class, and average sensitivity relative to the pants-on-fire class, it is not that large of a difference. The only metric including year as a predictor did not improve on average was specificity relative to the pants-on-fire class.

In general, if a person is not interested in classifying the extreme cases, then the use of classification trees is warranted. Classification trees never classify any statement as true or pants-on-fire, only in the middle classes. In terms of overall accuracy, this classification method is one of the worst, but it still does better than random guessing. FDA and random forest perform better with overall accuracy on average compared to classification trees, but they still do not perform accurately with the extreme cases. The fact that classification trees performs practically as well as the other classification methods overall while never classifying statements into the extreme classes makes us think there is not enough distinguishing characteristics between word embedding methods and the inclusion of year to distinguish the classes, and one is better off just making a guess.

More work is needed to explore other predictors to be used to distinguish between the six classes in the PolitiFact data set. Semantic information such as number of nouns, verbs, etc. could be considered as additional predictors of fake news. Additionally, information regarding the political affiliation of the speaker or writer could be beneficial, as well as an interaction between political affiliation with year it was made. At this time, we do not have this information and is considered for future work. Lastly, stemming was again not considered with the multi-class classification and is hypothesized to have a significant effect.

Chapter 8

Conclusions and Future Work

We set out to determine how well word embedding methods alone detected fake news, if treating words as independent and methods capturing context behaved differently. The Bag of Words and TF-IDF methods treat words as independent. Because of the nature of these methods, the matrices are highly sparse, leading to computational issues with many of the classification methods. As a result, they were only used with the binary outcome classification. As seen in Chapter 5, these methods perform poorly compared to the embedding methods capturing context, except in the probability of correctly identifying fake news. While PCA on the document-term matrix has the underlying assumption of independence between the words, this method does perform similarly to the word embedding methods capturing context. There does not appear to be a single embedding and classification method combination outperforming the others. We did not have major differences indicating word embeddings alone classified statements well.

However, all of these embedding methods are dependent on the word preprocessing methods utilized in natural language processing procedures. A determination of the usefulness of the word embeddings alone cannot be made without taking into consideration the effects of the preprocessing techniques. In this case, we explored the effects of filtering stop words, filtering words not appearing a minimum number of times, converting words to lowercase, and removing numbers on the classification of fake news in Chapter 6. When only considering the binary case of real and fake news, other than classification method and embedding method, the filtering of stop words and filtering based on minimum appearances had the most influence on the classification metrics explored. Again, not one embedding method and combination of preprocessing methods seemed to consistently surpass the others. It depended on the classification metric of interest. For accuracy and precision (positive predictive value), BERT and PCA worked the best. BERT did better with less filtering, but PCA was the opposite. They both did best with stop words but in combination with different classification methods. In terms of correctly identifying real news (sensitivity) and correctly identifying fake news (specificity), GloVe and Skip-Gram were the better embedding methods to use. There was an inverse relationship between these metrics. Combinations with high sensitivity had low specificity, as seen in Tables 8.1 and 8.2. The combination that produced the highest observed probability of correctly classifying real news preformed the worst in terms of the observed probability of correctly classifying fake news. Even through exploring the effects of the preprocessing methods, we still find the word embeddings alone do not provide enough to accurately classify political statements as they still do not perform much better than random chance. Thus, future work might include other predictors to explore how it affects classification.

Lastly, we also explored whether the results of the binary classes are confounded with the fact the PolitiFact data set contains six classes. Additionally, we explored the effect of including the year the statement was made. We only focused on the preprocessing methods deemed significant in the binary case: filtering stop words and filtering words not appearing a minimum number of times. In the case of the multiple classes explored in Chapter 7, the preprocessing methods did not have the same effect on the embedding methods as they did with the binary case, only with the classification methods. In terms of overall accuracy and correctly identifying the extreme cases, employing the classification methods of LDA and MDA offers the most reliable improvement. QDA also does, but it is the most unreliable and variable. Classification trees never classify any statement into those categories. Including year as a predictor does also seem to improve the overall accuracy and identifying the extreme cases. However, the use of word embeddings and year still does not appear to be enough in the multiple categories classification as it does not improve much beyond random chance. Tables 8.3 and 8.4 reveal the maximum and minimum observed value for the metrics we explored for the multiple classes scenario. One will see that the combination giving the maximum sensitivity for each label is the same combination producing the minimum specificity for each label. This again shows the classification methods that appear to be working "best" may actually be unreliable and just selecting a random category to classify statements.

Even though the inclusion of the year a statement was made only slightly improved classification with the multiple classes, it is still of interest to explore how this affects the binary case. Figure 8.1 reveals that much like the multiple classes, the year a statement was made is associated with the category of the statement. Because multi-class classification is a much more difficult problem, the effect of year may be more pronounced with binary classification. This will be pursued in future work.

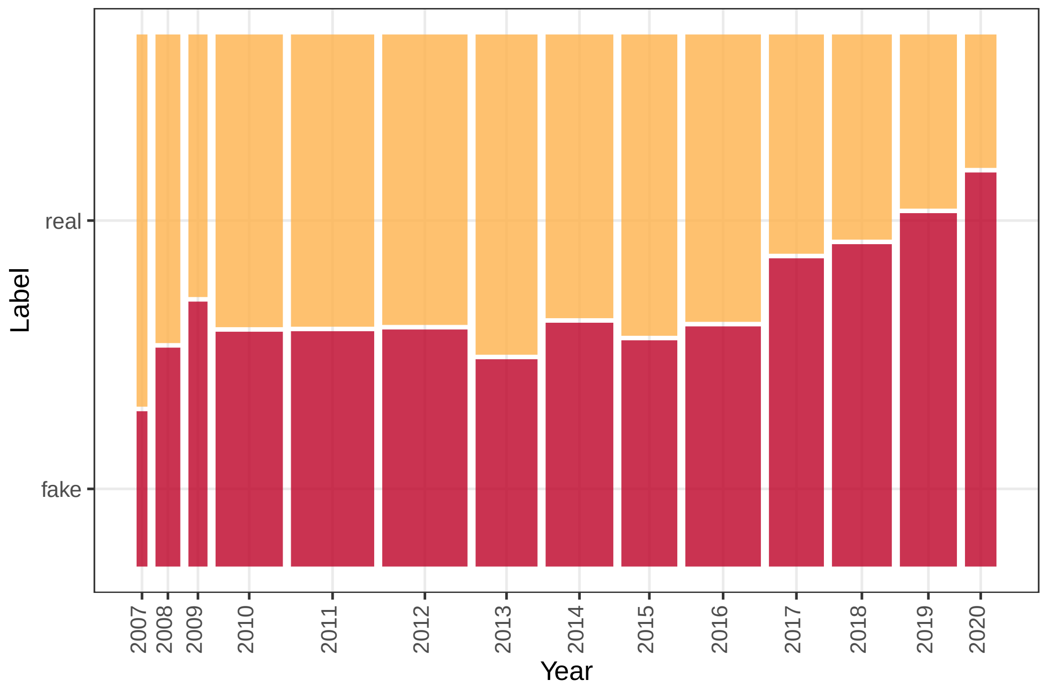

Figure 8.1: The breakdown of the binary classes per year observed in our data set.

In addition to including year as a predictor, other demographic information could be of use in the classification of fake news, such as political party of the speaker of the statement, whether or not it was from a blog, etc. We have information regarding the speaker or writer of the statement and can easily find this information, such as political party, to add to our classification methods. Relating to the classification methods, we utilized more classical methods, such as discriminant analyses. These classical methods are non-sequential, meaning they do not take into account the order of the words, which could be worthwhile. Investigating sequential classification methods which take into account order of the words is also a future direction of this research.

With the preprocessing methods, we did not explore the effect of stemming words. Recall the act of stemming is trimming a word down to its root. For instance, in the previous sentence, stemming would shorten the word "trimming" down to "trim." This allows words with essentially the same meaning to be treated as the same word, which, in theory, greatly reduces the amount of noise in the data set. There are multiple stemming algorithms in existence, some more conservative in how much they trim than others. Thus, the type of stemming algorithm used can play a role in the embedding methods as well.

It is important to note that all of this exploration into the effect of the preprocessing methods is limited to only this data set. We do not have a random sample and, more than likely, is not representative of all possible political statements ever made. In addition, we are limited to politics in the United States and English speakers. PolitiFact does its best to pick statements from a wide range of speakers and political ideologies, but they openly admit the party in power may be sampled more often. Thus, finding a data set from another reliable source documenting fake news as well as collecting more recent statements from PolitiFact is important to establishing the relevance of this work to a broader context. We need to explore just how generalizable these classification methods are. We can also expand past fake news to other classification problems, such as hate speech detection.

Lastly, with the word embedding methods, we averaged across the dimensions of the word embedding vectors for the words in the statement. This may not be the best method to get a statement level embedding. Other methods involve using the minimum or maximum of each dimension across all words in the statement. The median of each dimension could also be a useful method of finding statement level embeddings. Exploring other options to obtain these statement level of embedding is of interest in a future work.

Table 8.1: The classification method, embedding method, and preprocessing methods combination creating the maximum for each metric for binary classification.

Metric	Classification Method	Embedding Method	Filter stop words?	Convert to Lowercase?	Filter Numbers?	Minimum Appearances	Maximum
Accuracy	LogReg	PCA	No	No	No	60	0.63950
Precision	QDA	SG768	No	No	No	0	0.68290
Sensitivity	LogReg	SG768	No	No	Yes	60	0.90870
Specificity	LogReg	SG768	No	No	No	60	0.91838

Table 8.2: The classification method, embedding method, and preprocessing methods combination creating the minimum for each metric for binary classification.

Metric	Classification Method	Embedding Method	Filter stop words?	Convert to Lowercase?	Filter Numbers?	Minimum Appearances	Minimum
Accuracy	LogReg	CBoW768	Yes	No	Yes	60	0.50660
Precision	LogReg	CBoW768	Yes	No	Yes	60	0.49740
Sensitivity	LogReg	SG768	No	No	No	60	0.17719
Specificity	LogReg	SG768	No	No	Yes	60	0.19750

Table 8.3: The classification method, embedding method, and preprocessing methods combination creating the maximum for each metric for the muliple classes classification.

Metric	Classification Method	Embedding Method	Filter stop words?	Include Year?	Minimum Appearances	Maximum
Overall Accuracy	RF	CBoW768	Yes	Yes	5	0.28340
Sensitivity-True	OrdReg	PCA	No	No	5	0.30205
Sensitivity-PantsFire	QDA	SG100	Yes	No	0	0.64375

Table 8.4: The classification method, embedding method, and preprocessing methods combination creating the minimum for each metric for the muliple classes classification.

Metric	Classification Method	Embedding Method	Filter stop words?	Include Year?	Minimum Appearances	Minimum
Overall Accuracy	OrdReg	PCA	Yes	No	0	0.1938
Specificity-True	OrdReg	GloVe768	Yes	No	5	0.7880
Specificity-PantsFire	QDA	SG100	Yes	No	0	0.4896

Appendix A

Plots for Chapter 6

This appendix contains all interaction and main effect mean plots discussed in Chapter 6 for effects with a F-ratio greater than 2. It also contains appropriate violin plots for the main effect and two-way interactions to give us an understanding of the distribution of observations.

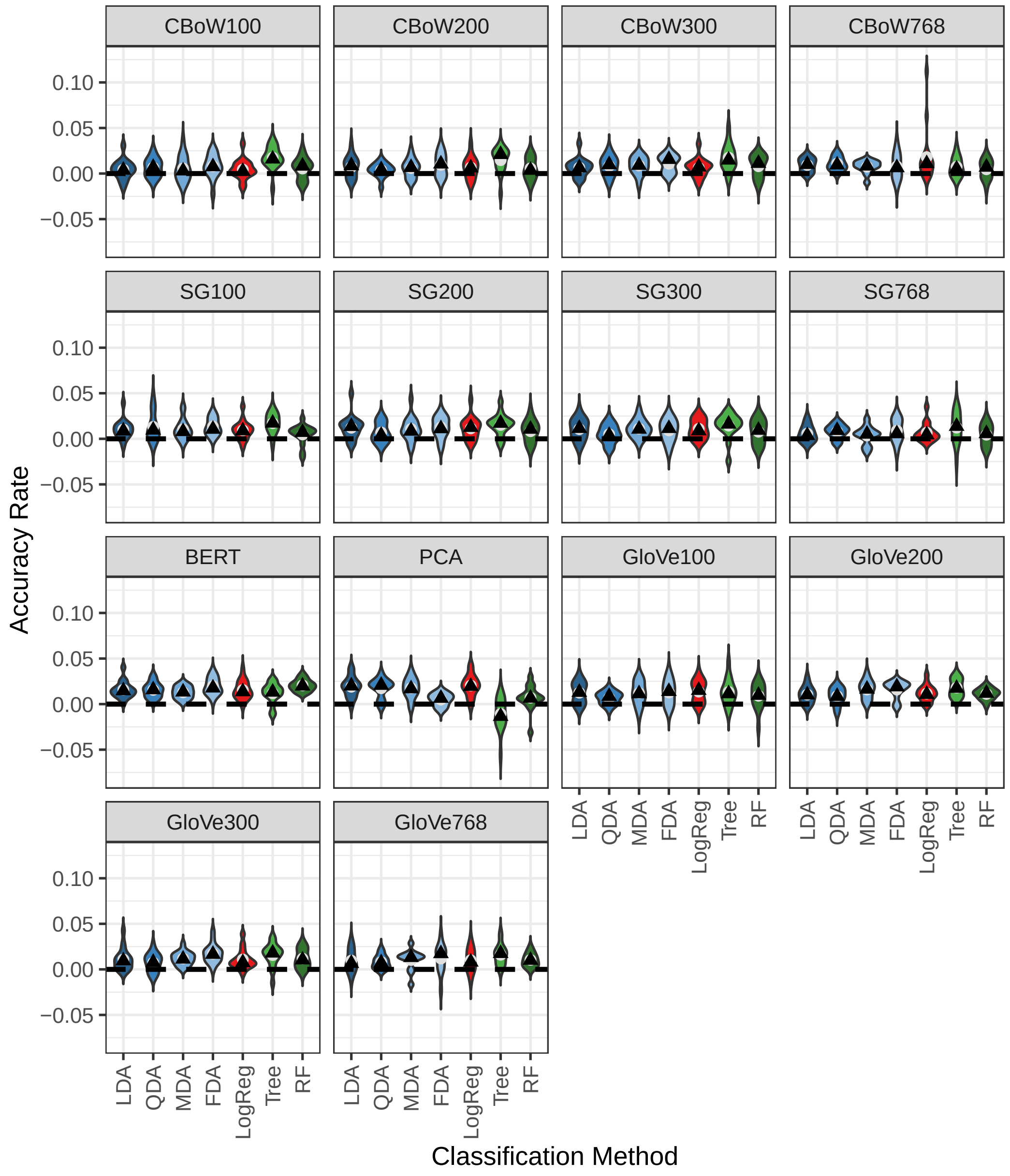

Figure A.1: $C \times E \times S$, F-ratio = 2.459. The difference in accuracy between filtering of stop words (no-yes) for each combination of classification and embedding method. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

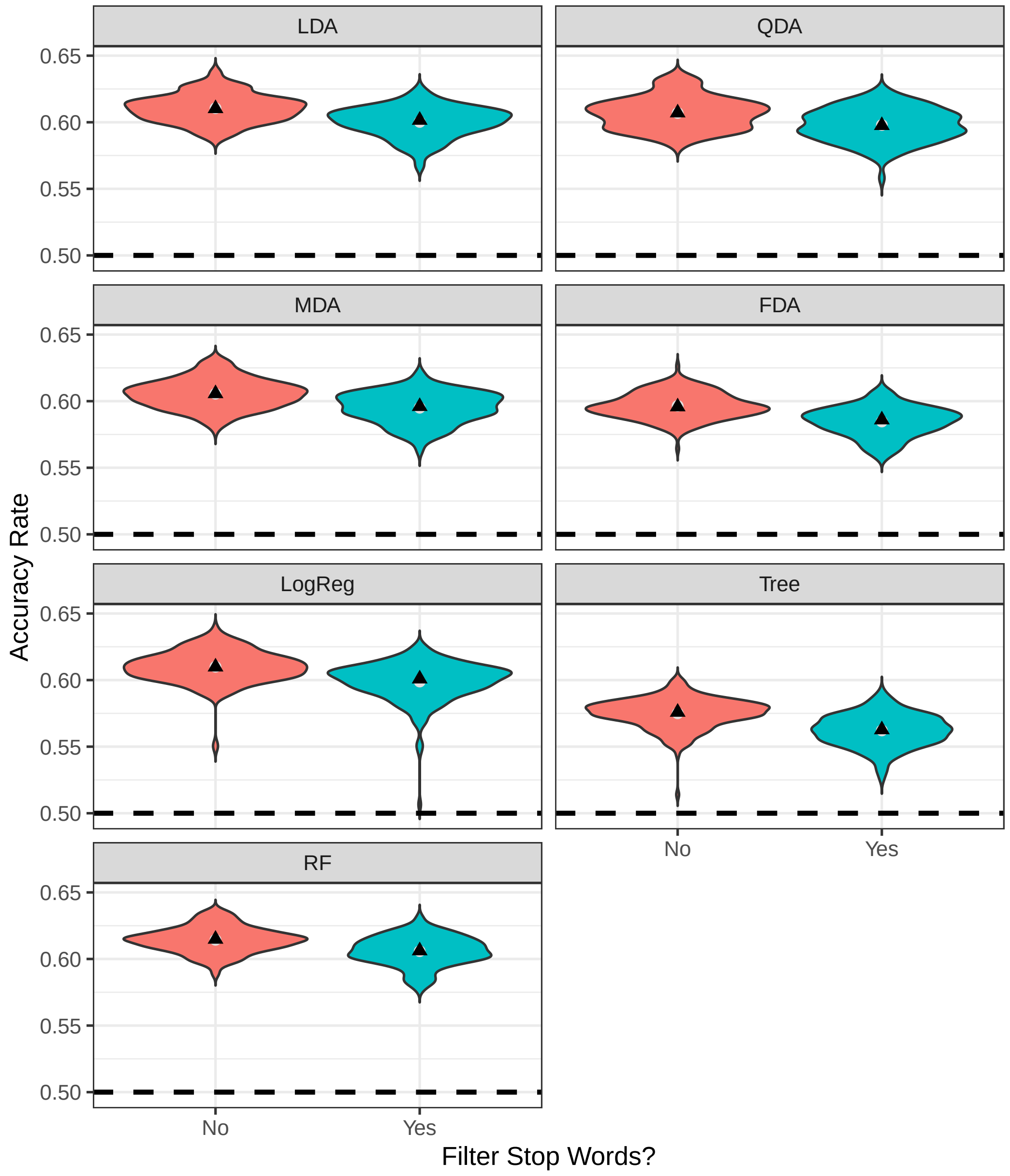

Figure A.2: $C \times S$, F-ratio = 4.065. The effect of filtering stop words by classification method on accuracy. The median is a black triangle, and the mean is a grey circle.

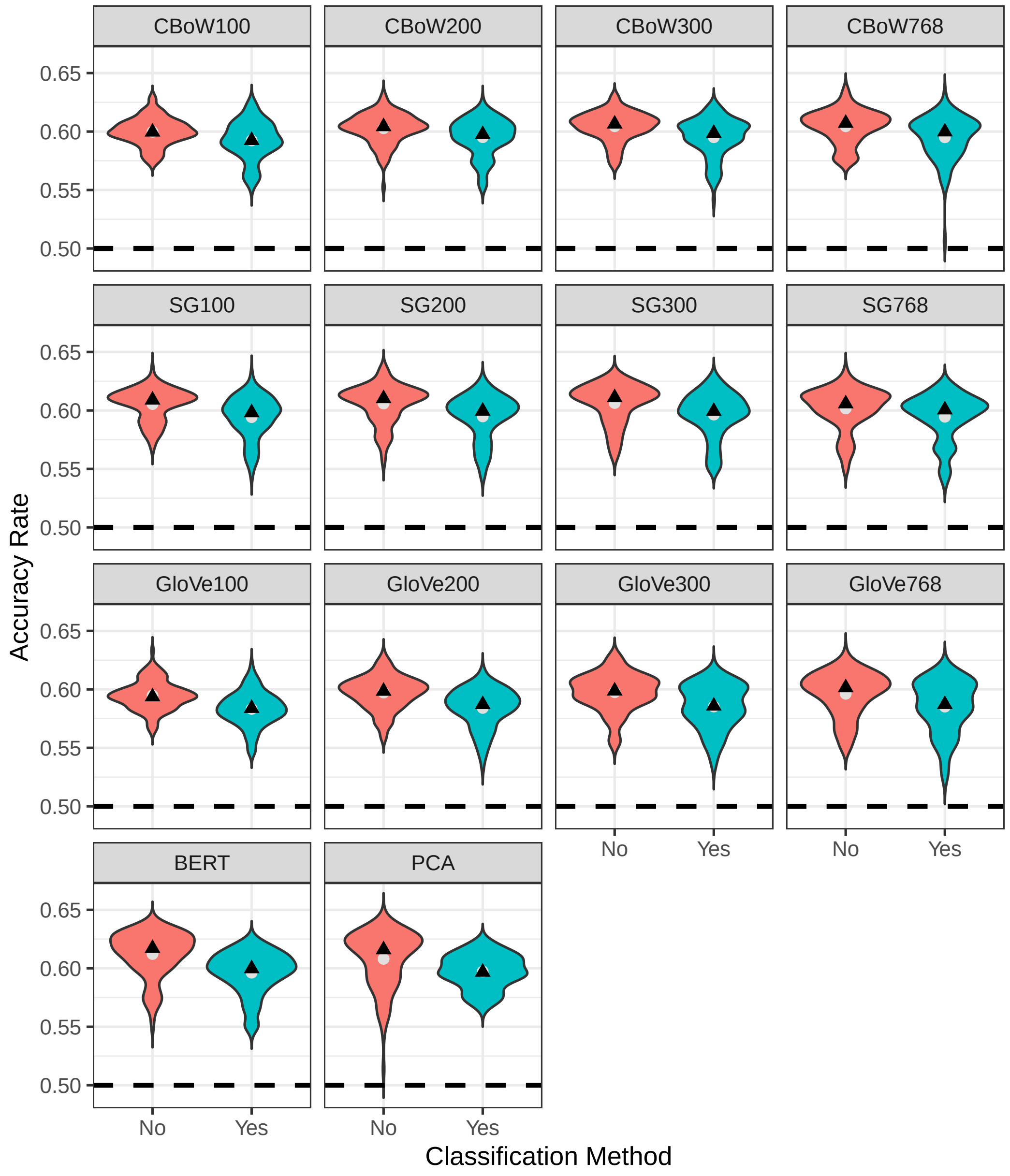

Figure A.3: $E \times S$, F-ratio = 5.061. The effect of filtering stop words by embedding method on accuracy. The median is a black triangle, and the mean is a grey circle.

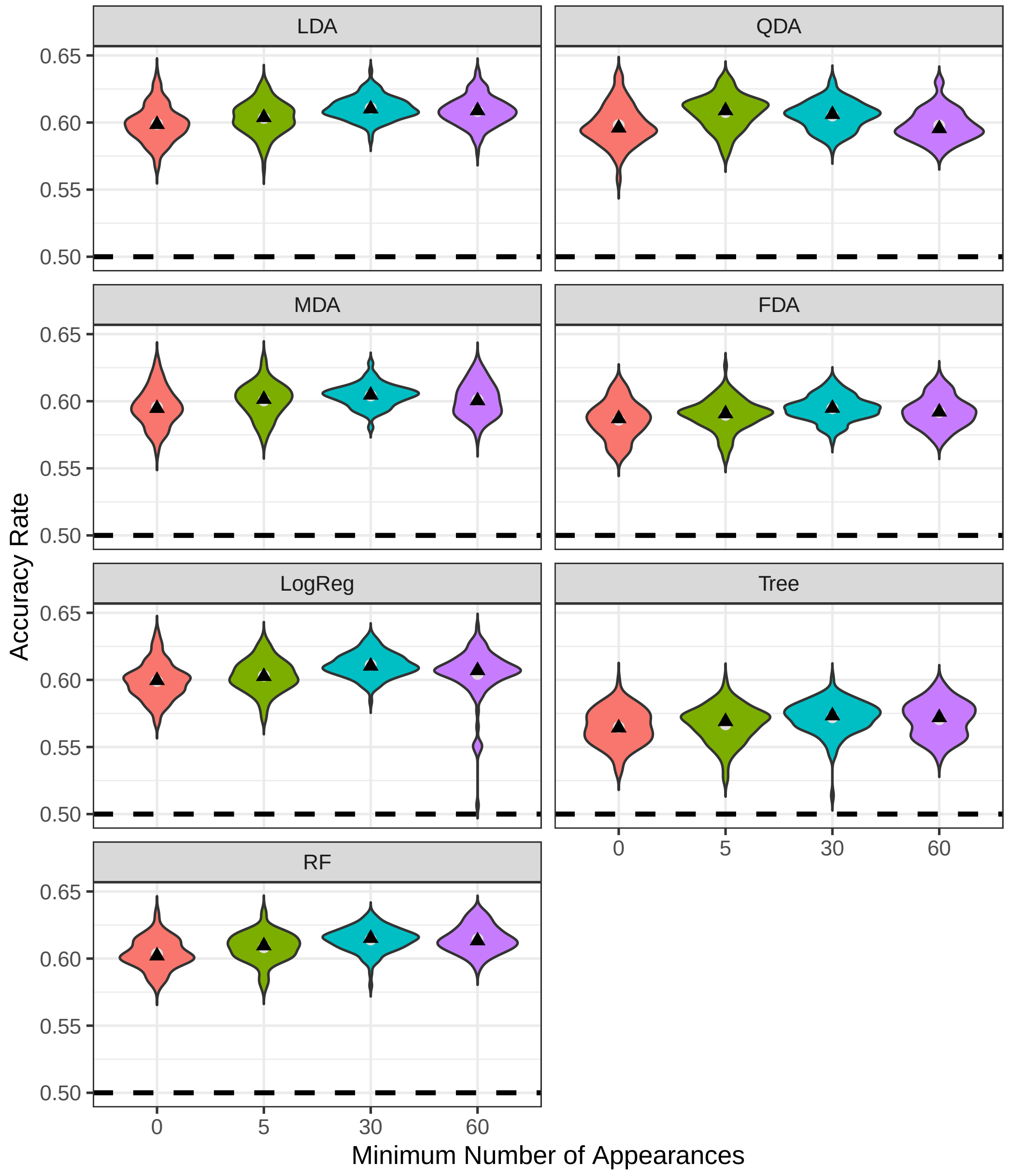

Figure A.4: $C \times F$, F-ratio = 8.586. The effect of filtering based on minimum number of appearances by classification method on accuracy. The median is a black triangle, and the mean is a grey circle.

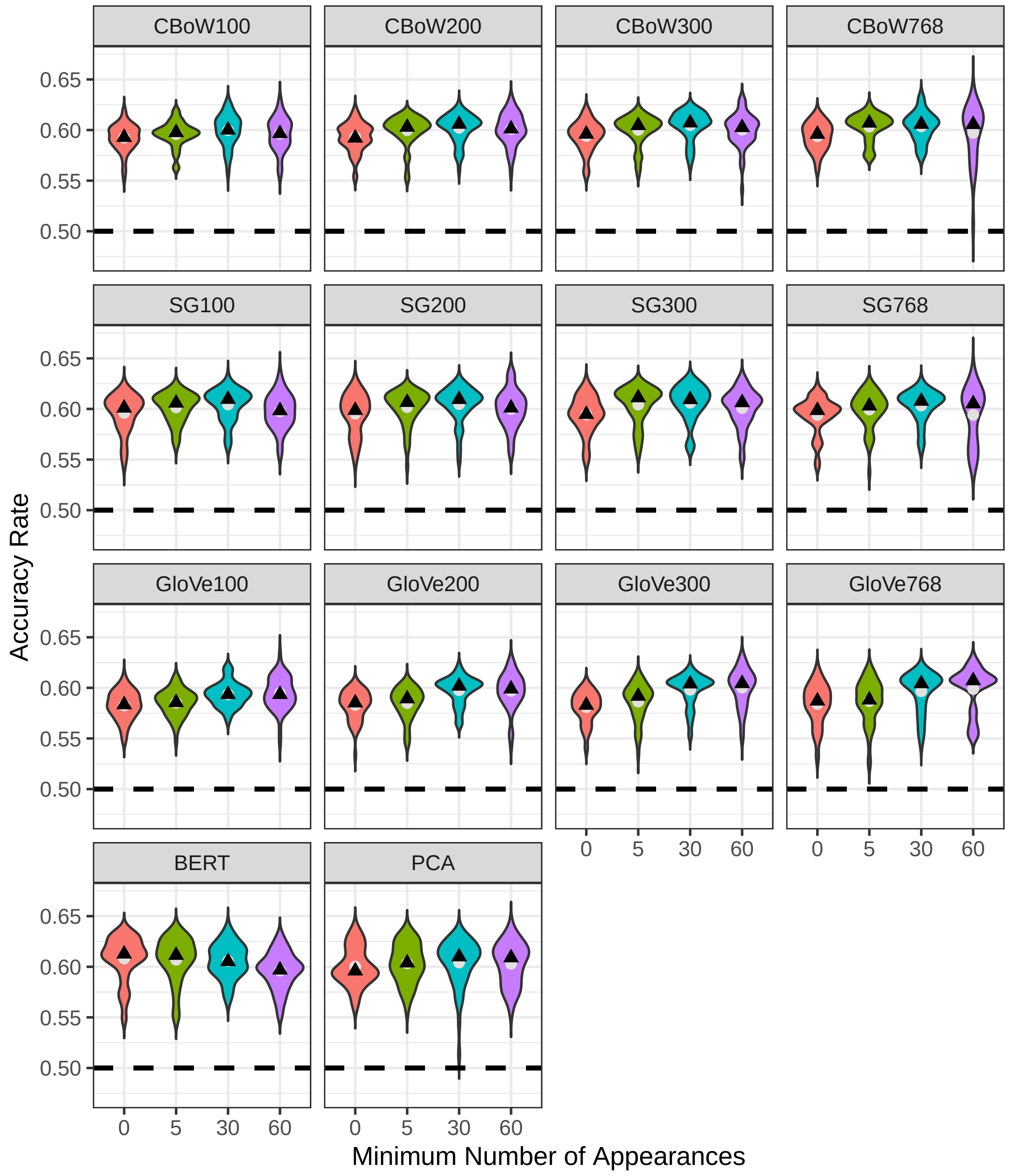

Figure A.5: $E \times F$, F-ratio = 13.043. The effect of filtering based on minimum number of appearances by embedding method on accuracy. The median is a black triangle, and the mean is a grey circle.

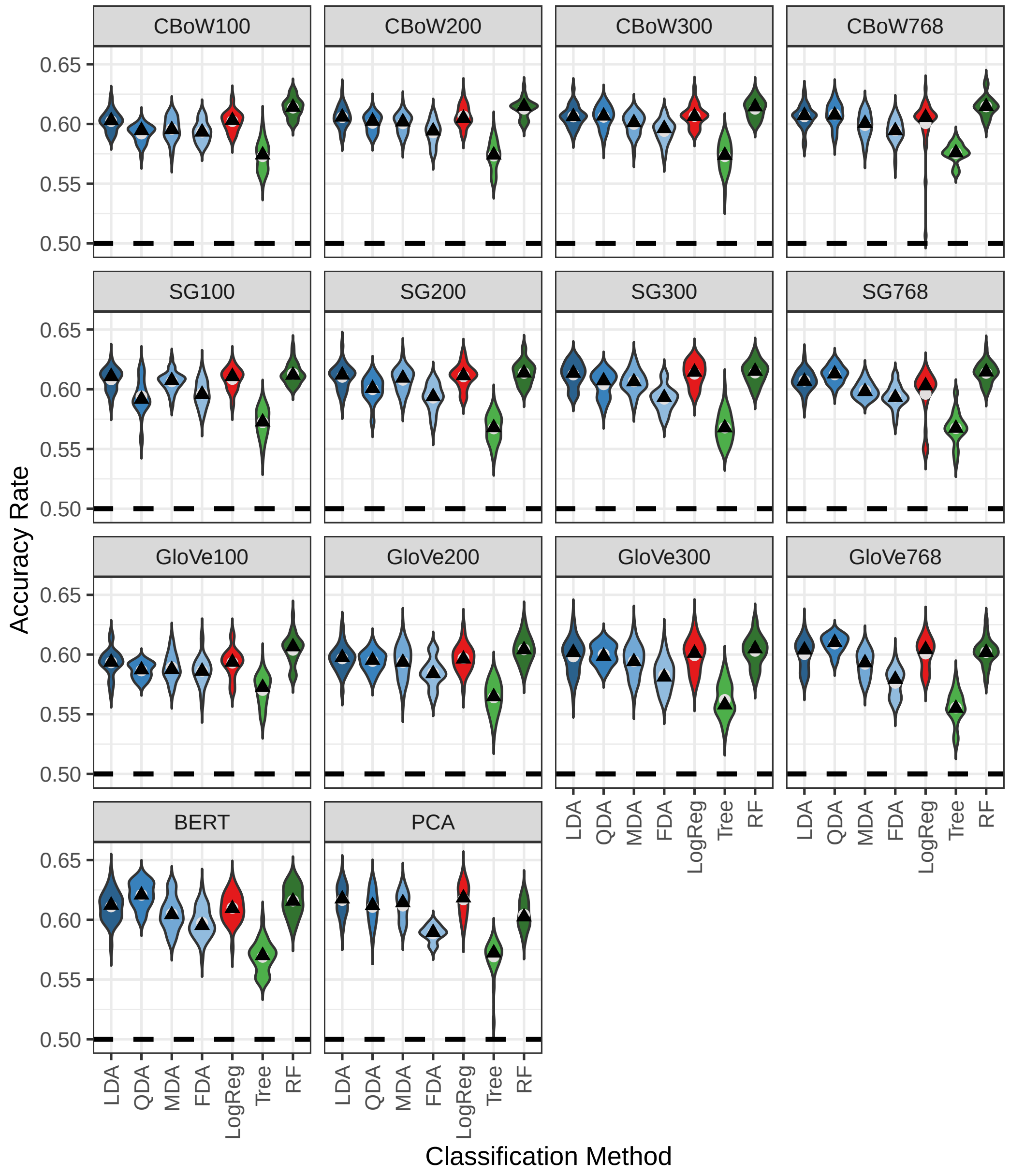

Figure A.6: $C \times E$, F-ratio = 11.359. The effect of classification method by embedding method on accuracy. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

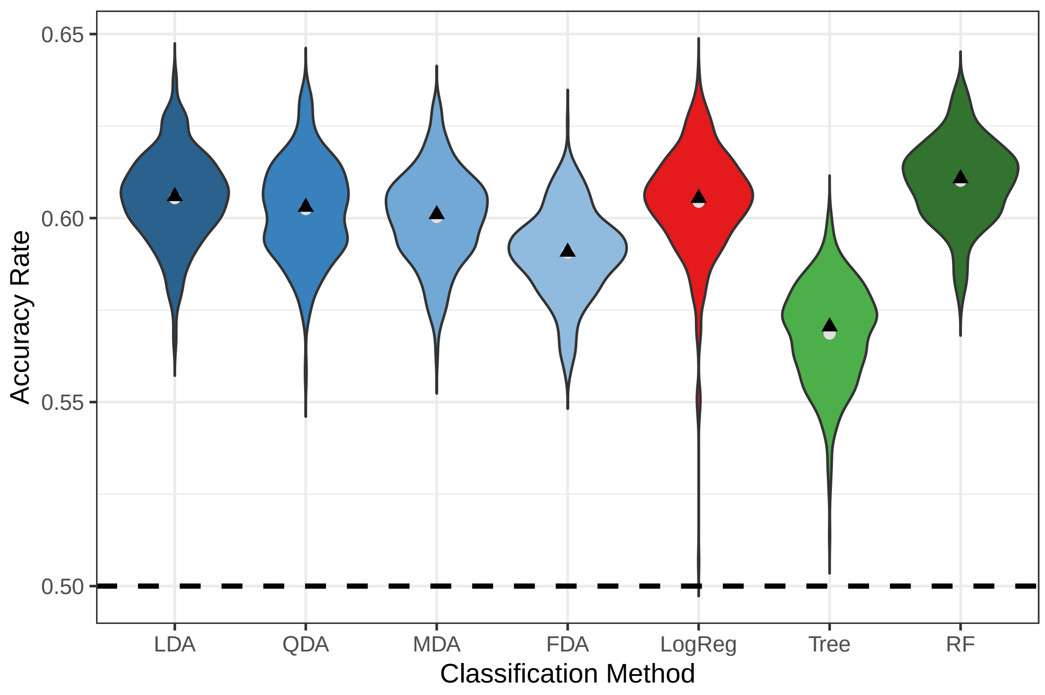

Figure A.7: C, F-ratio = 1425.005. The effect of classification method on accuracy. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

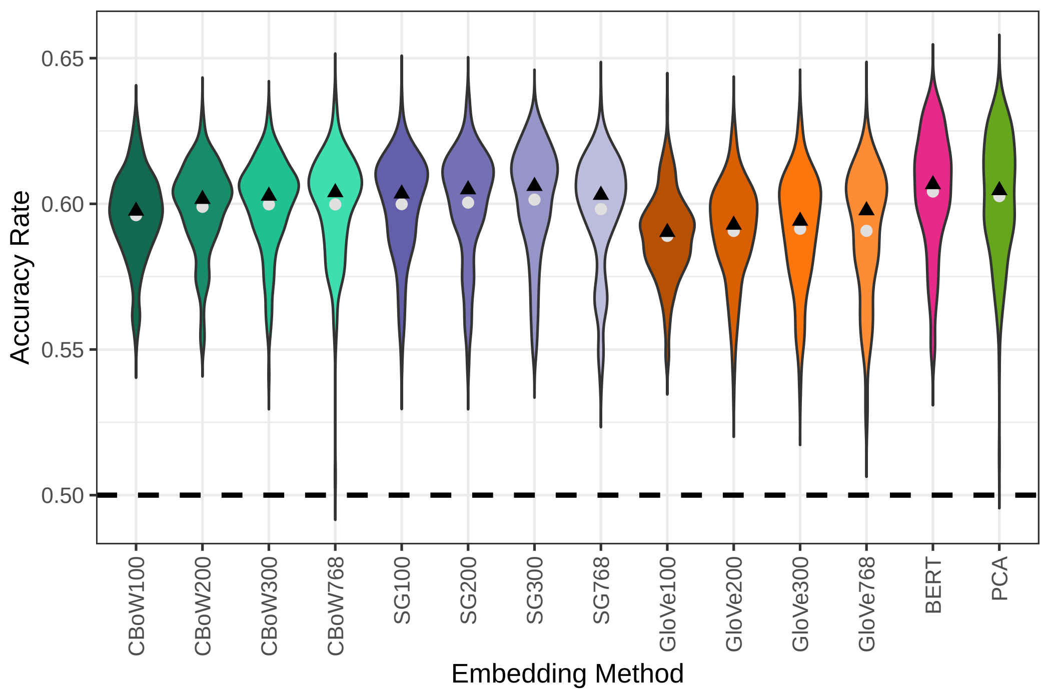

Figure A.8: E, F-ratio = 86.977. The effect of embedding method on accuracy. Word2Vec Continuous Bag of Words are in the blue-green shades, and Word2Vec Skip-Gram are in the blue-purple shades. GloVe is in the brown-orange shades. BERT is in pink, and PCA is in green. The median is a black triangle, and the mean is a grey circle.

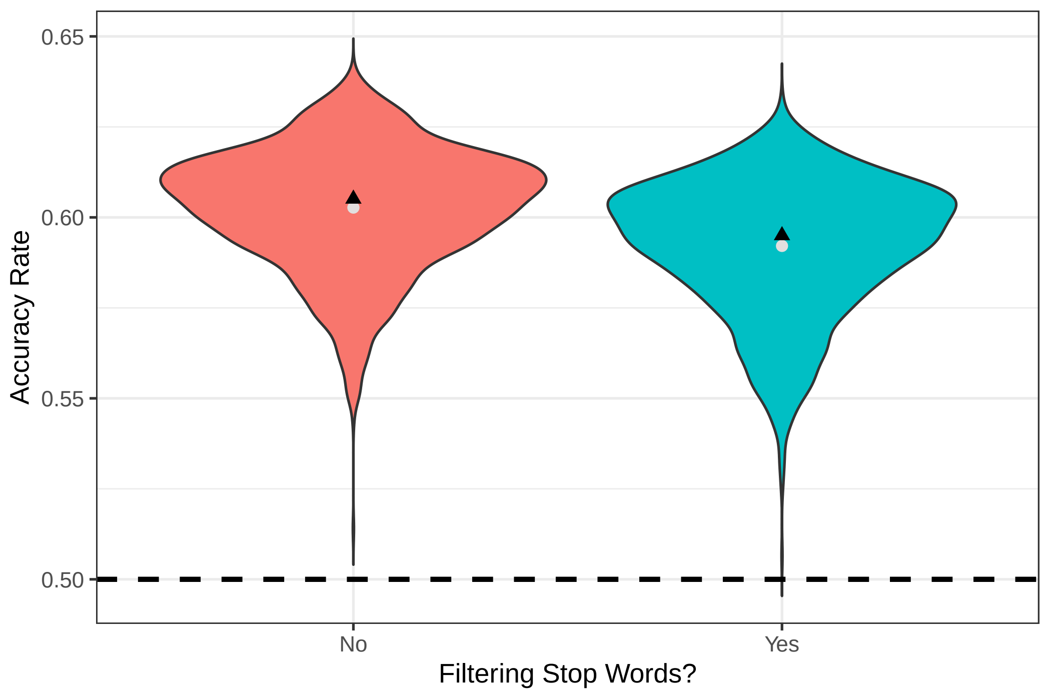

Figure A.9: S, F-ratio = 1389.398. The effect of filtering stop words on accuracy. The median is a black triangle, and the mean is a grey circle.

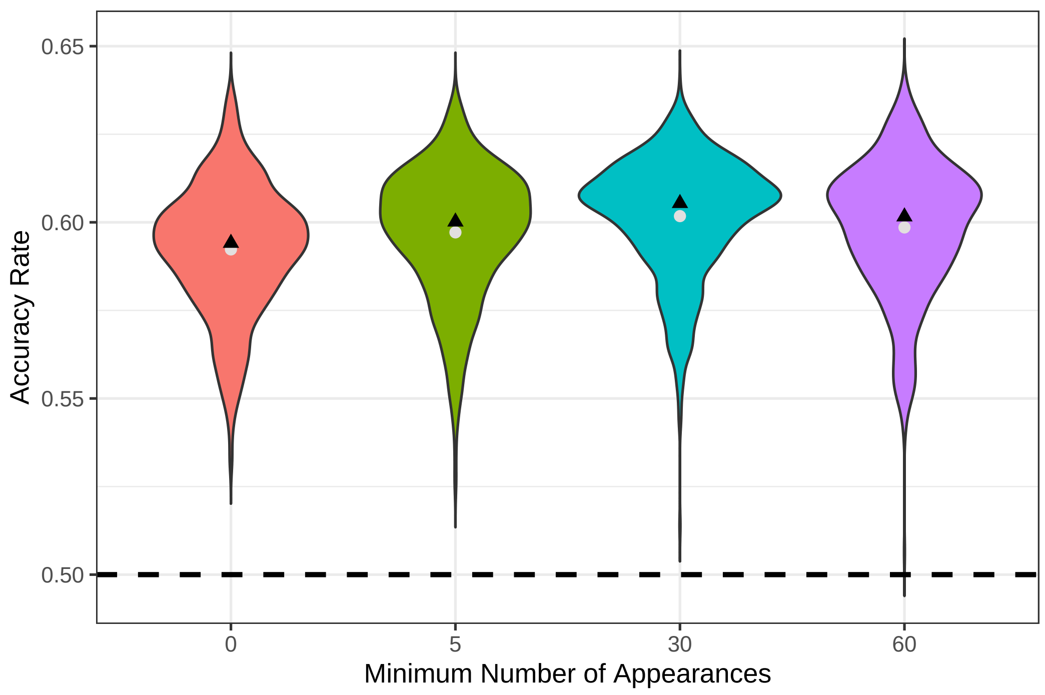

Figure A.10: F, F-ratio = 196.127. The effect of filtering words based on minimum number of appearances on accuracy. The median is a black triangle, and the mean is a grey circle.

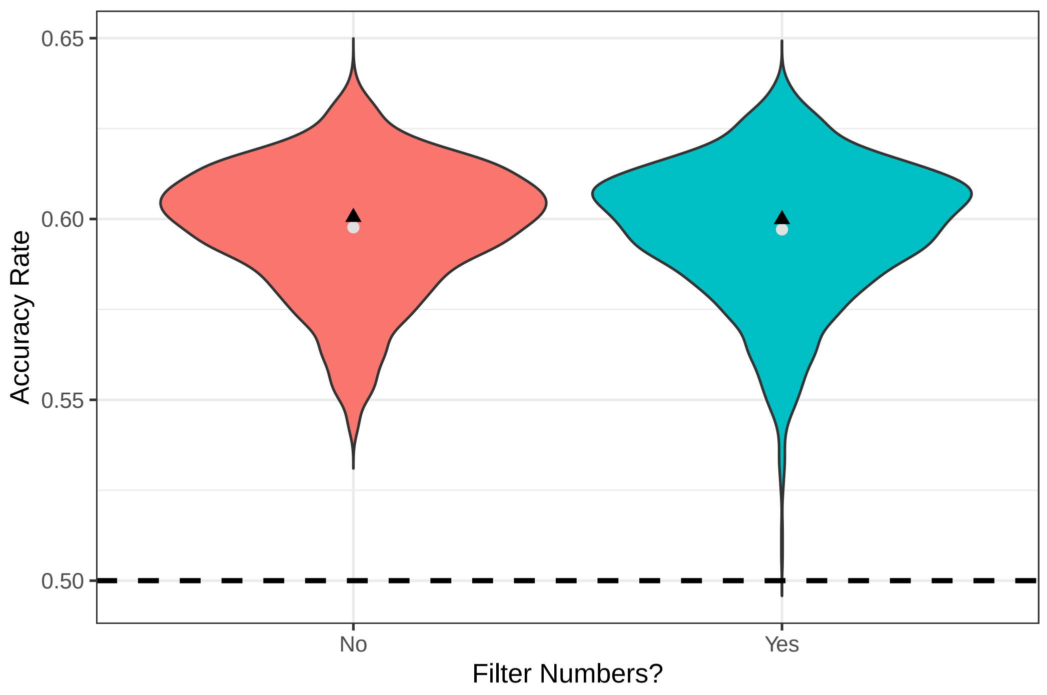

Figure A.11: N, F-ratio = 4.074. The effect of filtering numbers on accuracy. The median is a black triangle, and the mean is a grey circle.

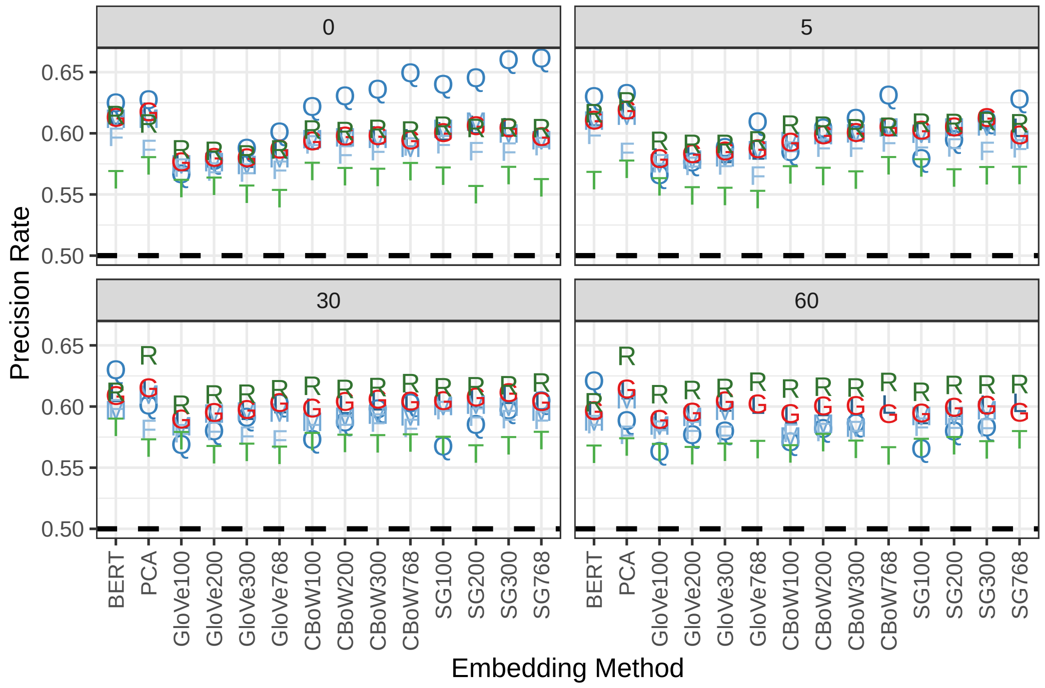

Figure A.12: $C \times E \times F$, F-ratio = 3.334. The effect of classification method, embedding method, and filtering threshold on precision. Each classification method is represented by a letter. T is for classification tree, and G is for logistic regression. All others are based on the first letter of the method name. Only the means are plotted here. Violin plots are not used due to the number of levels of filtering threshold.

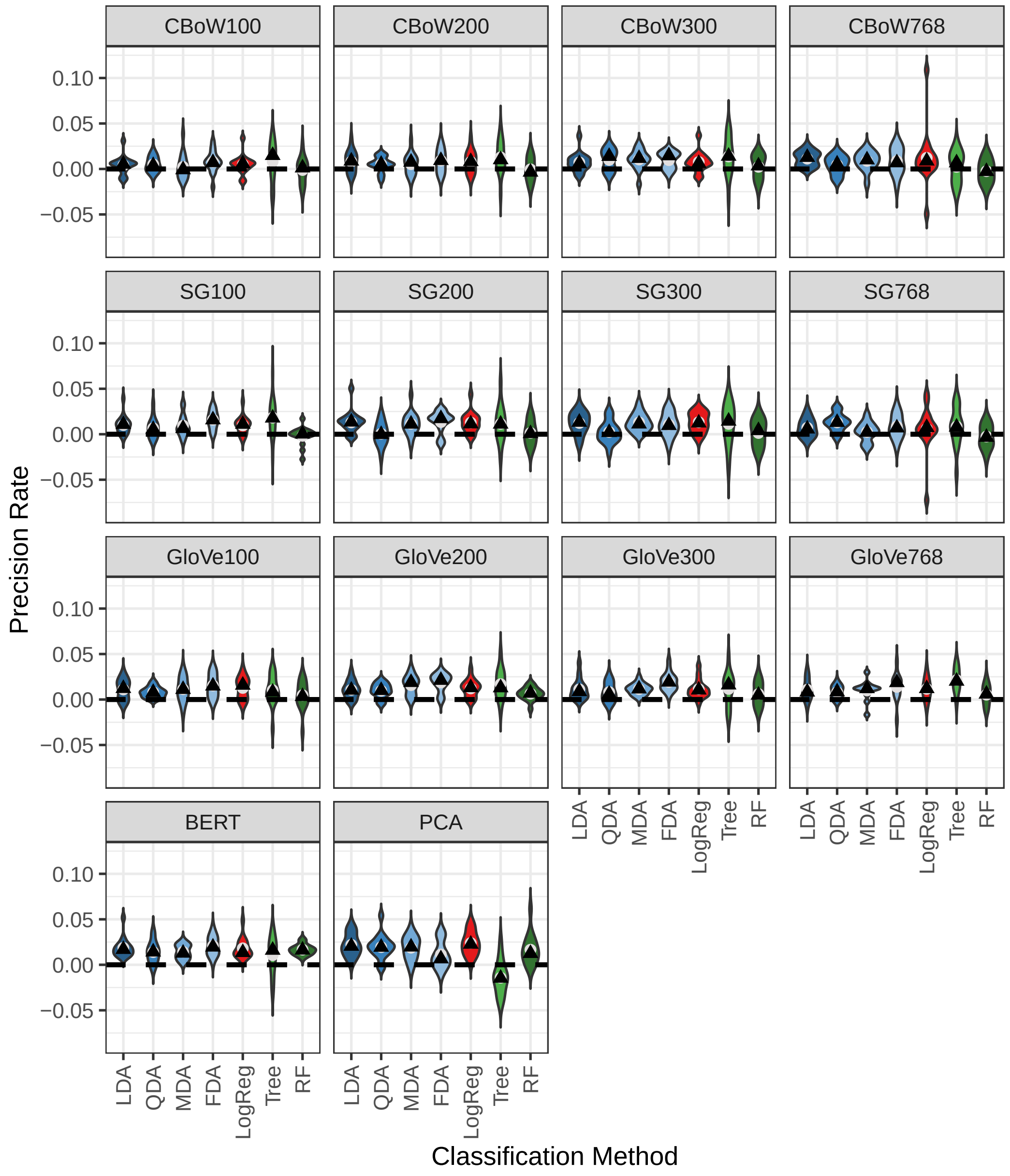

Figure A.13: $C \times E \times S$, F-ratio $= 2.17$. The difference in precision between filtering of stop words (no-yes) for each combination of classification and embedding method. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

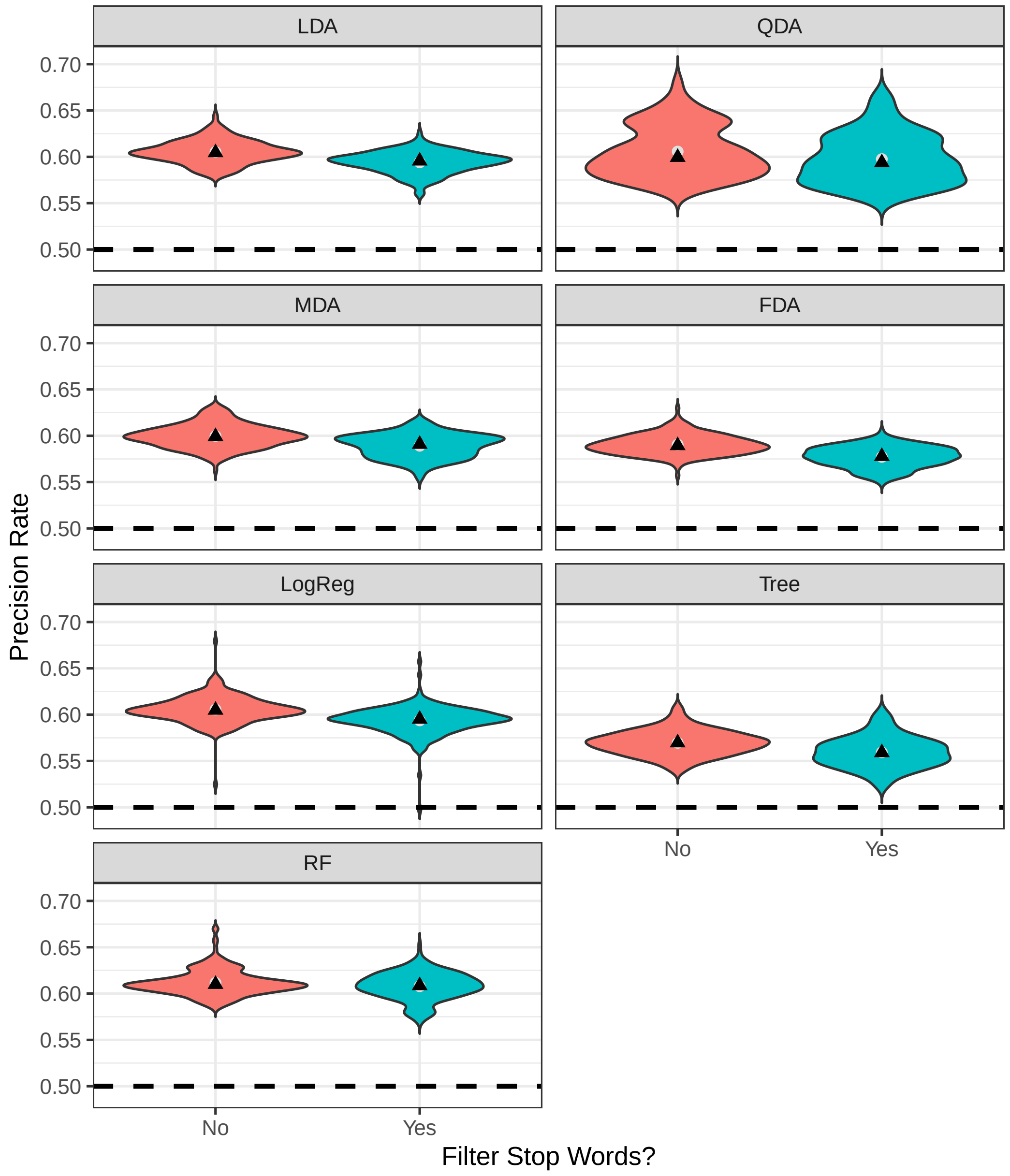

Figure A.14: $C \times S$, F-ratio = 11.891. The effect of filtering stop words by classification method on precision. The median is a black triangle, and the mean is a grey circle.

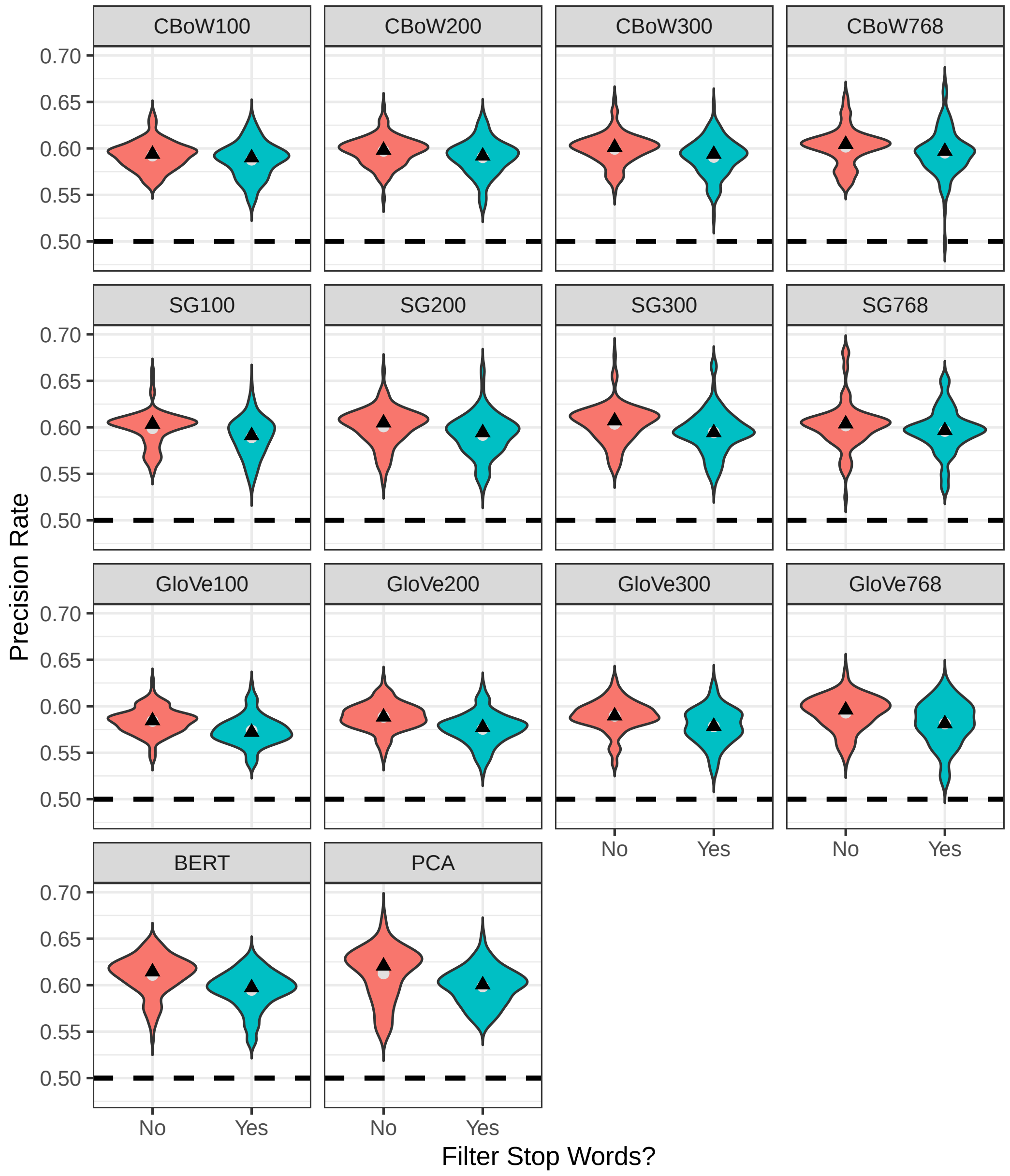

Figure A.15: $E \times S$, F-ratio = 6.576. The effect of filtering stop words by embedding method on precision. The median is a black triangle, and the mean is a grey circle.

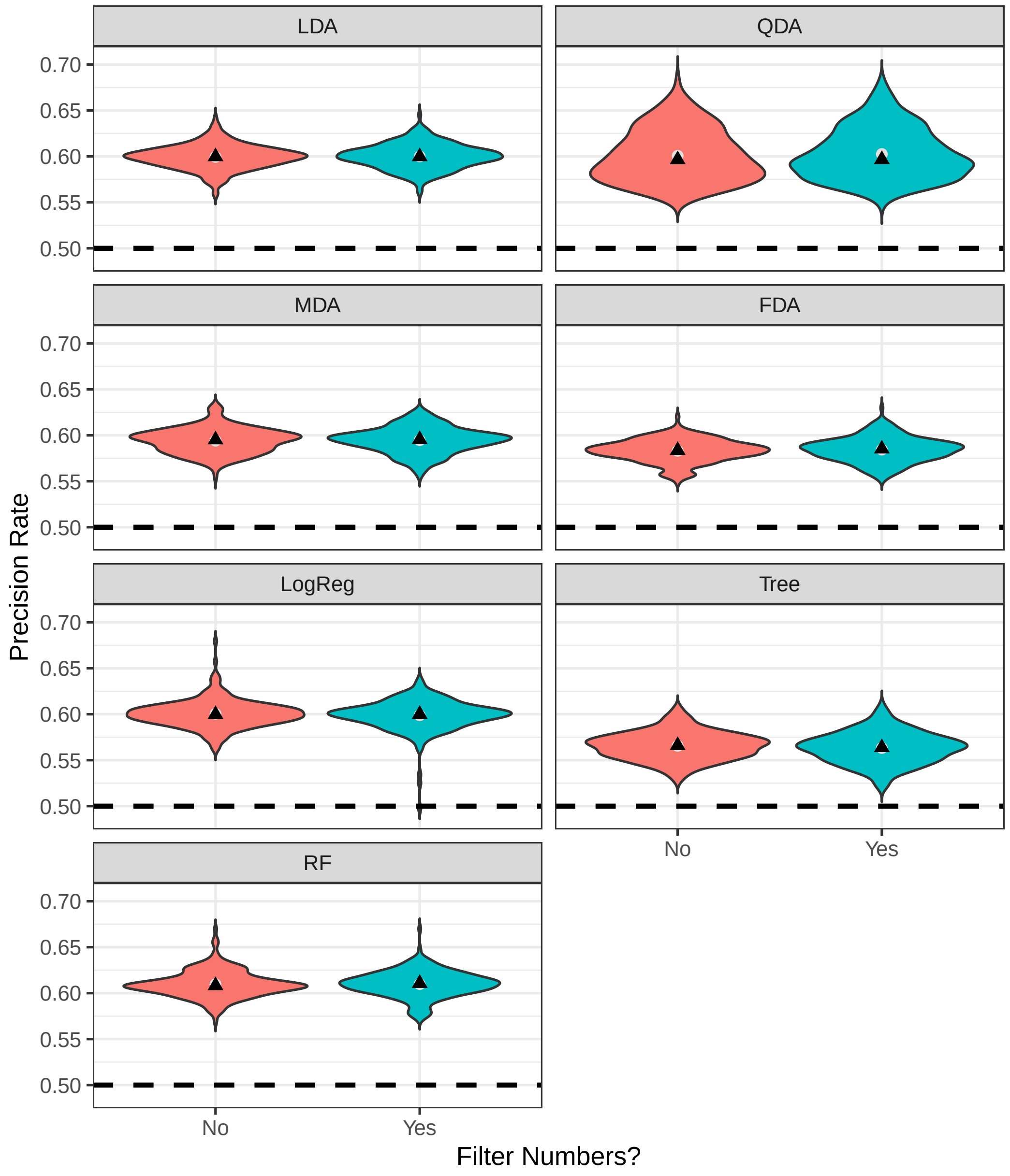

Figure A.16: $C \times N$, F-ratio = 2.814. The effect of filtering numbers by classification method on precision. The median is a black triangle, and the mean is a grey circle.

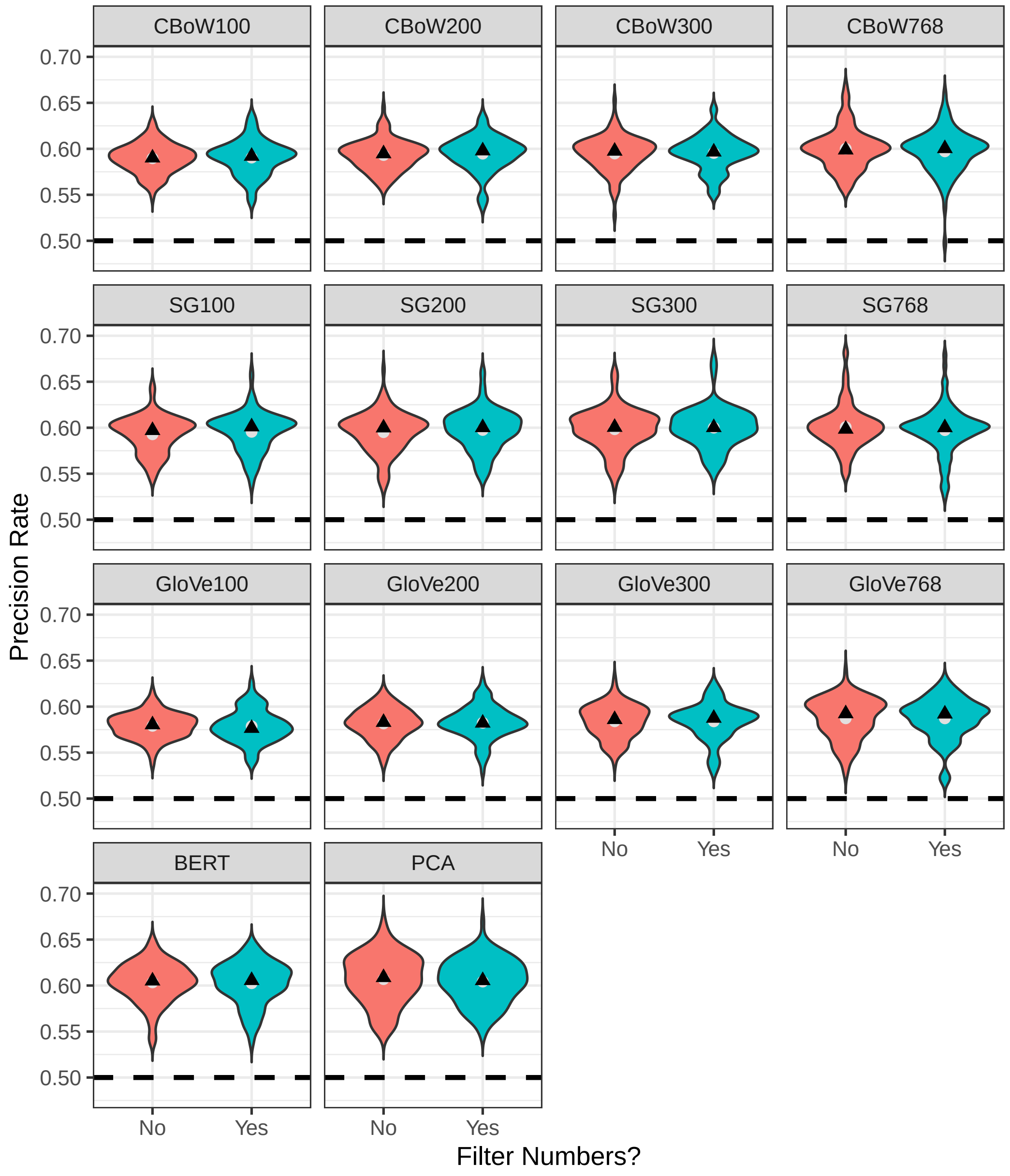

Figure A.17: $E \times N$, F-ratio = 2.089. The effect of filtering numbers by embedding method on precision. The median is a black triangle, and the mean is a grey circle.

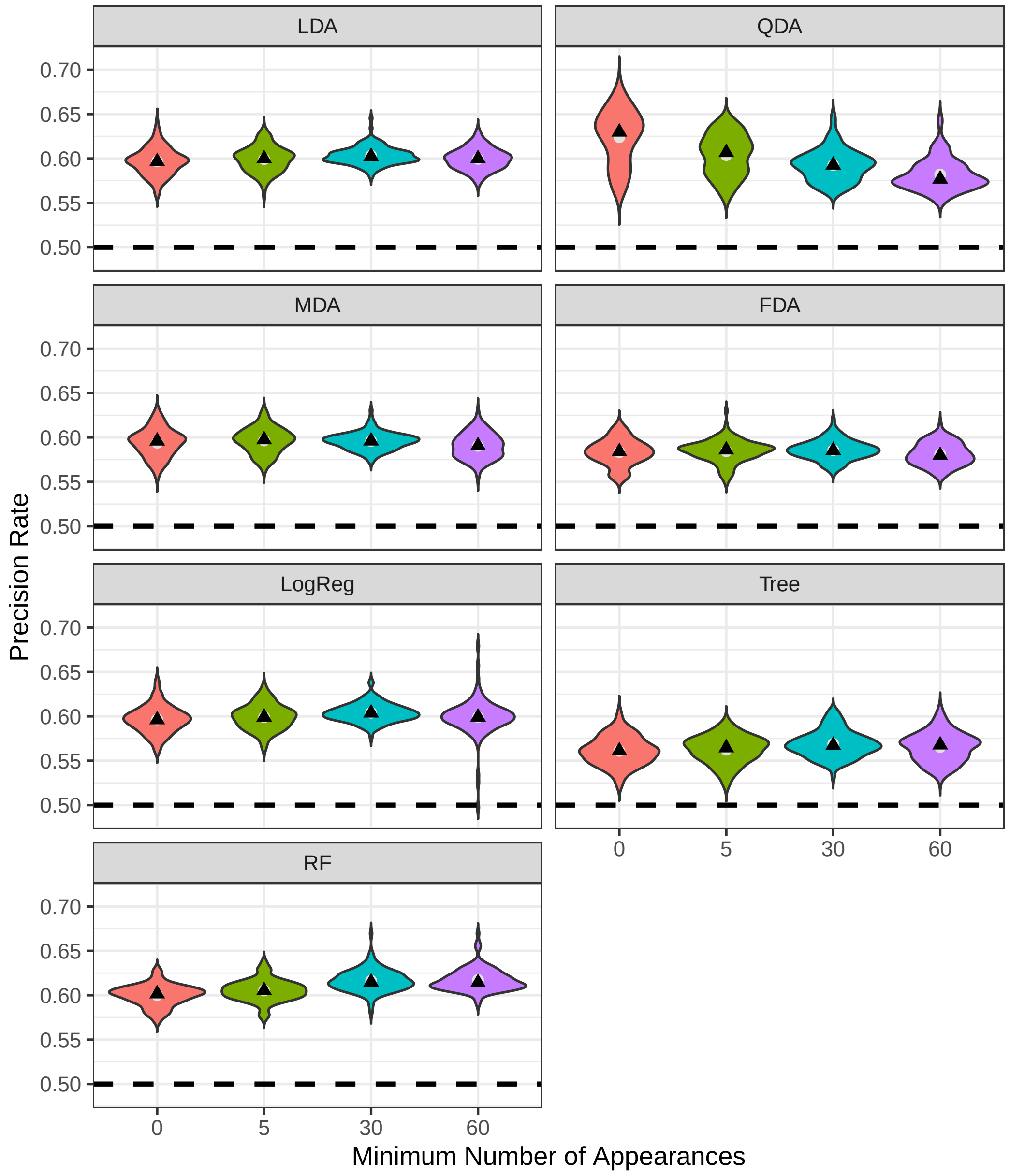

Figure A.18: $C \times F$, F-ratio = 71.954. The effect of filtering threshold by classification method on precision. The median is a black triangle, and the mean is a grey circle.

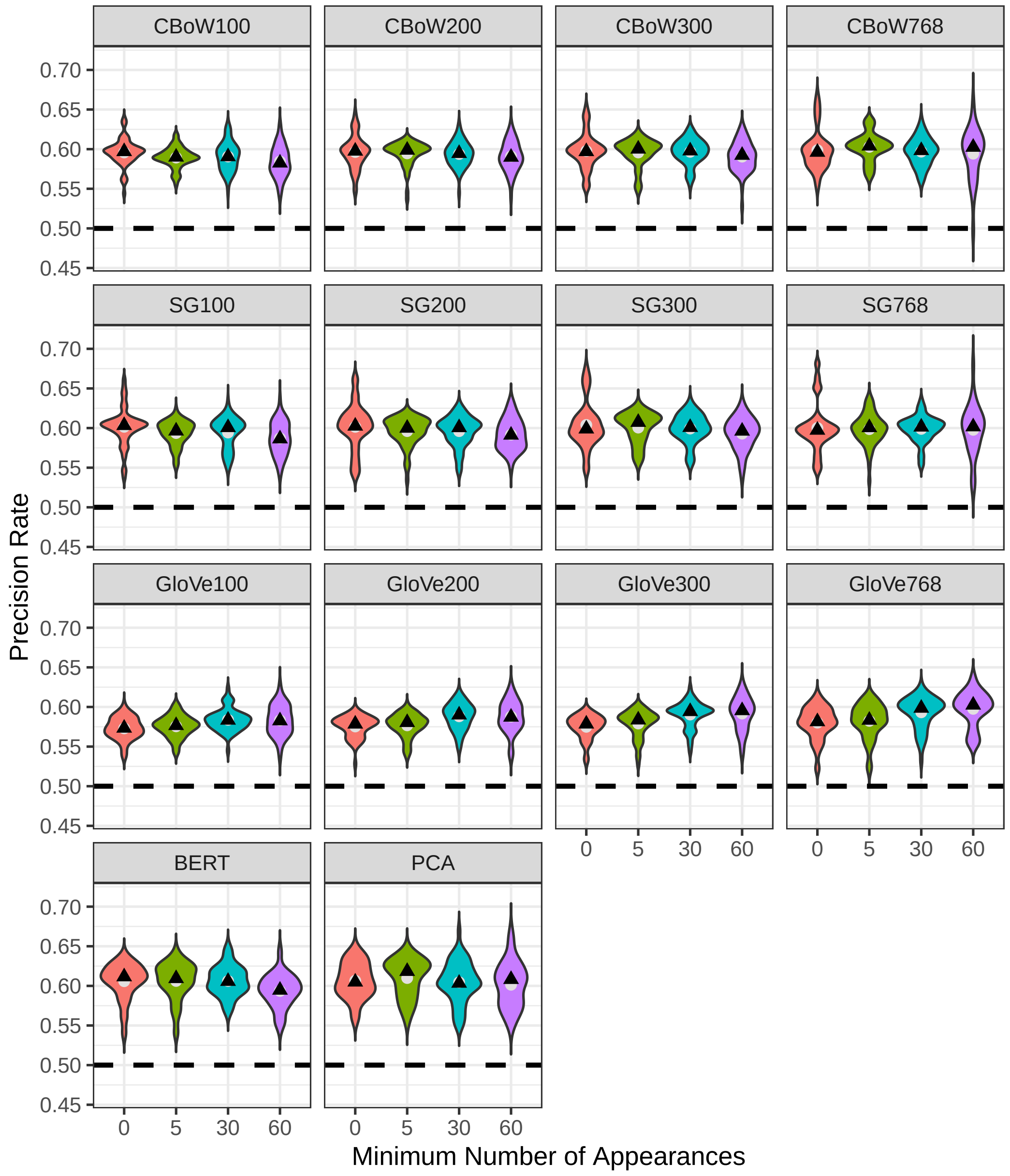

Figure A.19: $E \times F$, F-ratio $=$ 15.85. The effect of filtering threshold by embedding method on precision. The median is a black triangle, and the mean is a grey circle.

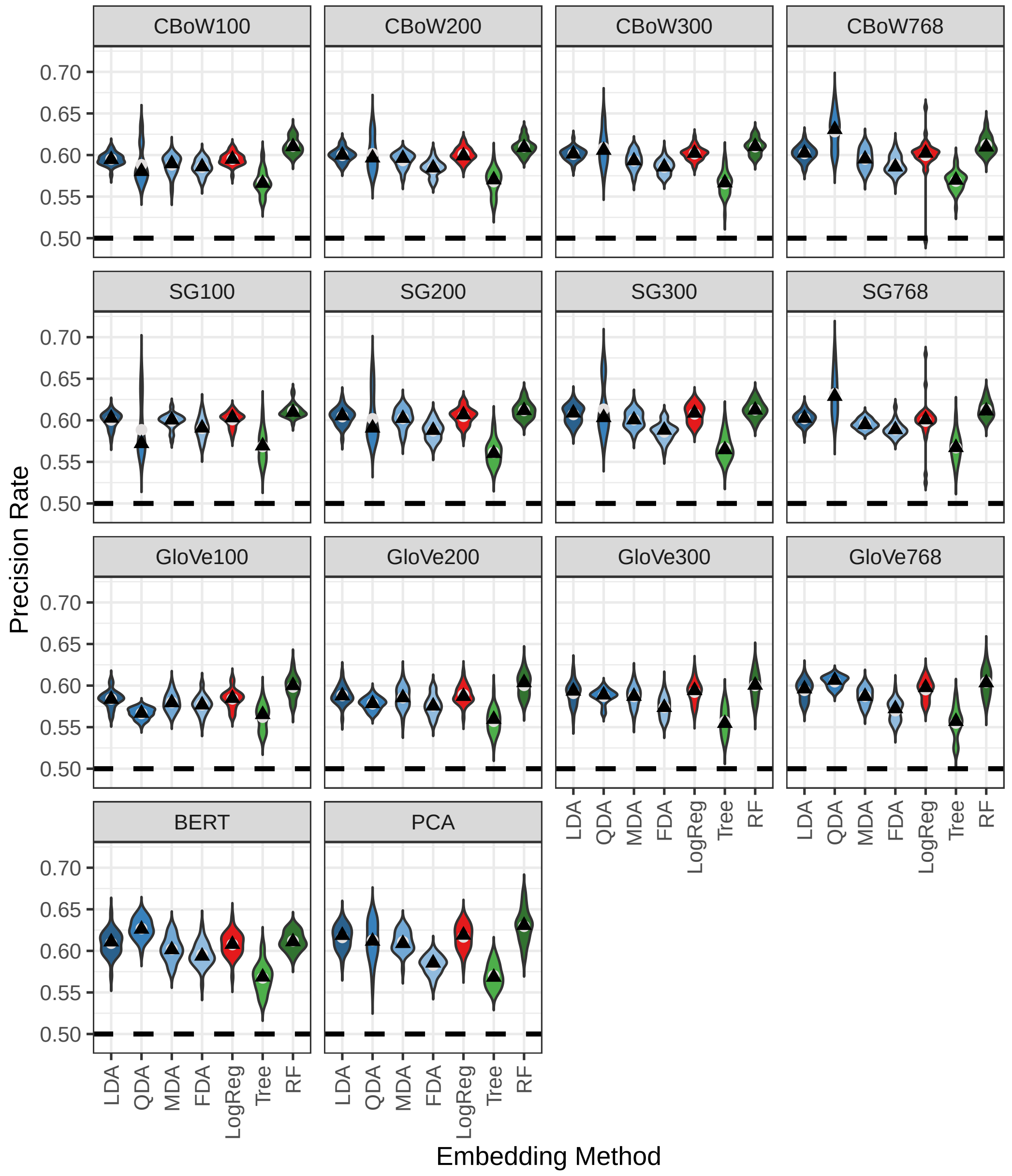

Figure A.20: $C \times E$, F-ratio = 13.202. The effect of classification method by embedding method on precision. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

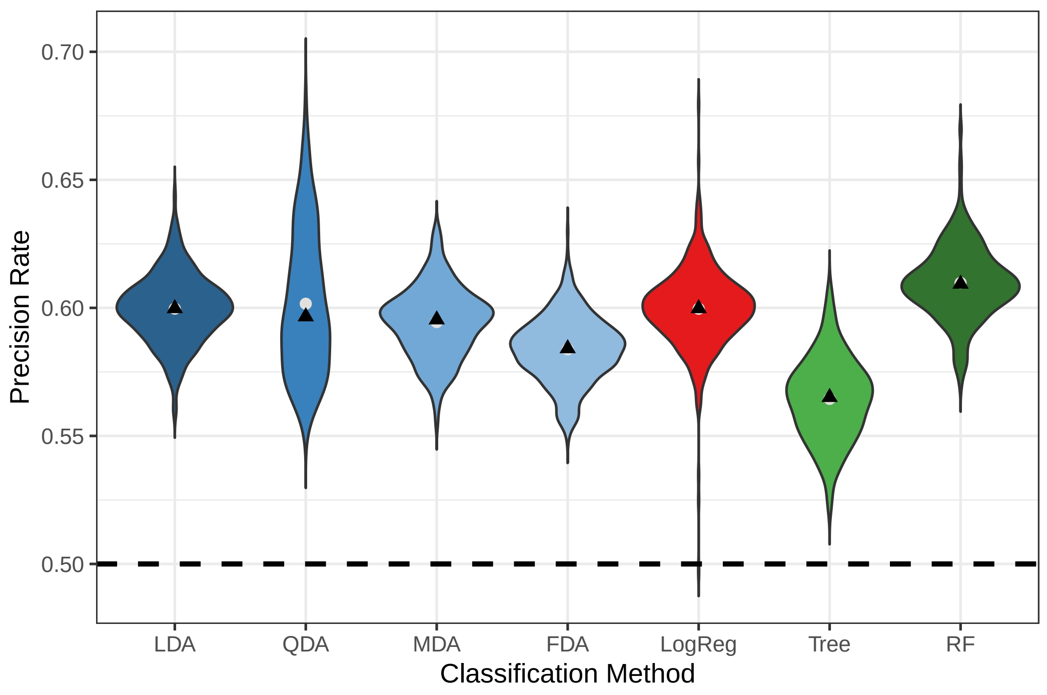

Figure A.21: C, F-ratio = 1126.108. The effect of classification method on precision. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

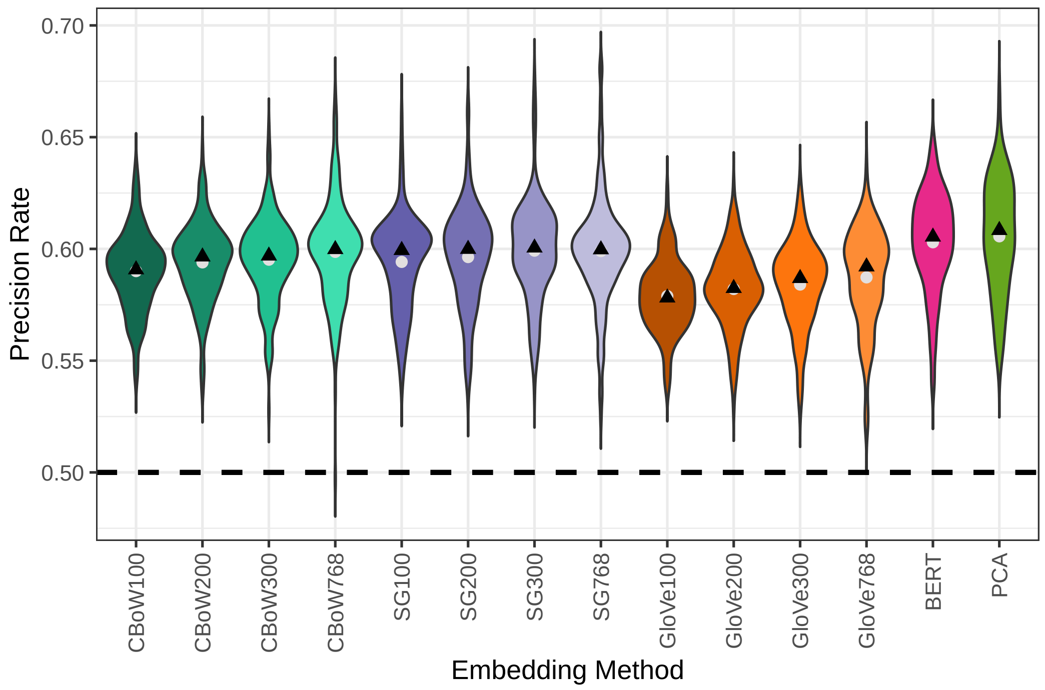

Figure A.22: E, F-ratio = 156.542. The effect of embedding method on precision. Word2Vec Continuous Bag of Words are in the blue-green shades, and Word2Vec Skip-Gram are in the blue-purple shades. GloVe is in the brown-orange shades. BERT is in pink, and PCA is in green. The median is a black triangle, and the mean is a grey circle.

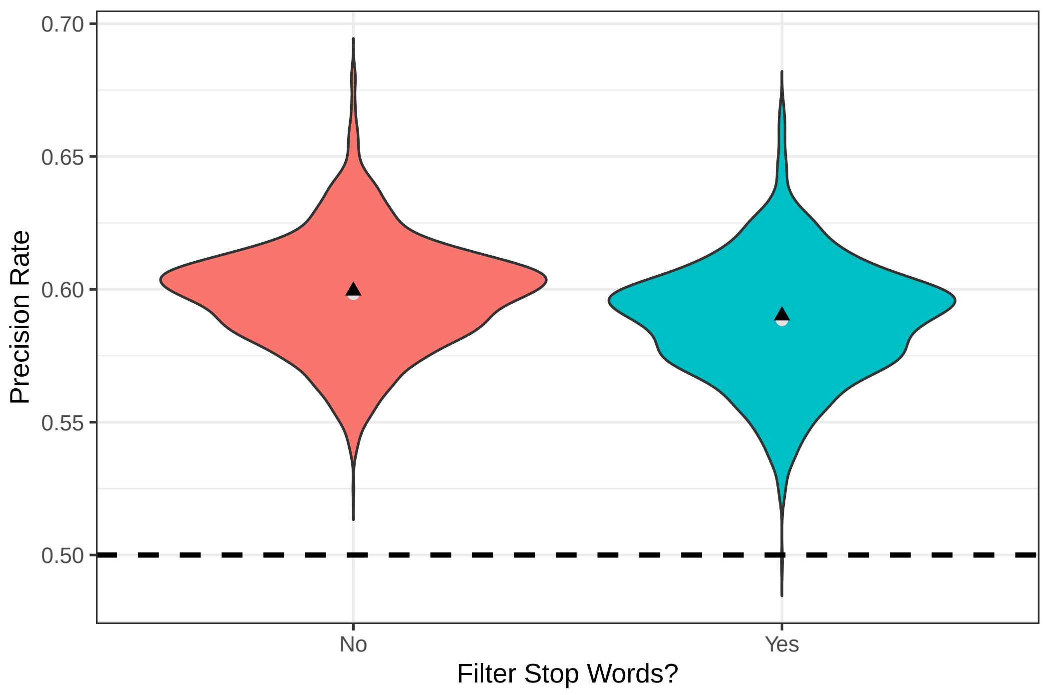

Figure A.23: S, F-ratio = 835.244. The effect of filtering stop words on precision. The median is a black triangle, and the mean is a grey circle.

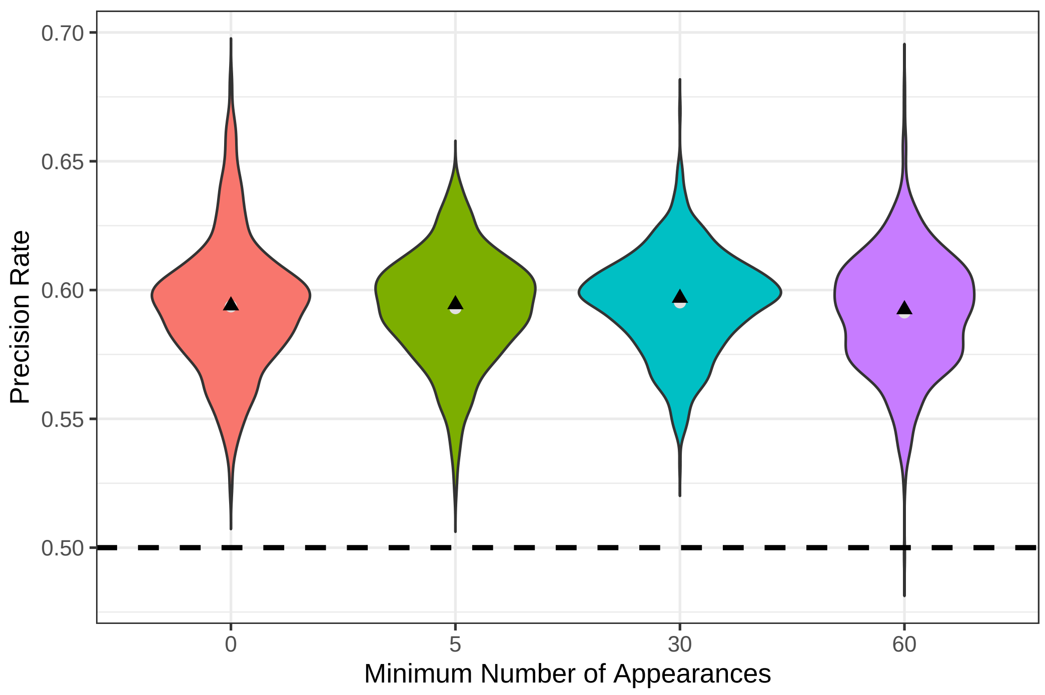

Figure A.24: F, F-ratio = 17.048. The effect of filtering threshold on precision. The median is a black triangle, and the mean is a grey circle.

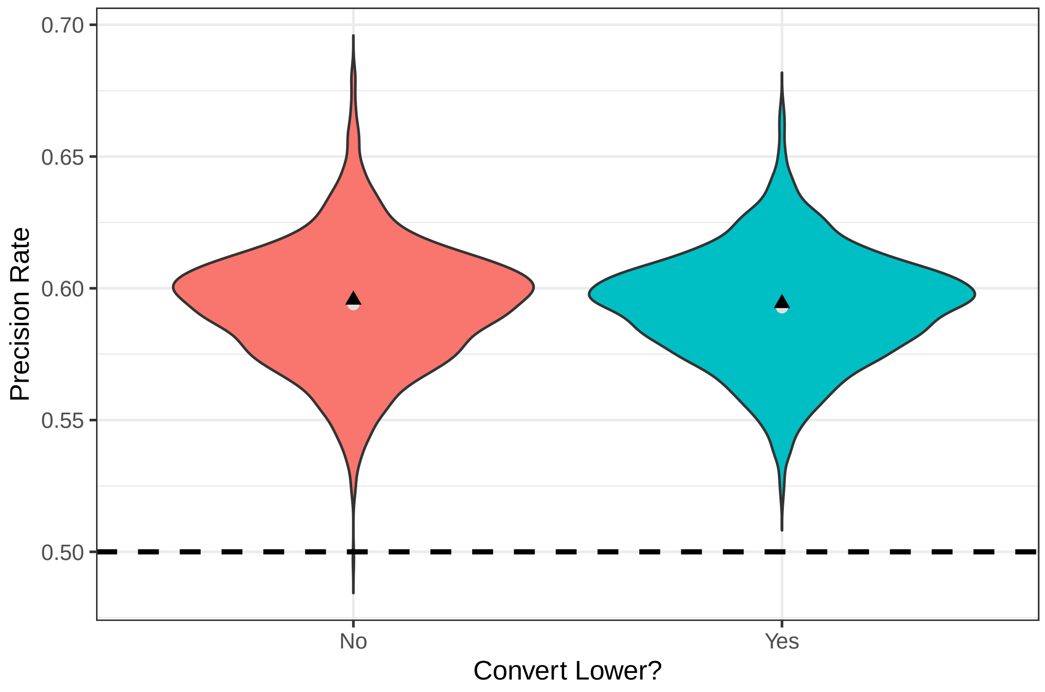

Figure A.25: L, F-ratio = 12.68. The effect of converting to lowercase on precision. The median is a black triangle, and the mean is a grey circle.

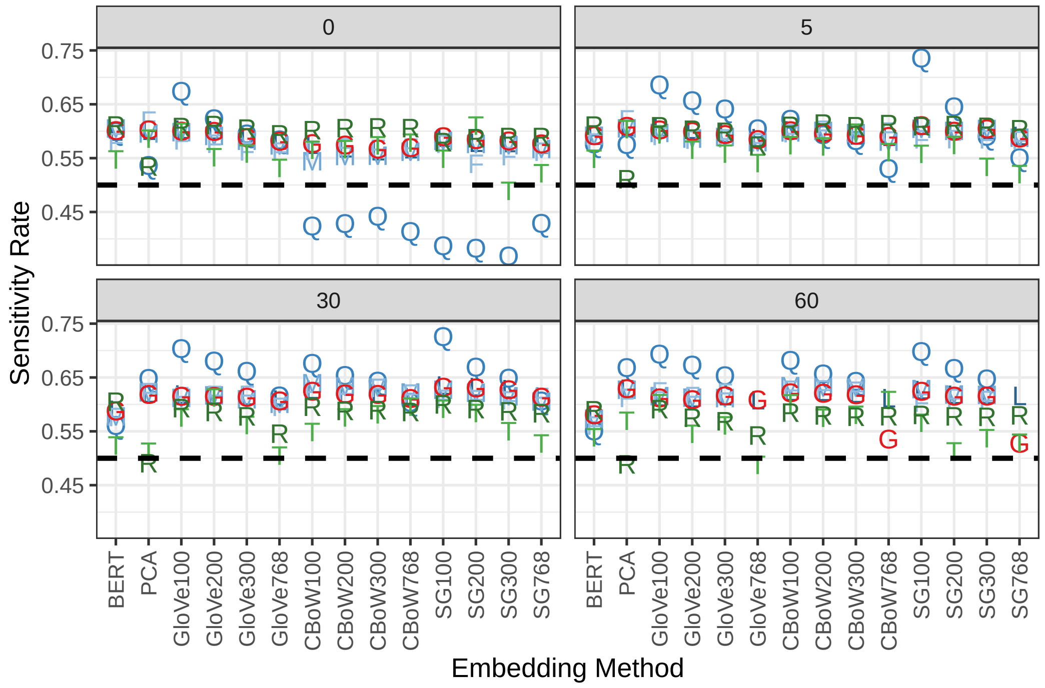

Figure A.26: $C \times E \times F$, F-ratio = 3.598. The effect of classification method, embedding method, and filtering threshold on sensitivity. Each classification method is represented by a letter. T is for classification tree, and G is for logistic regression. All others are based on the first letter of the method name. Only the means are plotted here. Violin plots are not used due to the number of levels of filtering threshold.

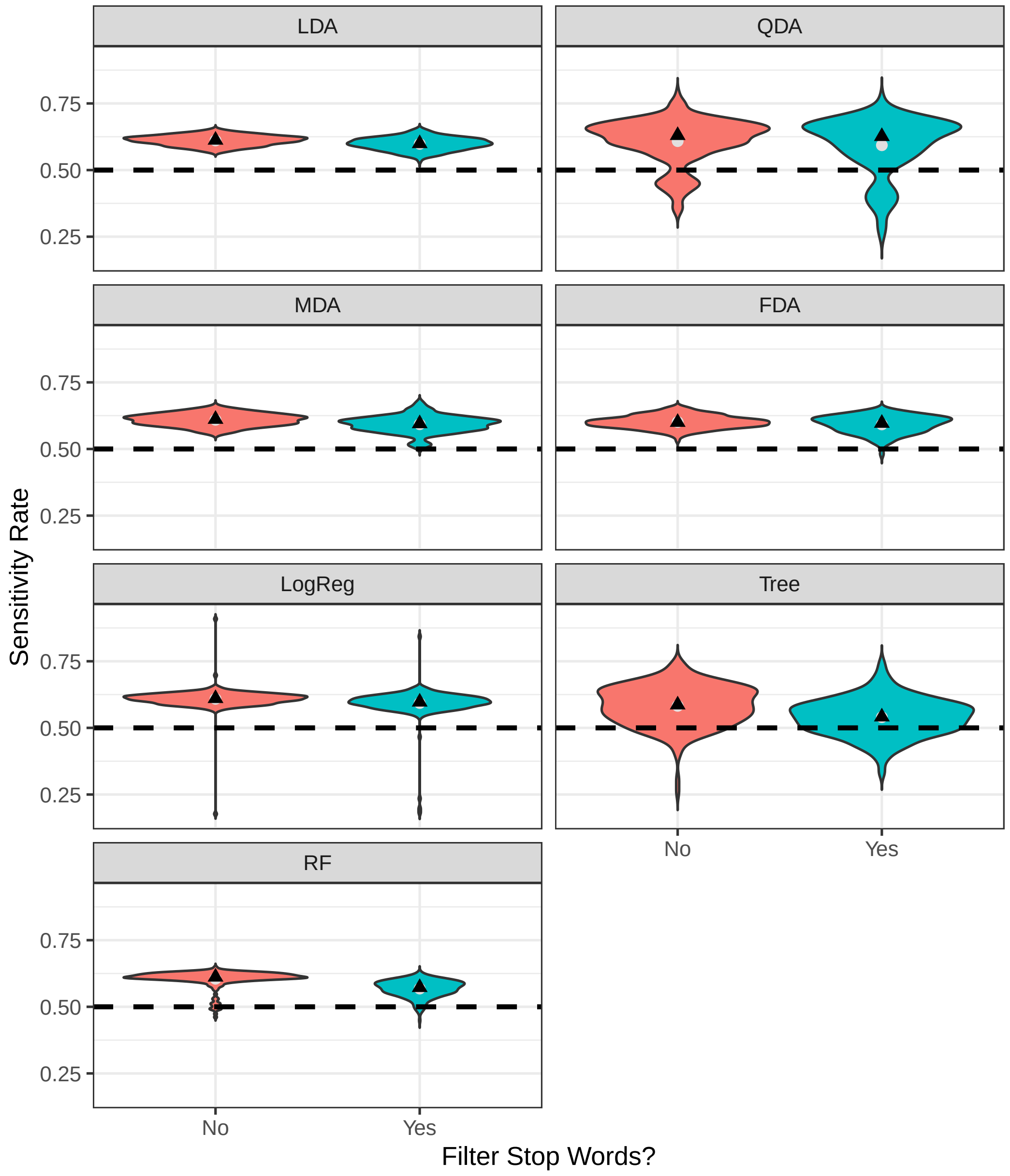

Figure A.27: $C \times S$, F-ratio = 15.512. The effect of filtering stop words by classification method on sensitivity. The median is a black triangle, and the mean is a grey circle.

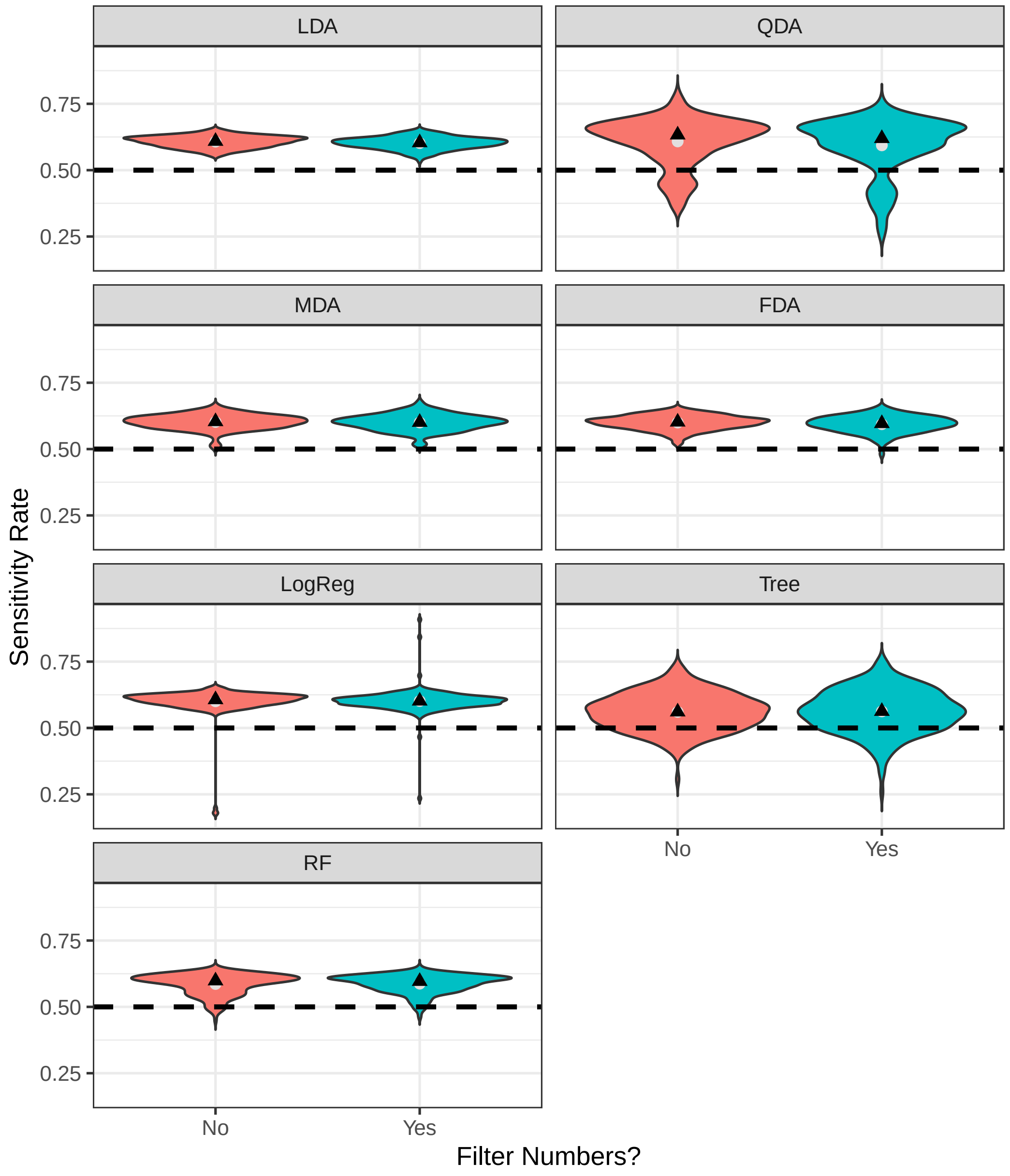

Figure A.28: $C \times N$, F-ratio = 3.149. The effect of filtering numbers by classification method on sensitivity. The median is a black triangle, and the mean is a grey circle.

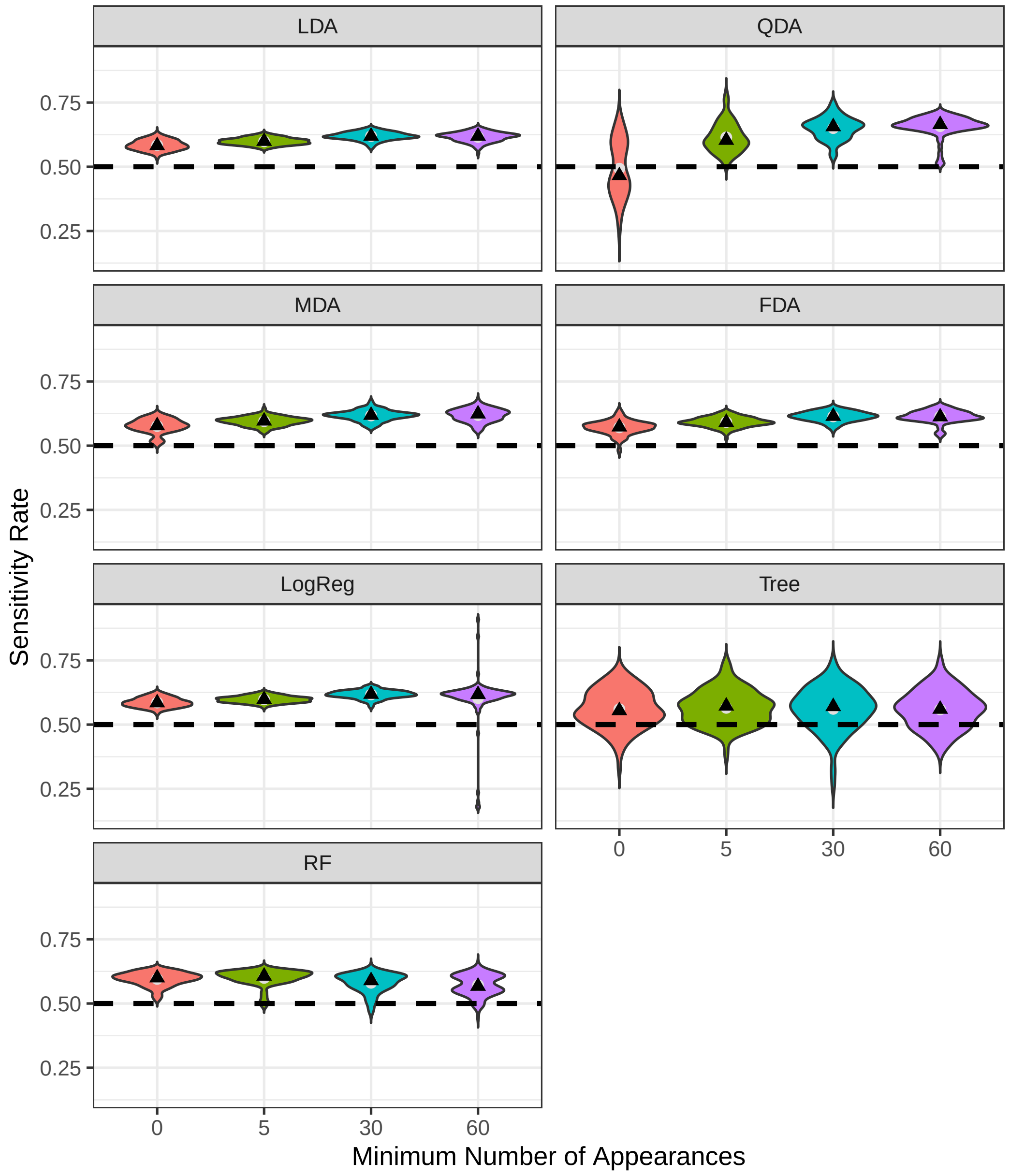

Figure A.29: $C \times F$, F-ratio = 65.52. The effect of filtering threshold by classification method on sensitivity. The median is a black triangle, and the mean is a grey circle.

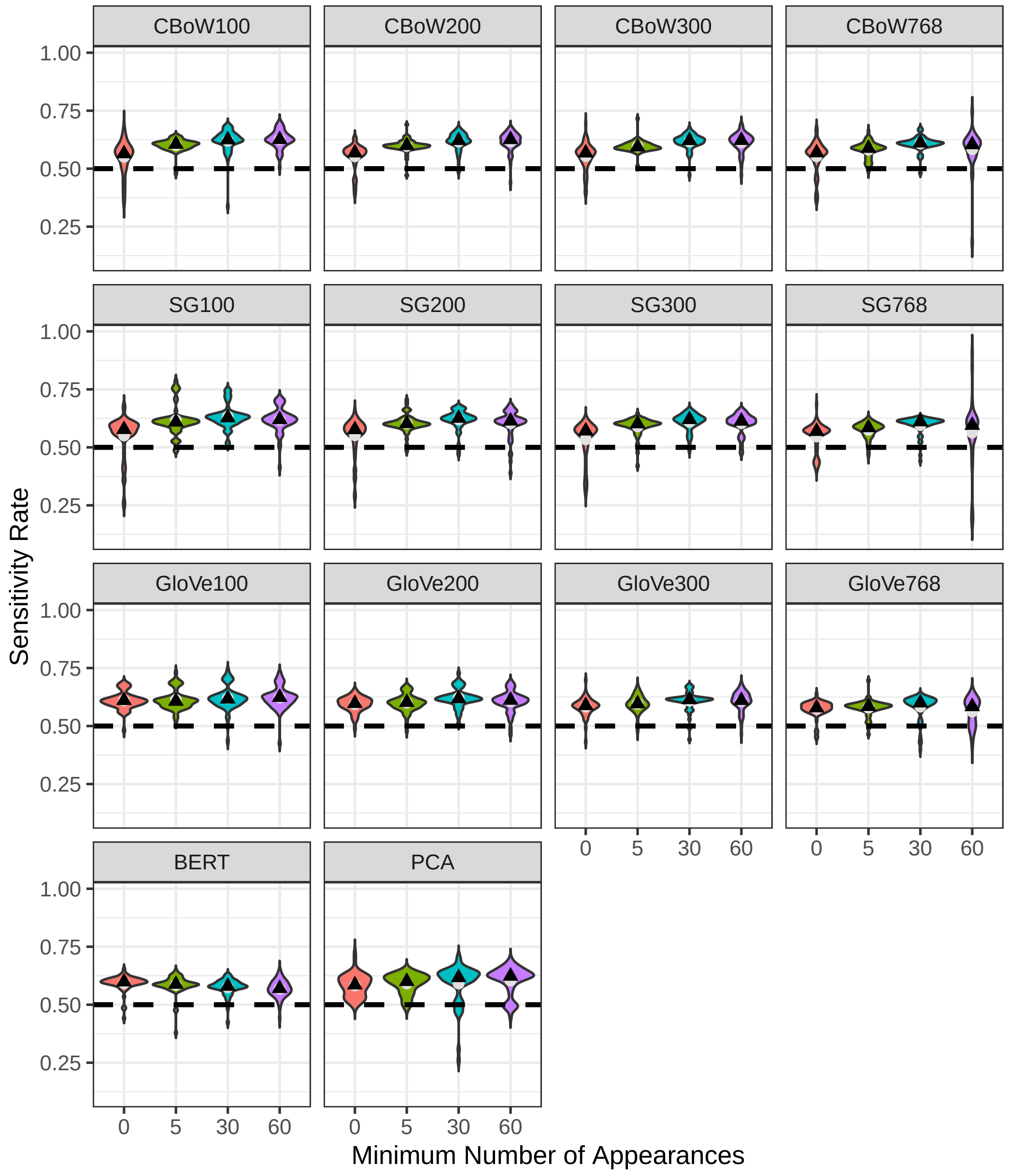

Figure A.30: $E \times F$, F-ratio = 10.2588. The effect of filtering threshold by embedding method on sensitivity. The median is a black triangle, and the mean is a grey circle.

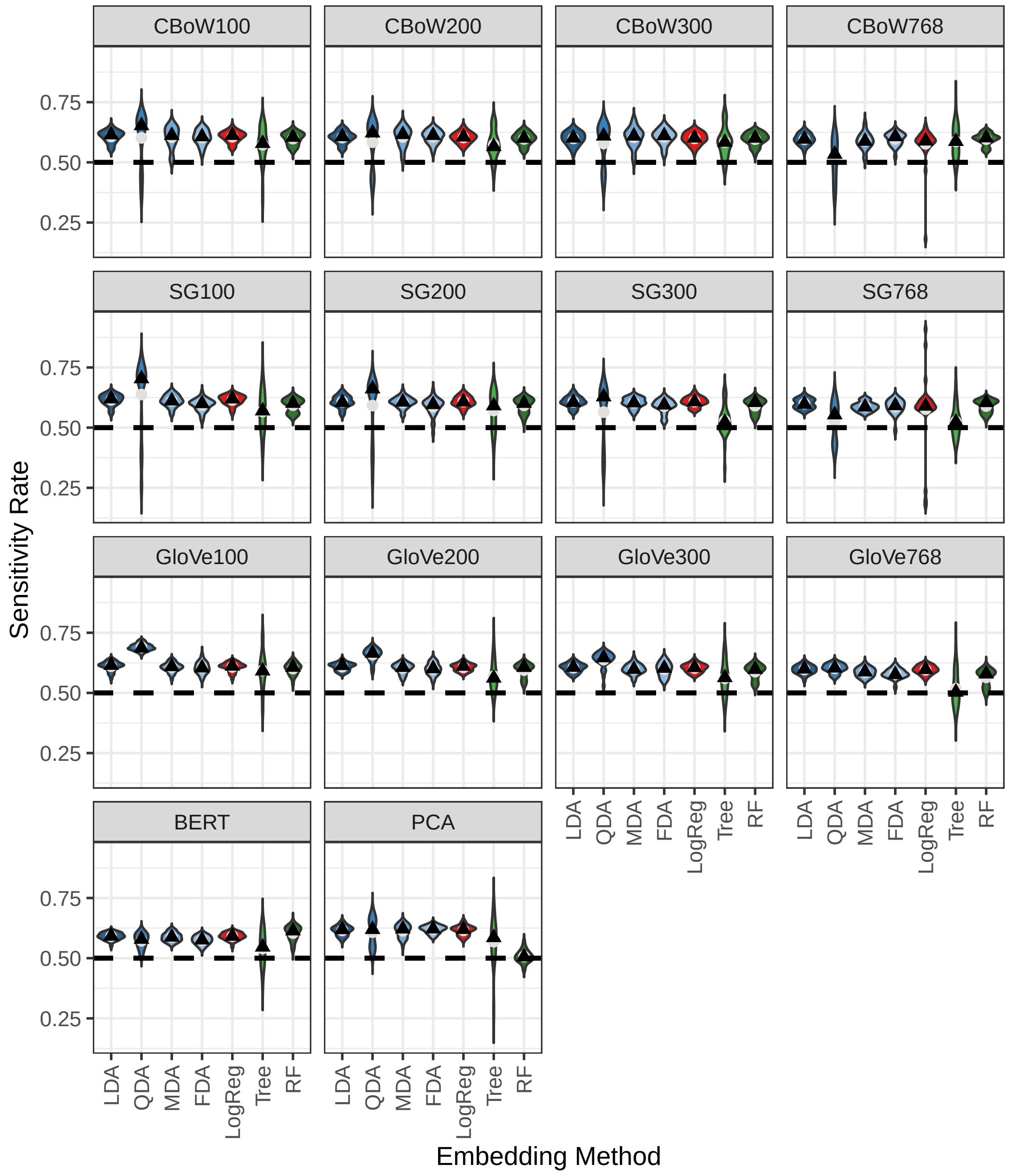

Figure A.31: $C \times E$, F-ratio = 10.811. The effect of classification method by embedding method on sensitivity. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

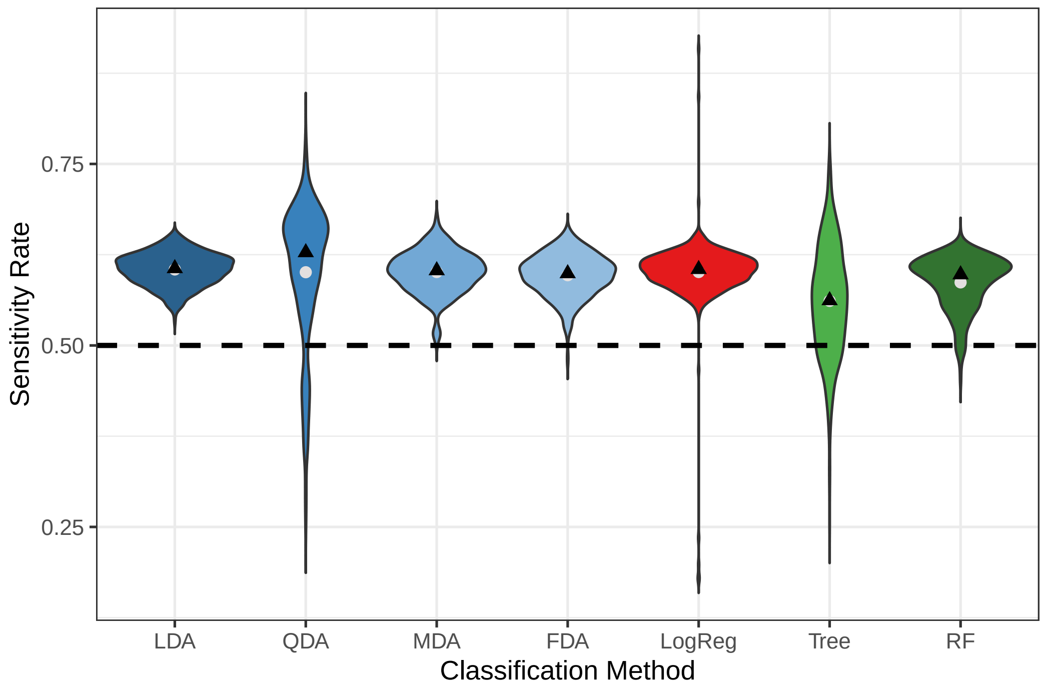

Figure A.32: C, F-ratio = 82.573. The effect of classification method on sensitivity. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

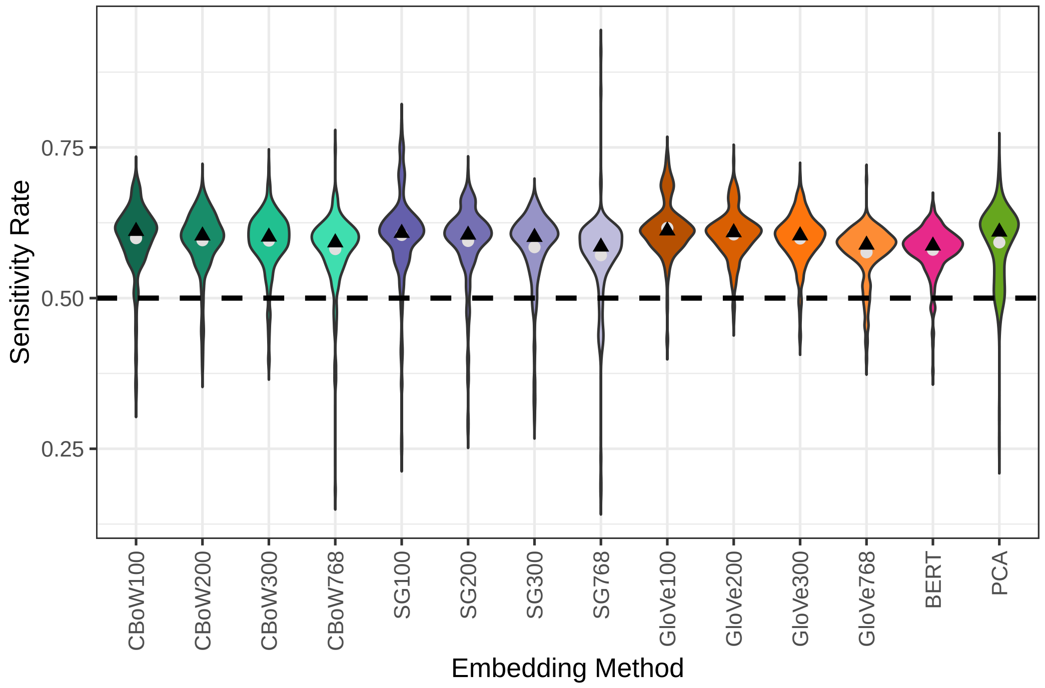

Figure A.33: E, F-ratio = 23.438. The effect of embedding method on sensitivity. Word2Vec Continuous Bag of Words are in the blue-green shades, and Word2Vec Skip-Gram are in the blue-purple shades. GloVe is in the brown-orange shades. BERT is in pink, and PCA is in green. The median is a black triangle, and the mean is a grey circle.

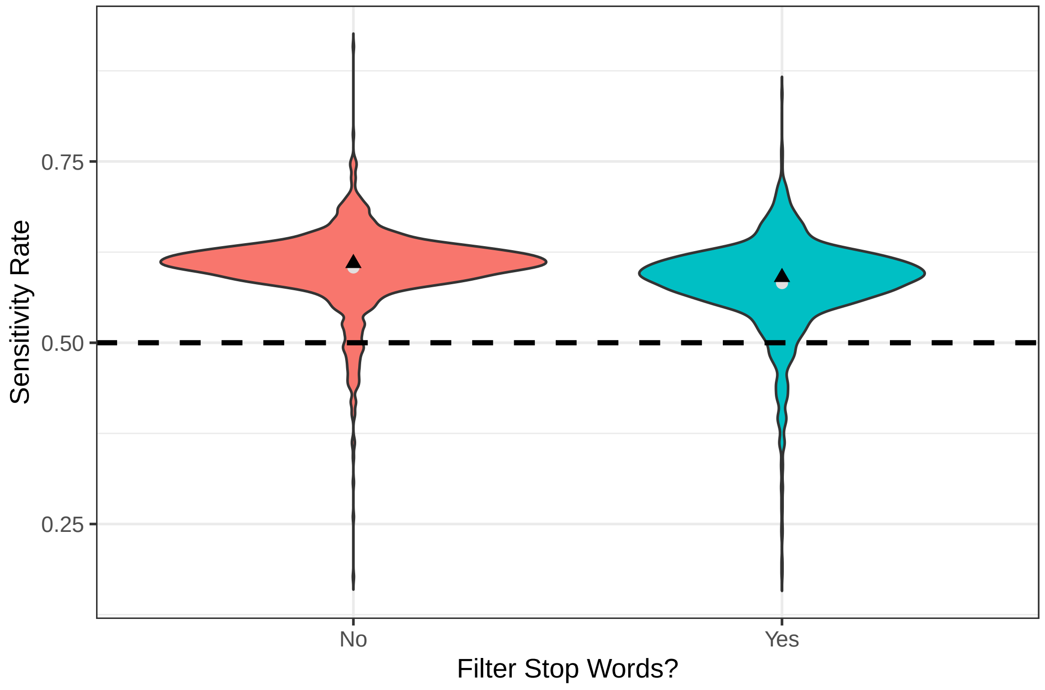

Figure A.34: S, F-ratio = 295.017. The effect of filtering stop words on sensitivity. The median is a black triangle, and the mean is a grey circle.

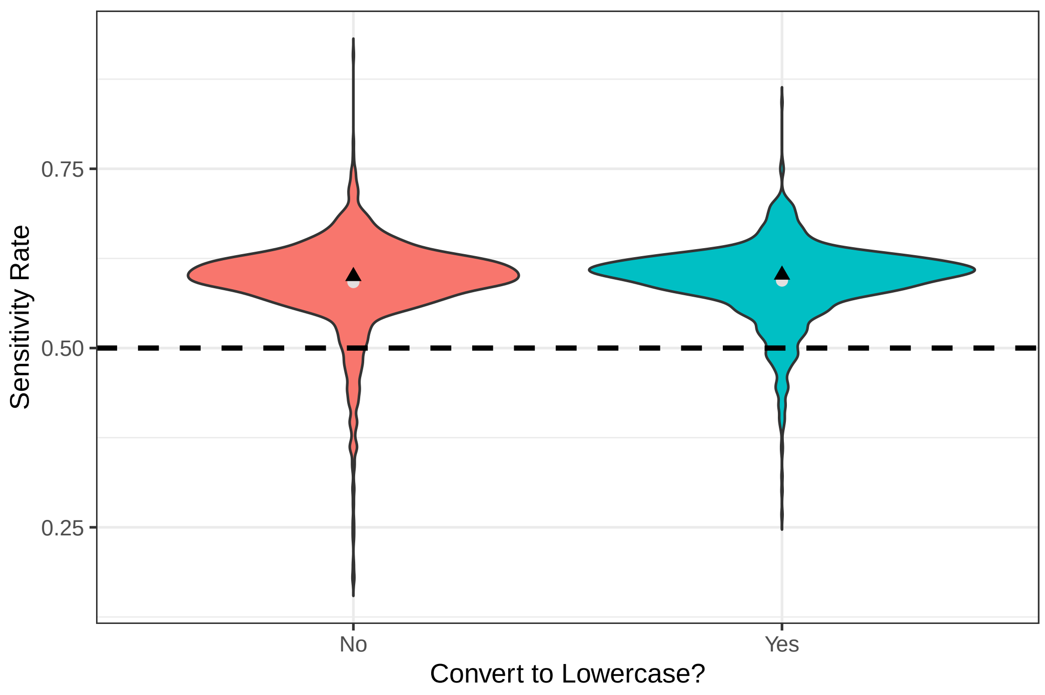

Figure A.35: L, F-ratio = 2.701. The effect of converting to lowercase on sensitivity. The median is a black triangle, and the mean is a grey circle.

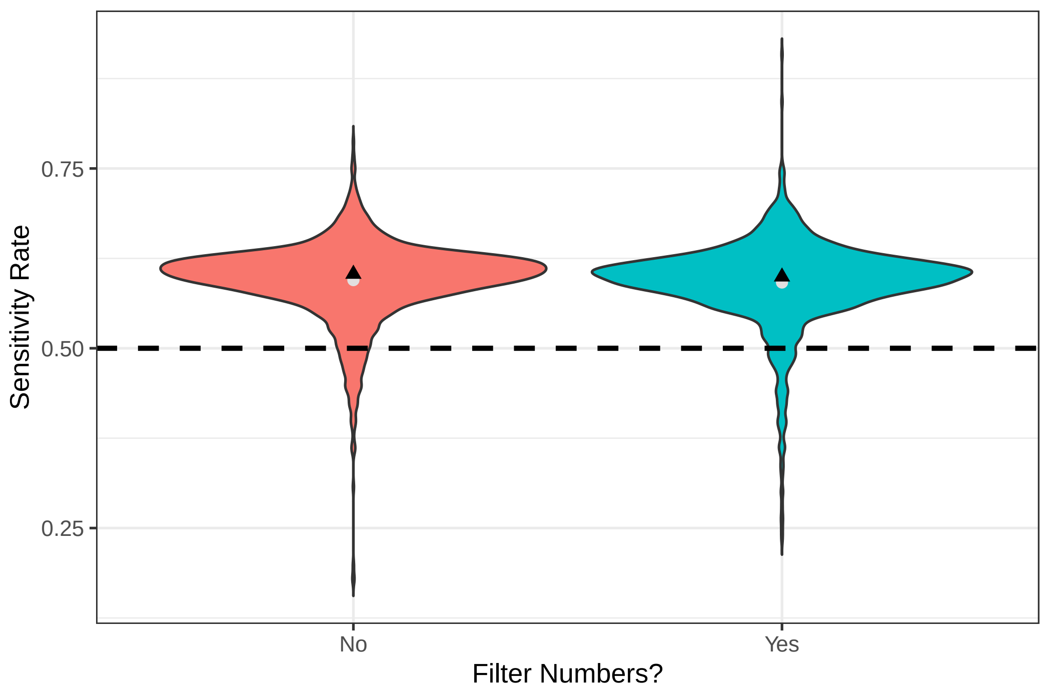

Figure A.36: N, F-ratio = 7.061. The effect of filtering numbers on sensitivity. The median is a black triangle, and the mean is a grey circle.

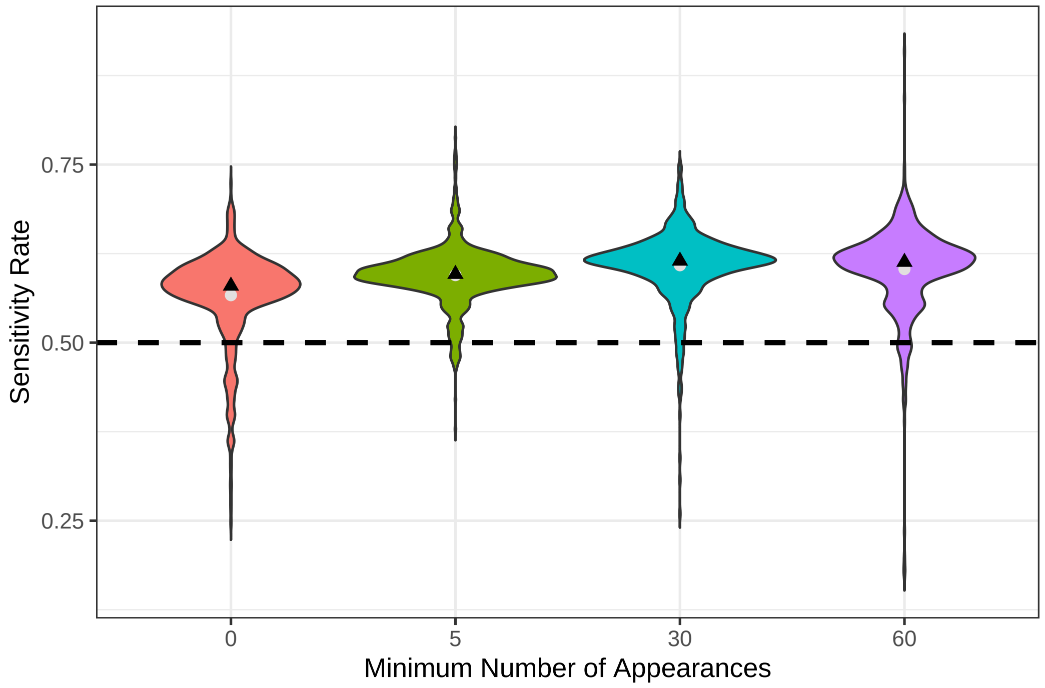

Figure A.37: F, F-ratio = 219.469. The effect of filtering threshold on sensitivity. The median is a black triangle, and the mean is a grey circle.

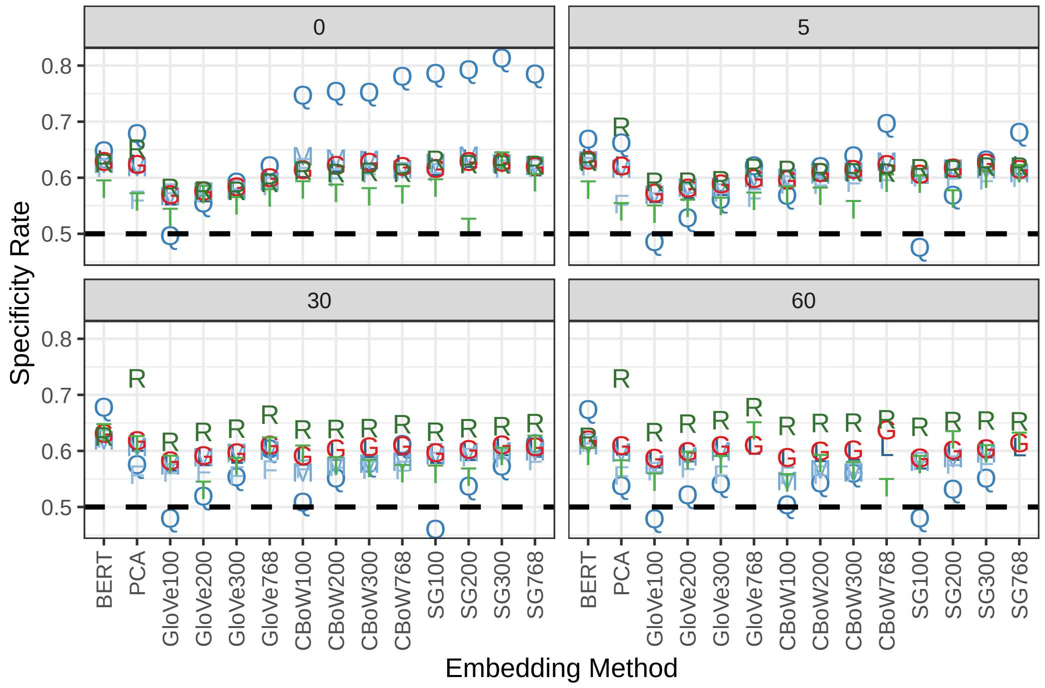

Figure A.38: $C \times E \times F$, F-ratio = 3.492. The effect of classification method, embedding method, and filtering threshold on specificity. Each classification method is represented by a letter. T is for classification tree, and G is for logistic regression. All others are based on the first letter of the method name. Only the means are plotted here. Violin plots are not used due to the number of levels of filtering threshold.

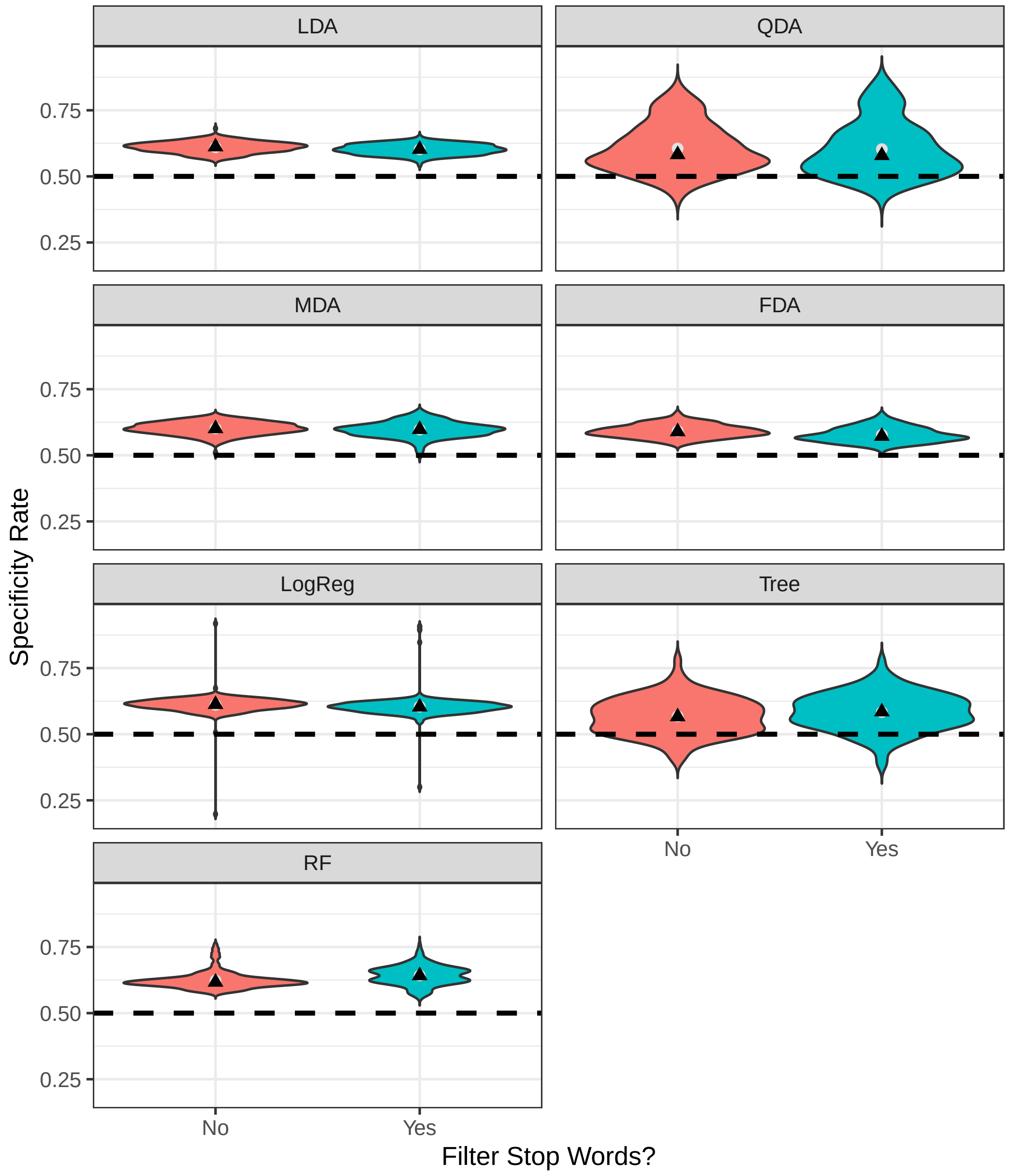

Figure A.39: $C \times S$, F-ratio = 16.452. The effect of filtering stop words by classification method on specificity. The median is a black triangle, and the mean is a grey circle.

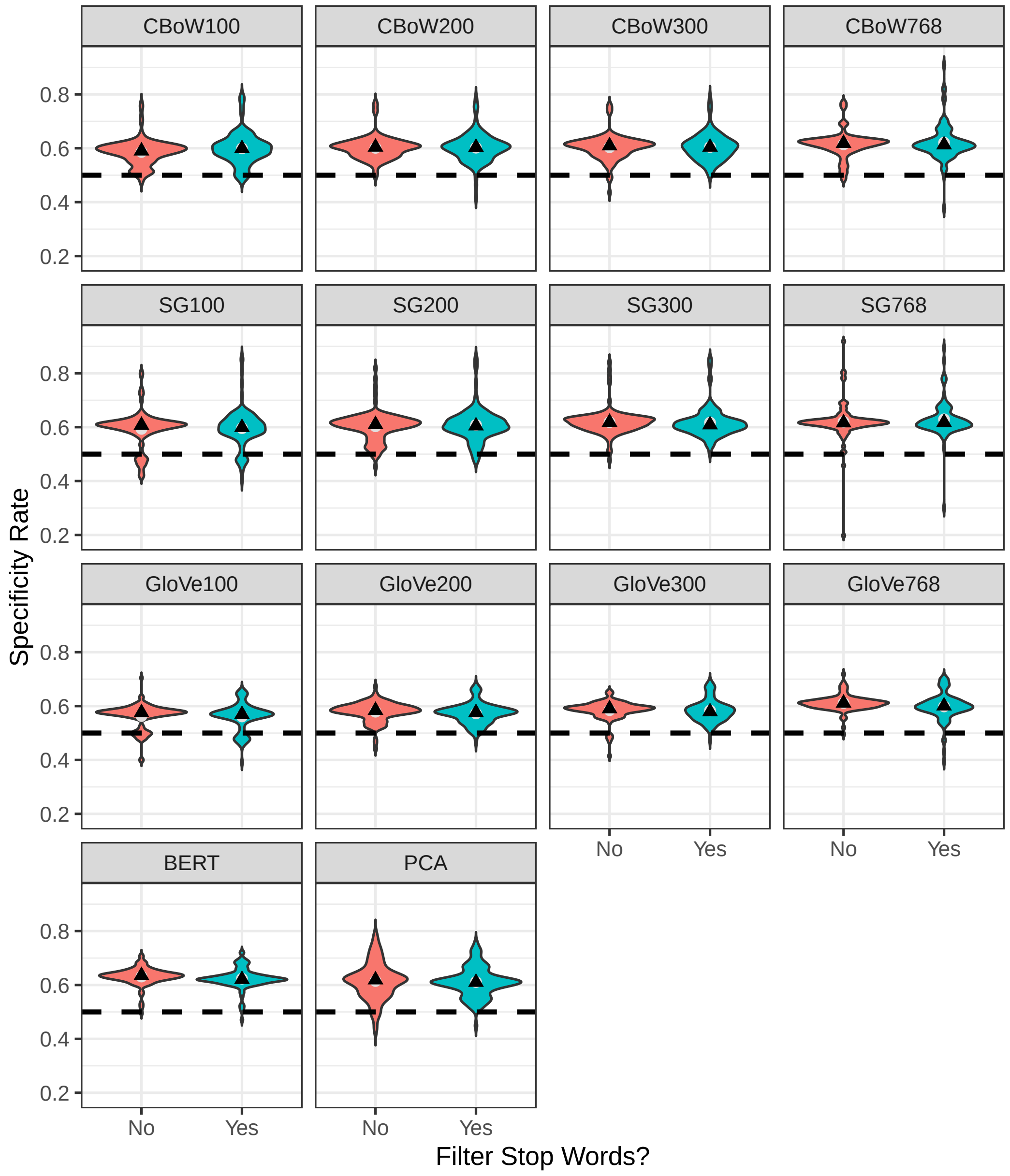

Figure A.40: $E \times S$, F-ratio = 2.347. The effect of filtering stop words by embedding method on specificity. The median is a black triangle, and the mean is a grey circle.

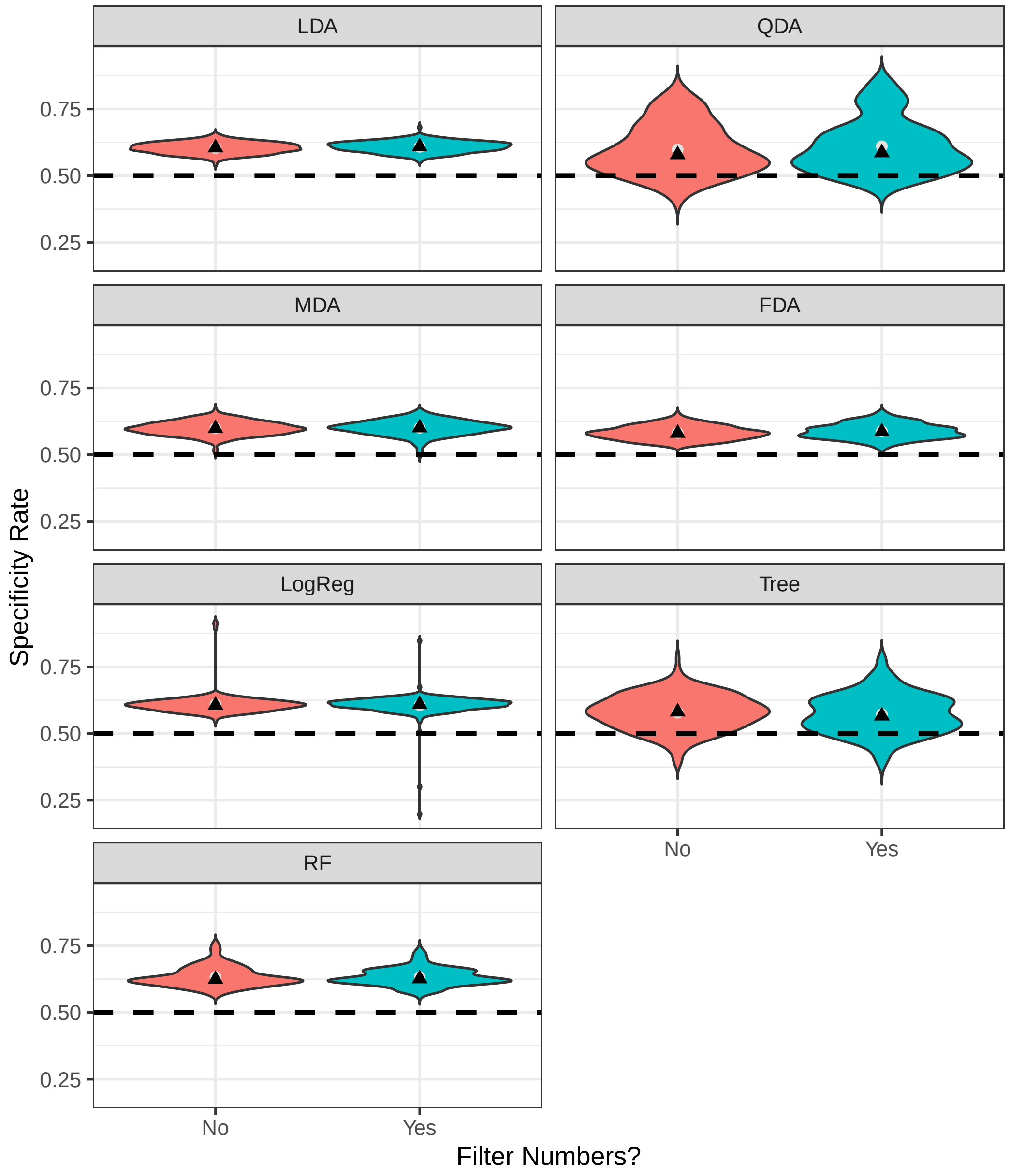

Figure A.41: $C \times N$, F-ratio = 3.67. The effect of filtering numbers by classification method on specificity. The median is a black triangle, and the mean is a grey circle.

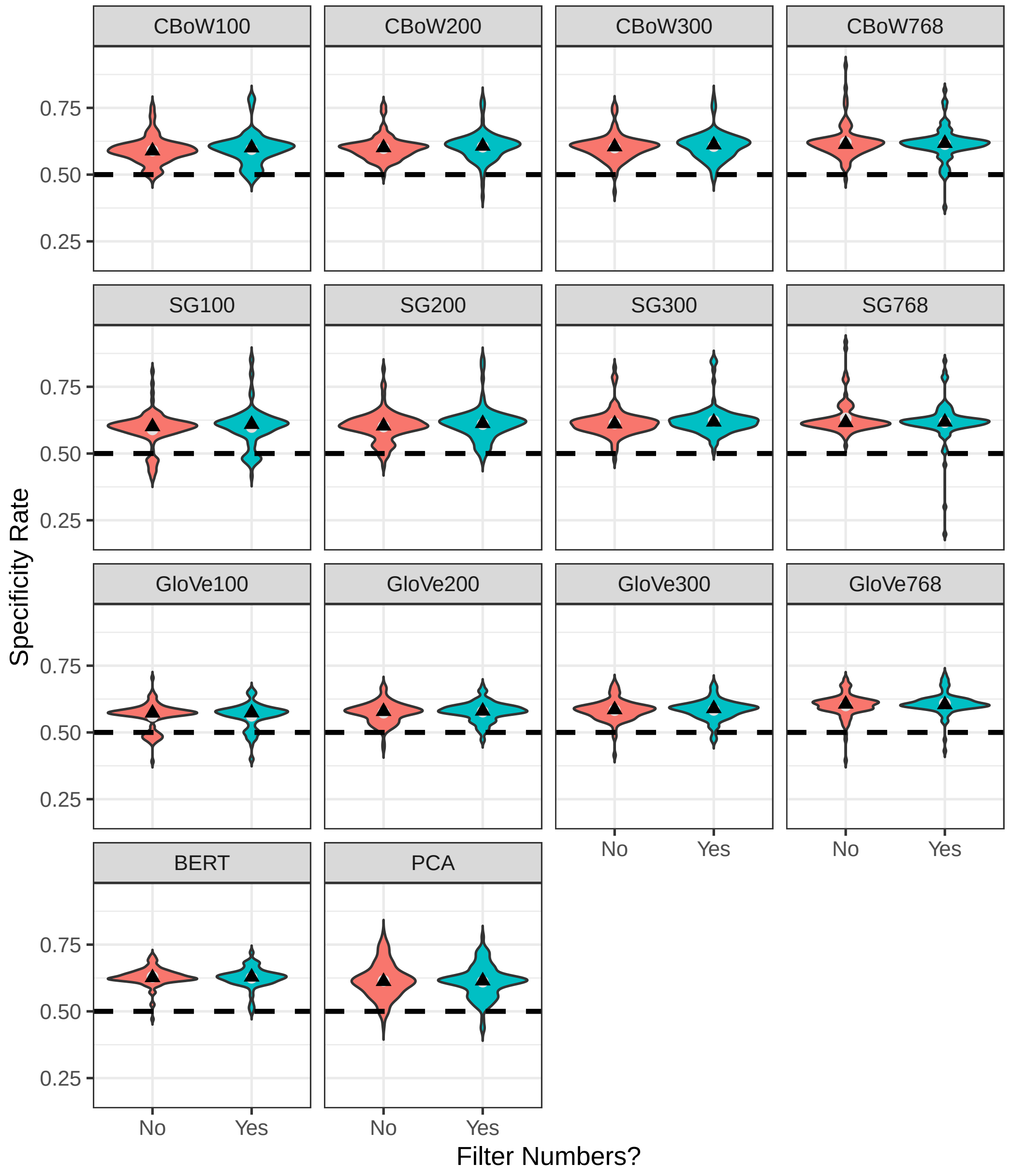

Figure A.42: $E \times N$, F-ratio = 2.173. The effect of filtering numbers by embedding method on specificity. The median is a black triangle, and the mean is a grey circle.

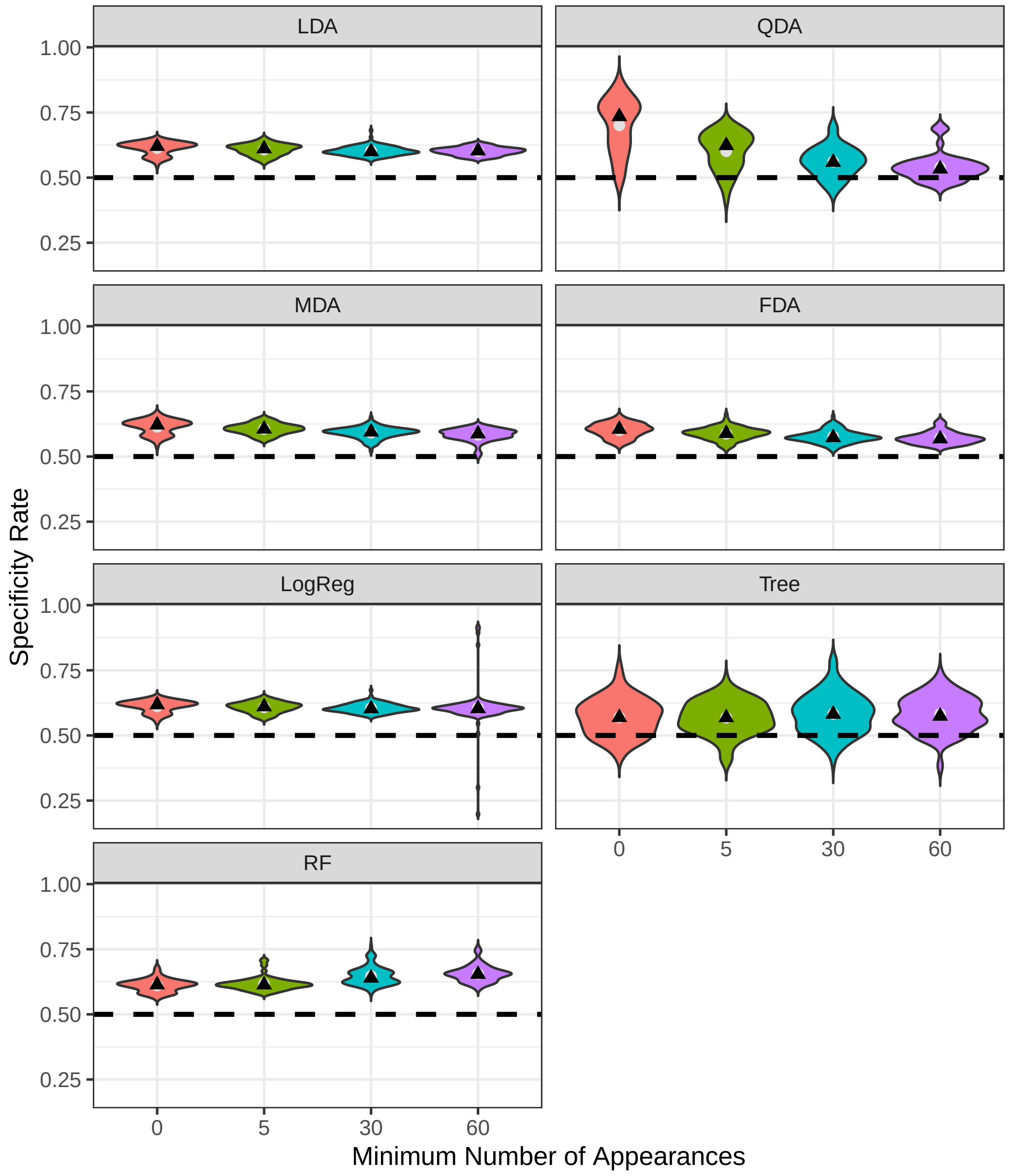

Figure A.43: $C \times F$, F-ratio = 76.906. The effect of filtering threshold by classification method on specificity. The median is a black triangle, and the mean is a grey circle.

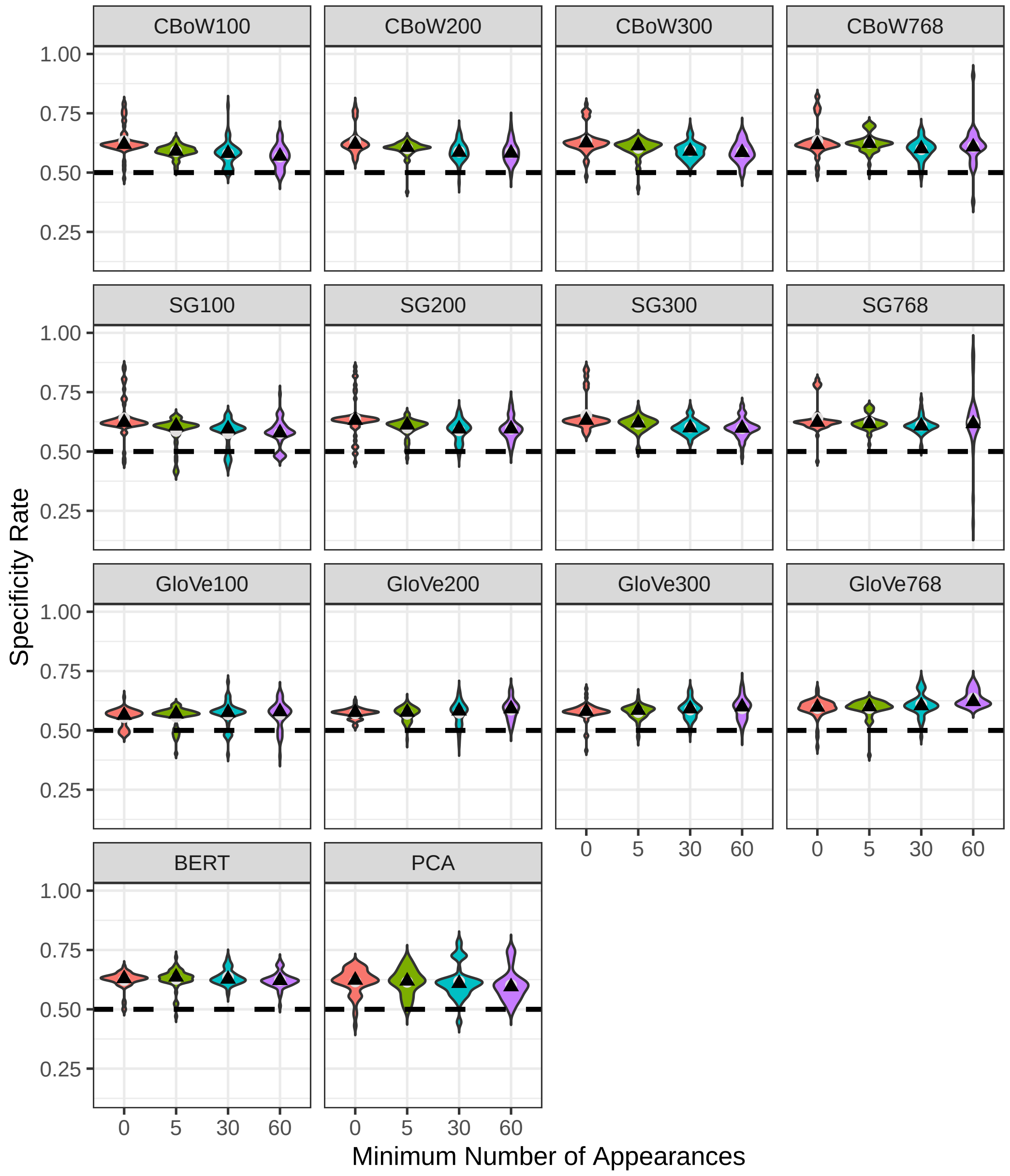

Figure A.44: $E \times F$, F-ratio = 11.126. The effect of filtering threshold by embedding method on specificity. The median is a black triangle, and the mean is a grey circle.

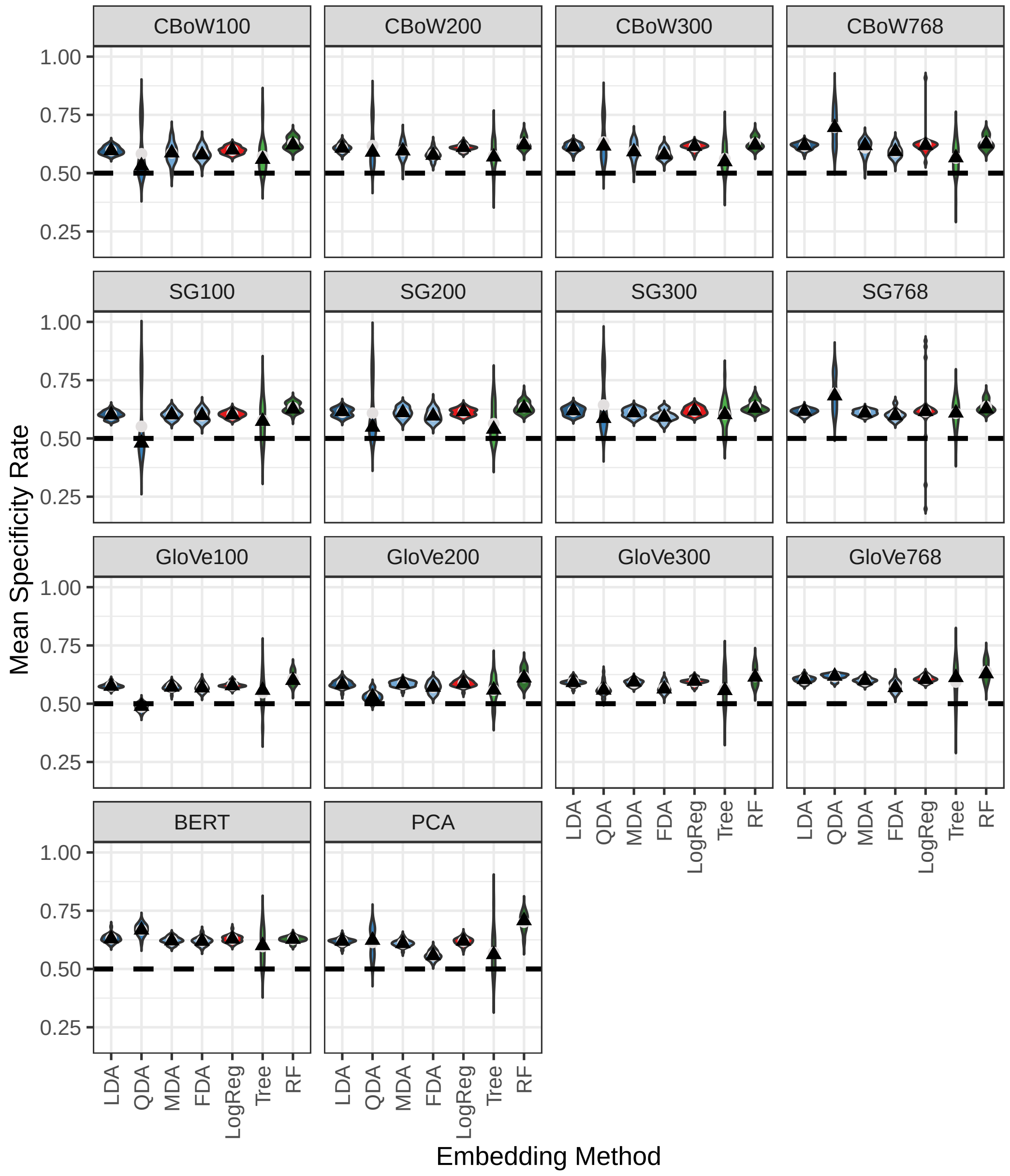

Figure A.45: $C \times E$, F-ratio = 11.752. The effect of embedding method by classification method on specificity. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

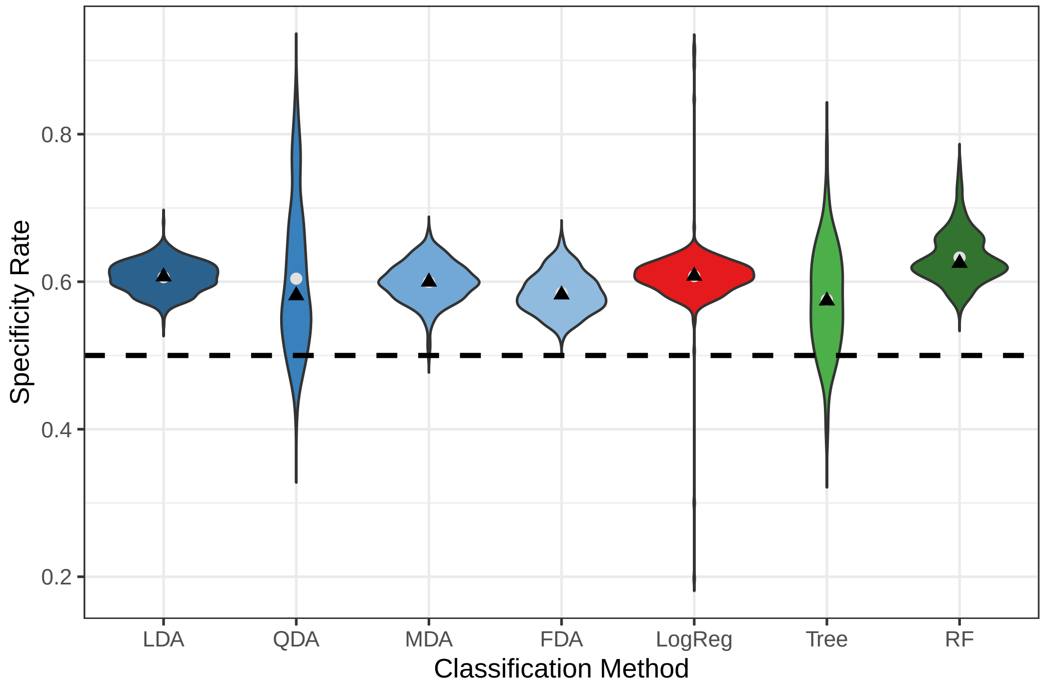

Figure A.46: C, F-ratio = 136.591. The effect of classification method on specificity. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

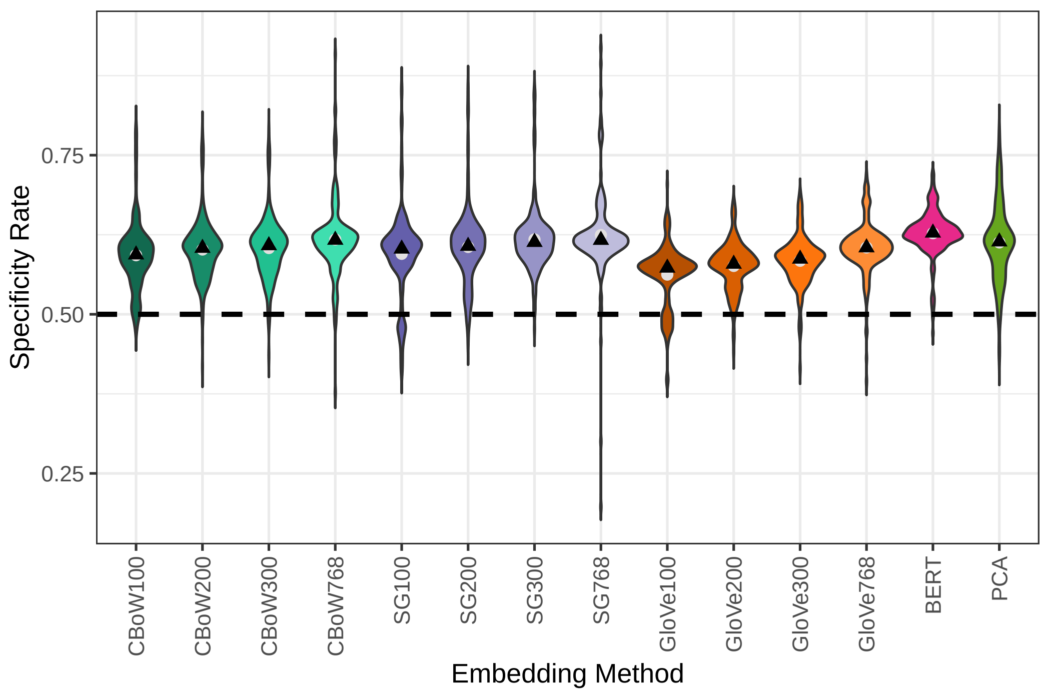

Figure A.47: E, F-ratio = 66.197. The effect of embedding method on specificity. Word2Vec Continuous Bag of Words are in the blue-green shades, and Word2Vec Skip-Gram are in the blue-purple shades. GloVe is in the brown-orange shades. BERT is in pink, and PCA is in green. The median is a black triangle, and the mean is a grey circle.

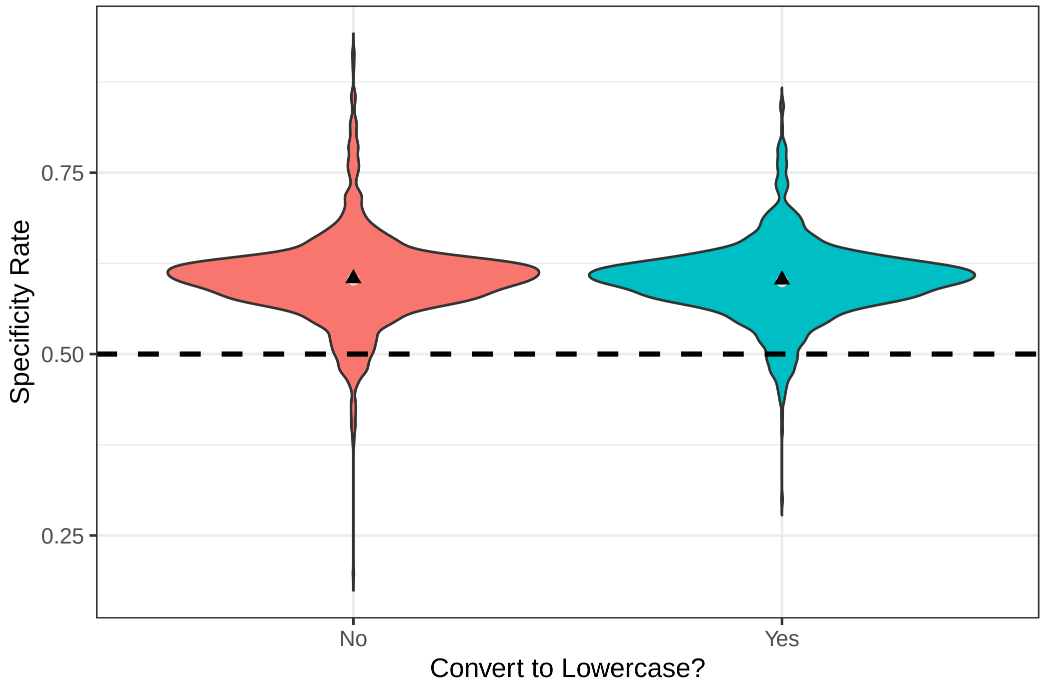

Figure A.48: L, F-ratio = 3.506. The effect of converting to lowercase on specificity. The median is a black triangle, and the mean is a grey circle.

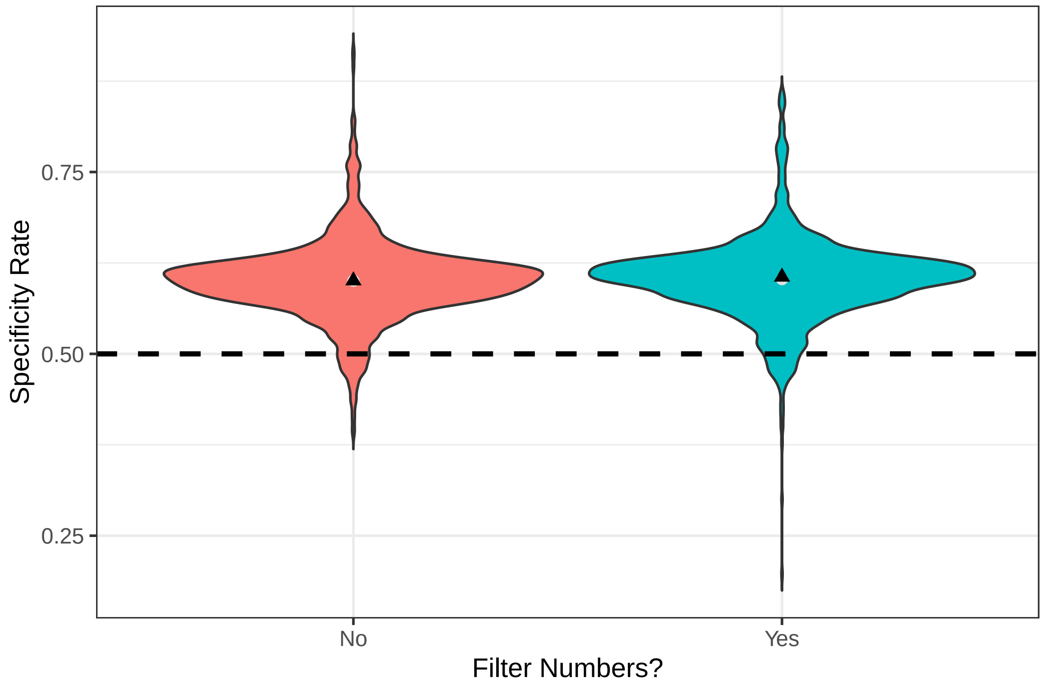

Figure A.49: N, F-ratio = 3.31. The effect of filtering numbers on specificity. The median is a black triangle, and the mean is a grey circle.

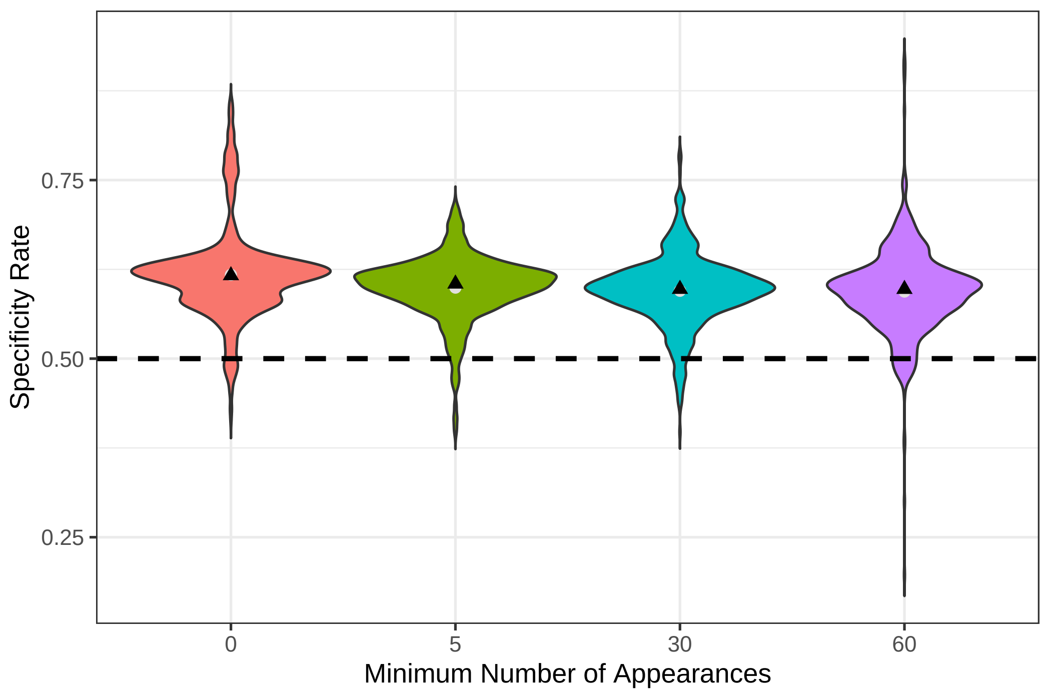

Figure A.50: F, F-ratio = 79.932. The effect of filtering threshold on specificity. The median is a black triangle, and the mean is a grey circle.

Appendix B

Plots for Chapter 7

This appendix contains all interaction and main effect mean plots discussed in Chapter 7 for effects with a F-ratio greater than 2. It also contains appropriate violin plots for the main effect and two-way interactions allowing us to visualize the distribution of observations.

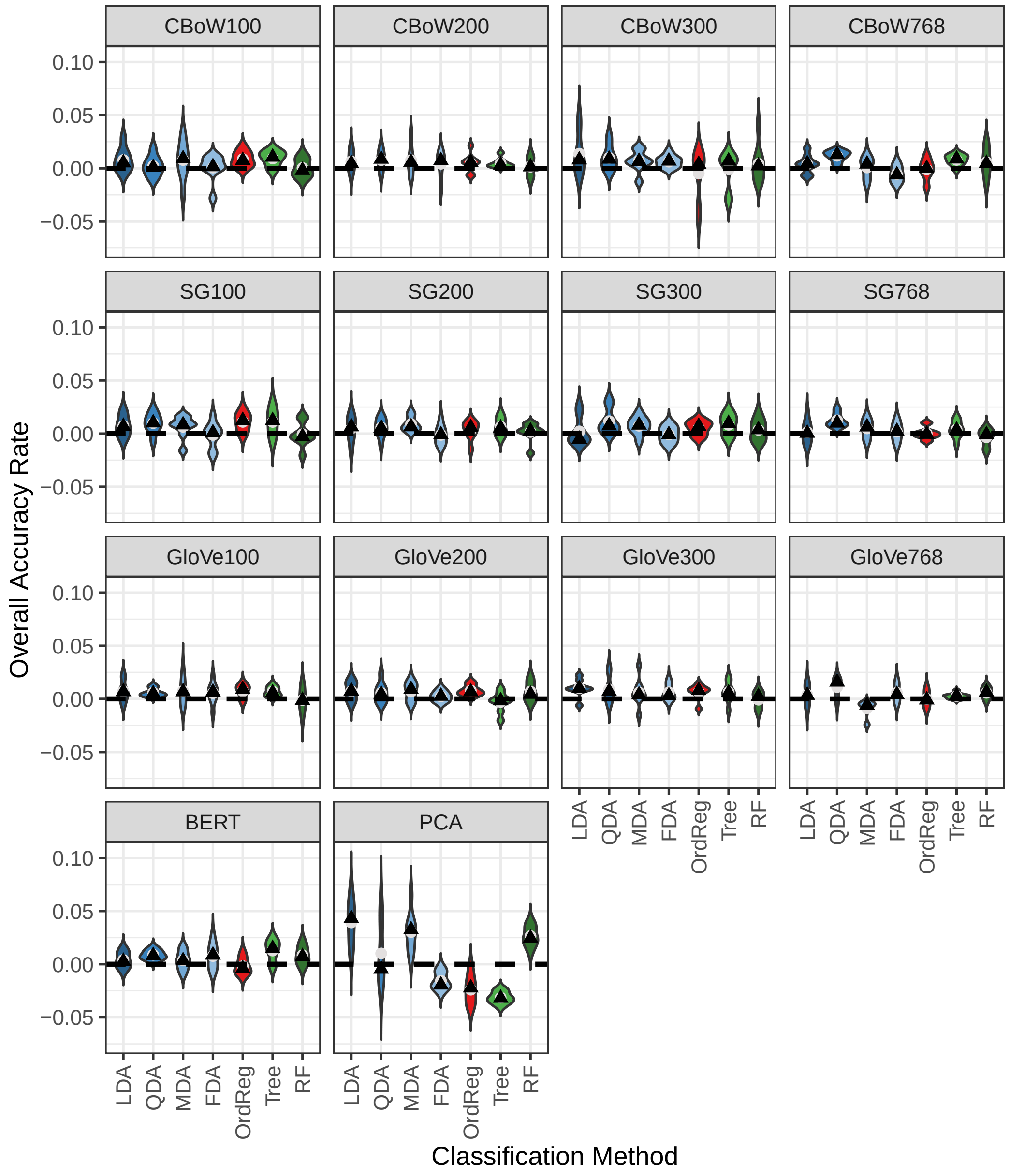

Figure B.1: $C \times E \times S$, F-ratio = 2.401. The difference in overall accuracy between filtering of stop words (no-yes) for each combination of classification and embedding method. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

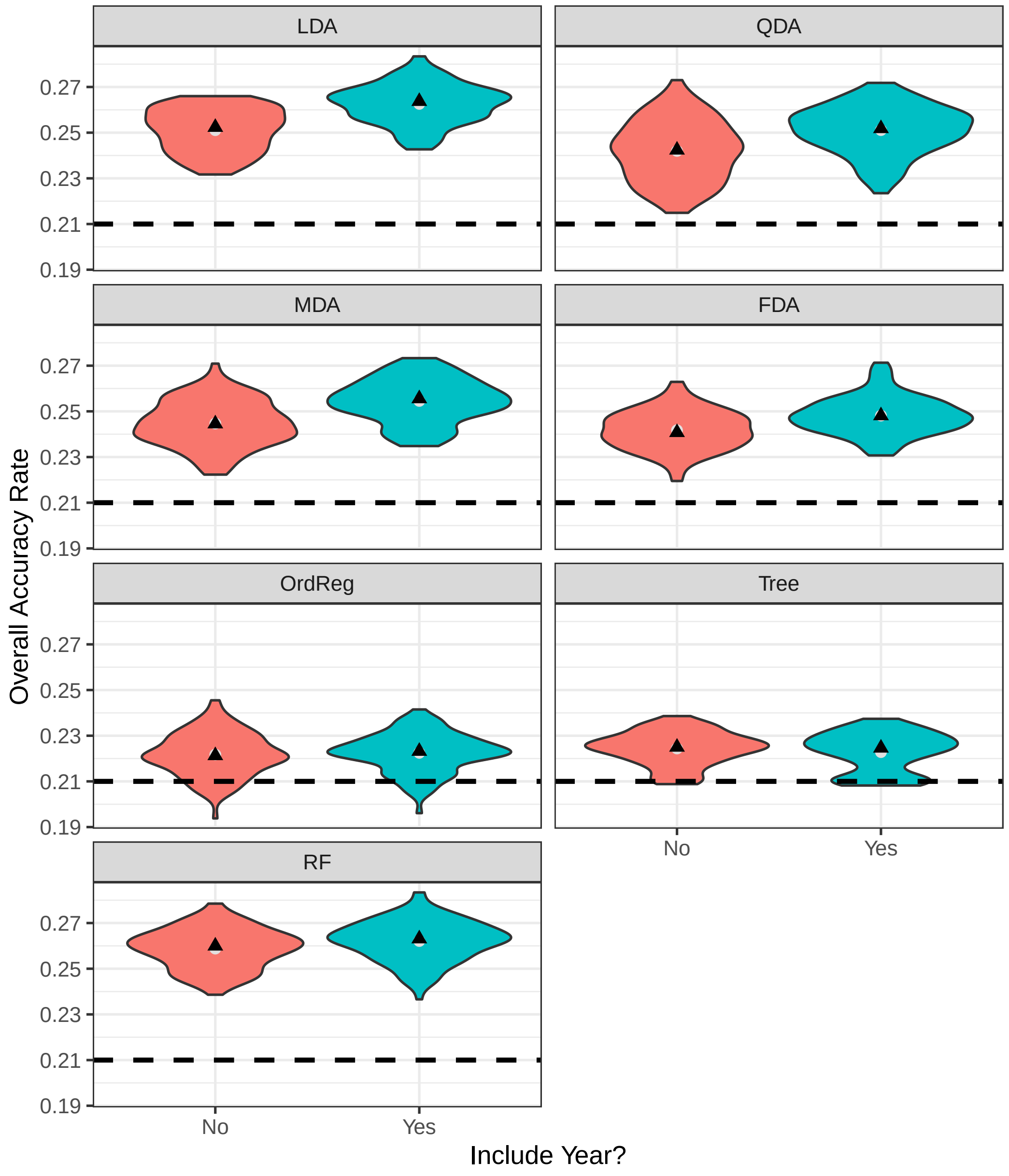

Figure B.2: $C \times R$, F-ratio $= 36.49$. The effect of including year as a predictor by classification method on the overall accuracy. The median is a black triangle, and the mean is a grey circle.

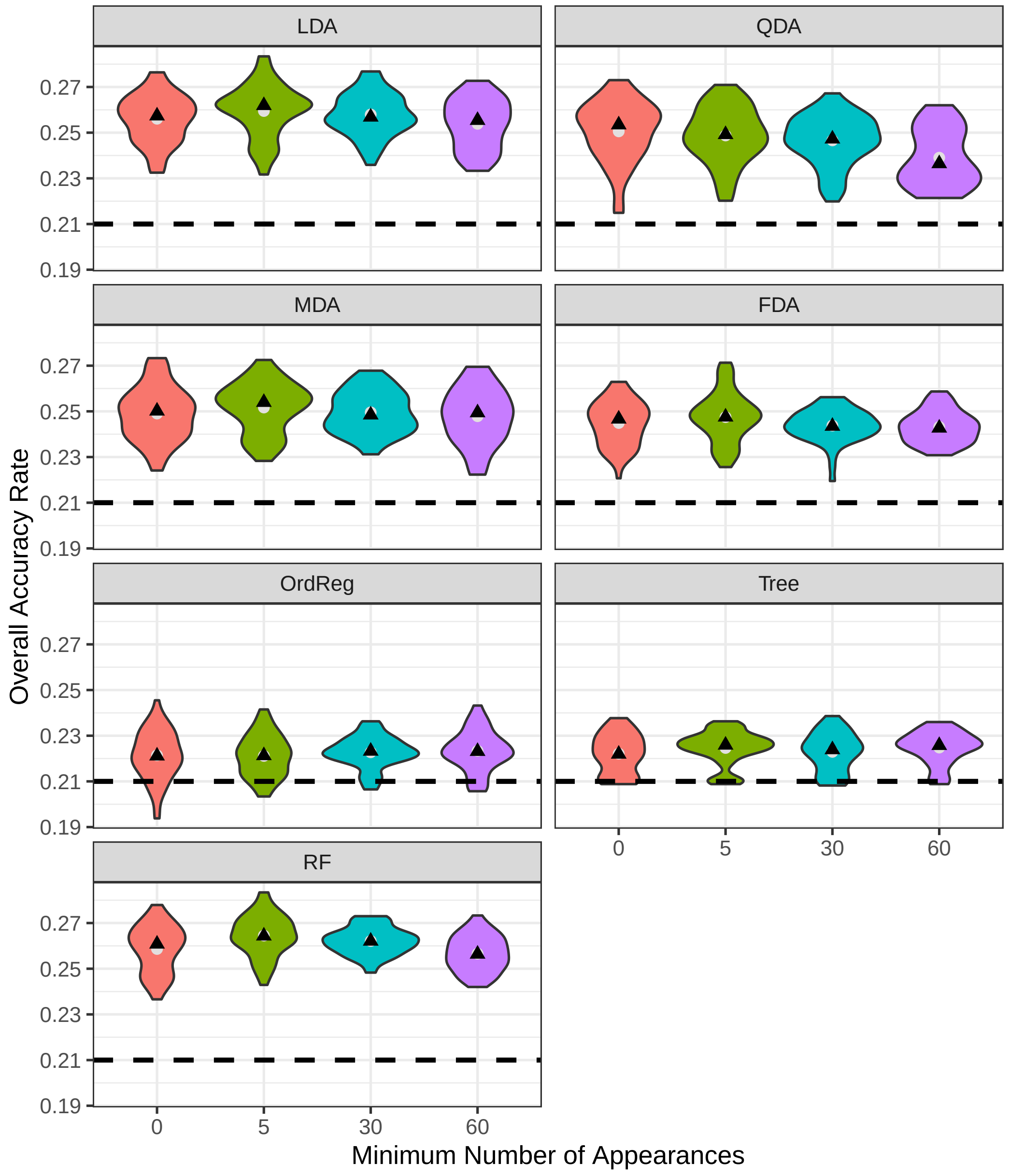

Figure B.3: $C \times F$, F-ratio = 5.213. The effect of filtering threshold by classification method on the overall accuracy. The median is a black triangle, and the mean is a grey circle.

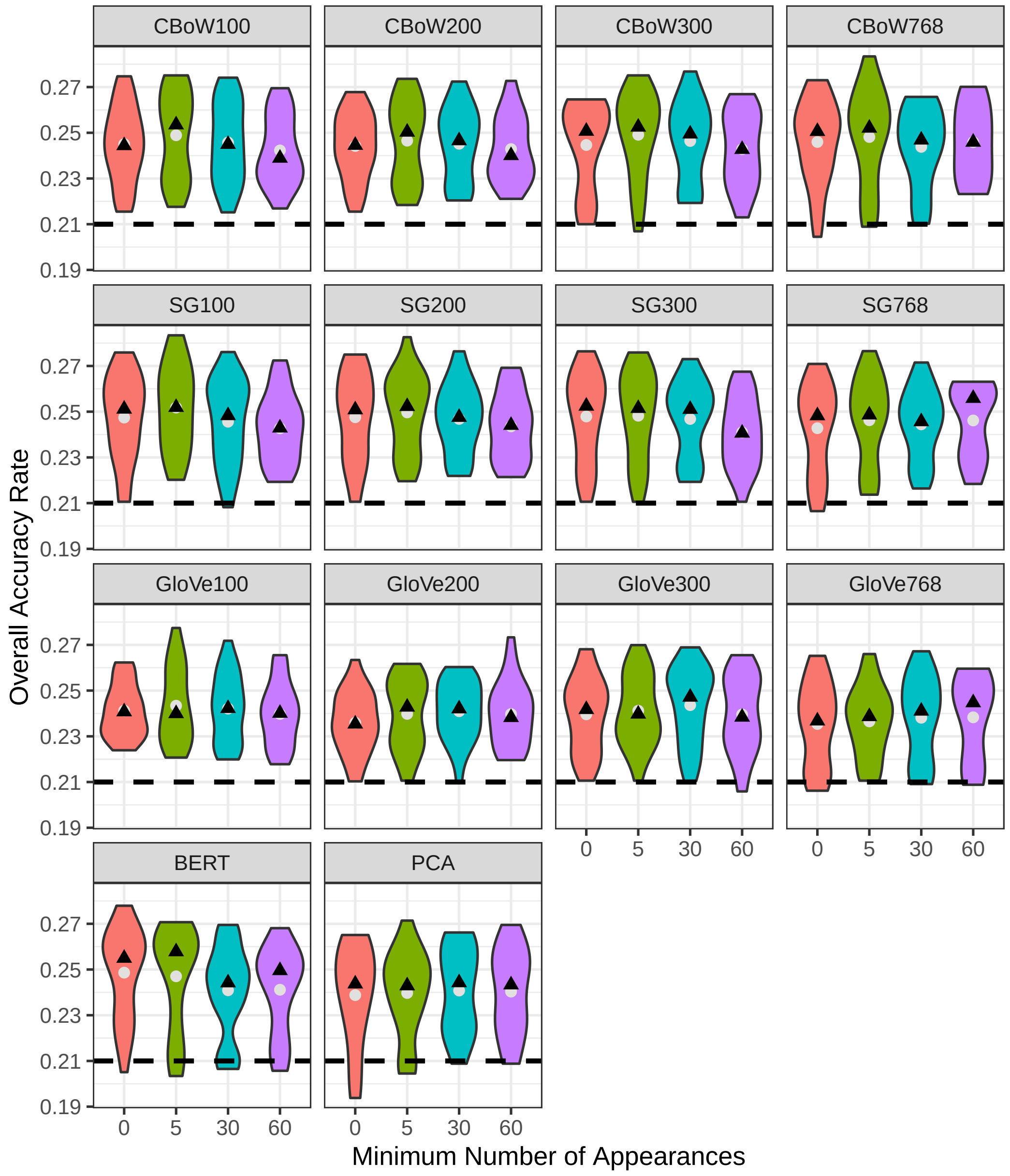

Figure B.4: $E \times F$, F-ratio = 2.805. The effect of filtering threshold by embedding method on the overall accuracy. The median is a black triangle, and the mean is a grey circle.

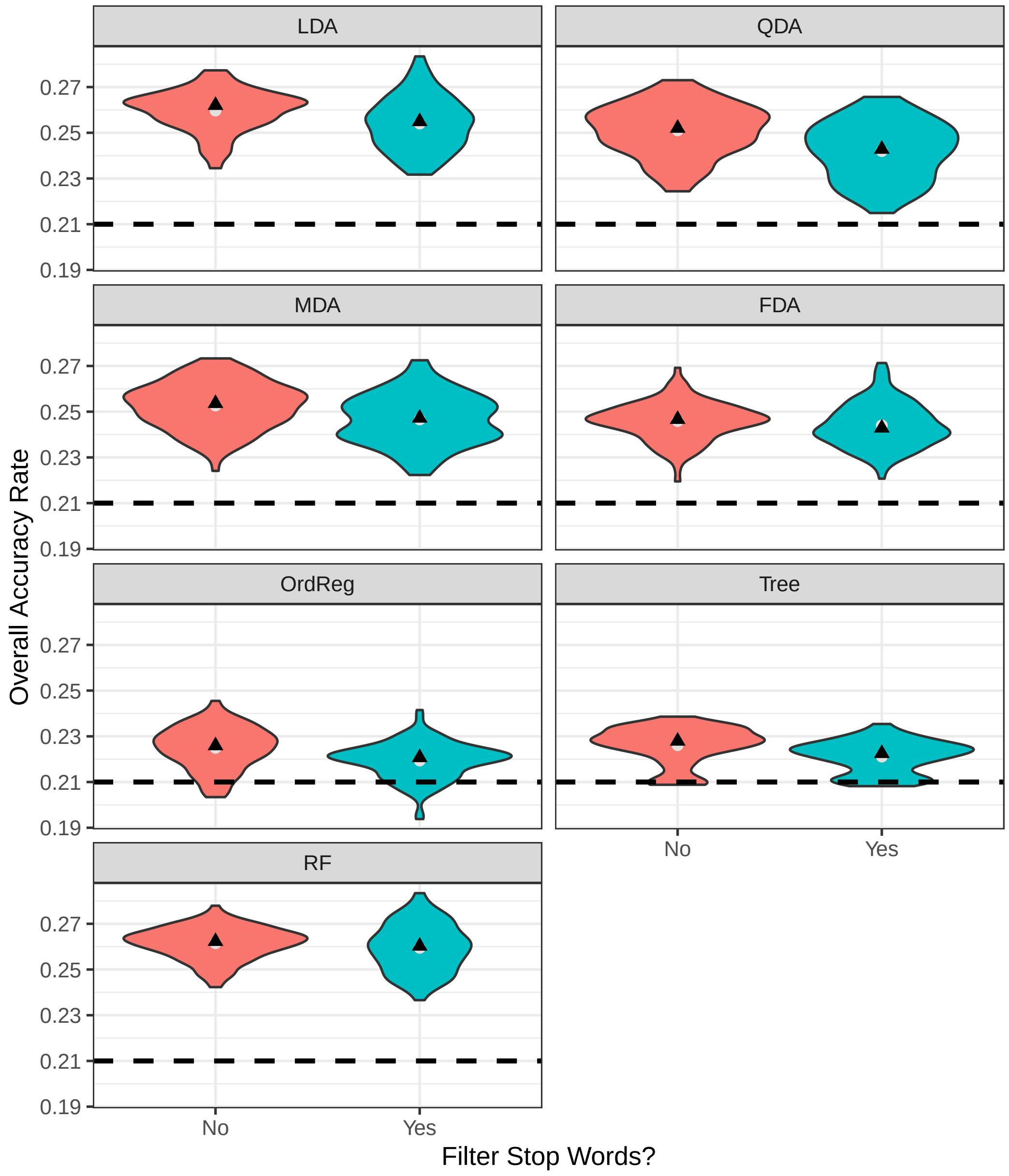

Figure B.5: $C \times S$, F-ratio = 8.694. The effect of filtering stop words by classification method on the overall accuracy. The median is a black triangle, and the mean is a grey circle.

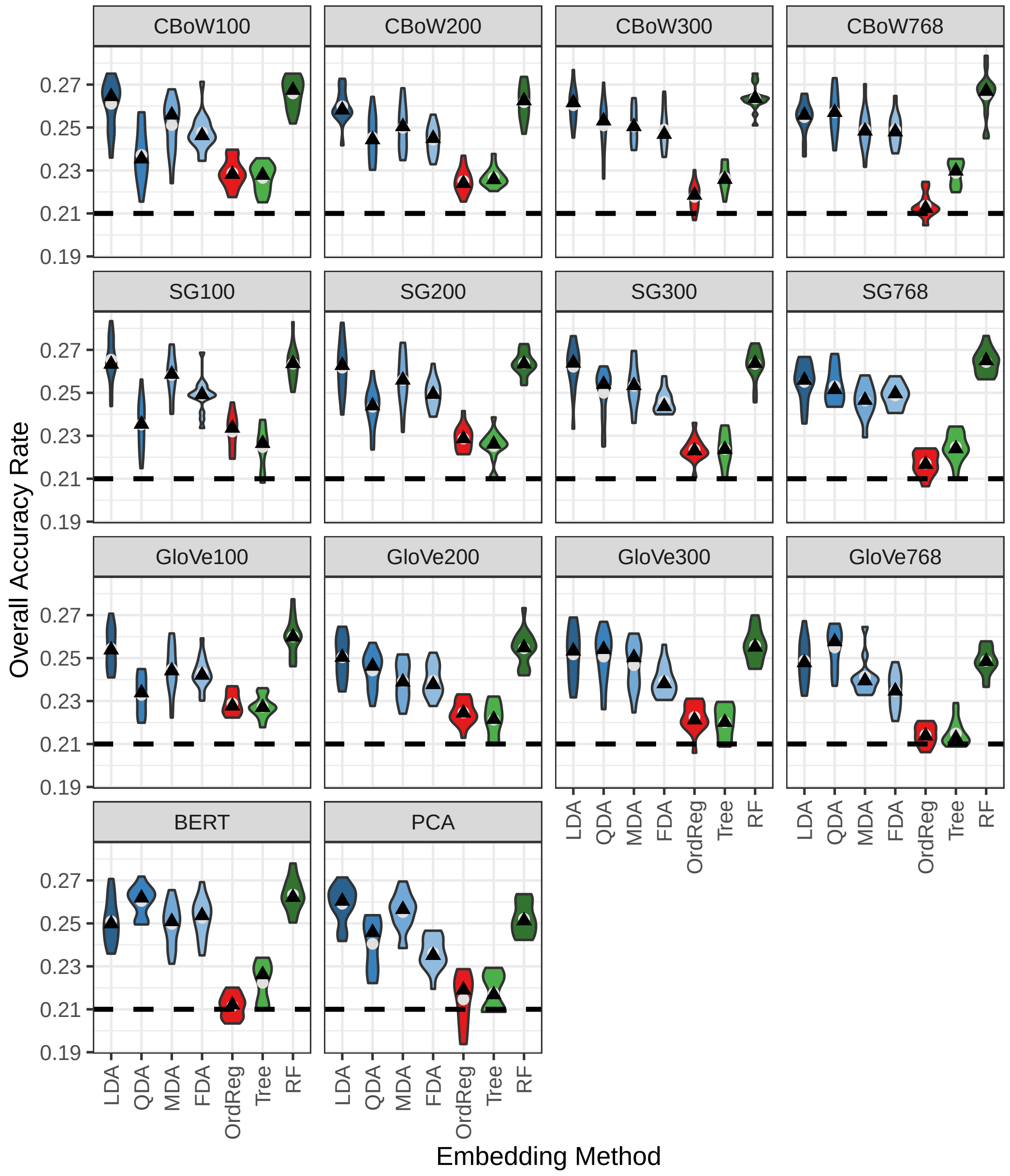

Figure B.6: $C \times E$, F-ratio = 11.719. The effect of classification method by embedding method on the overall accuracy. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

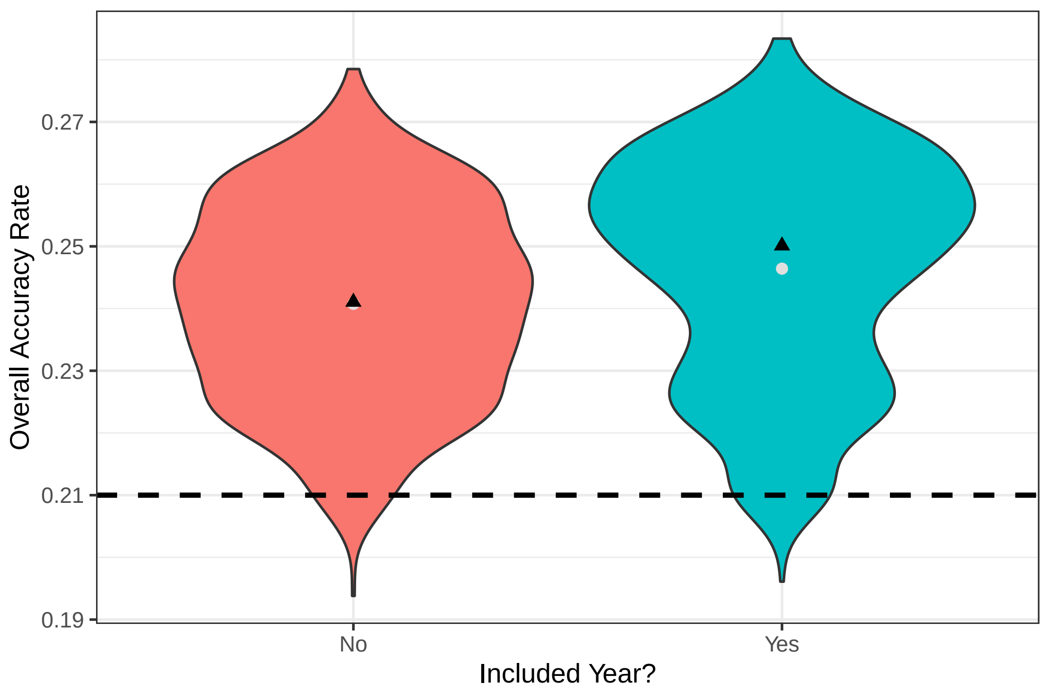

Figure B.7: *R*, F-ratio = 325.709. The effect of including year as a predictor on the overall accuracy. The median is a black triangle, and the mean is a grey circle.

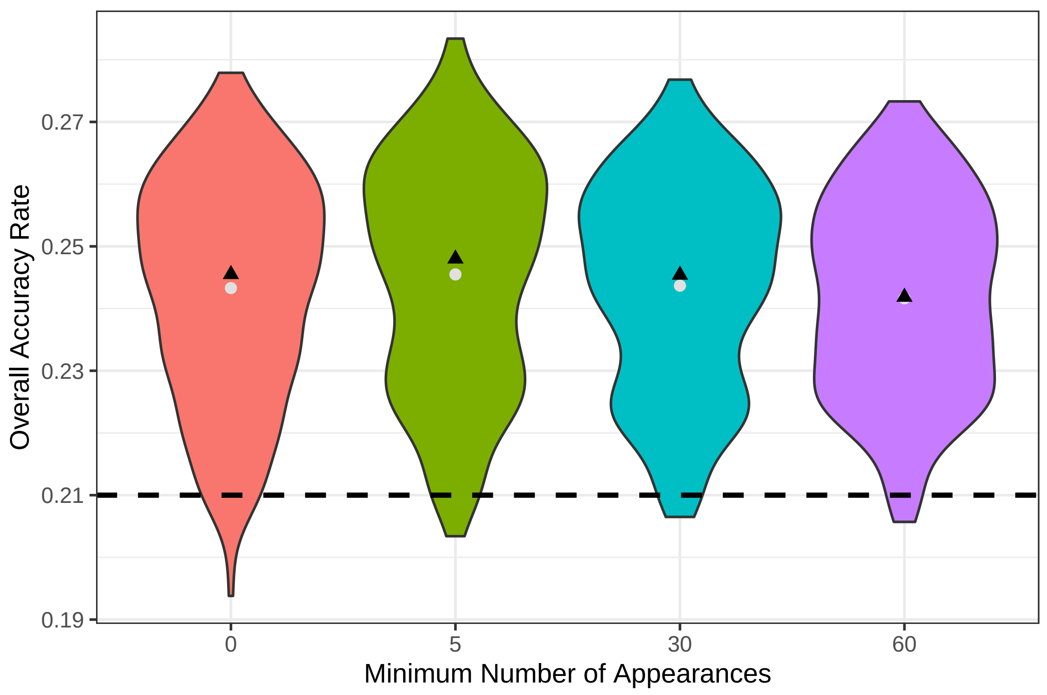

Figure B.8: F, F-ratio = 29.811. The effect of filtering threshold on the overall accuracy. The median is a black triangle, and the mean is a grey circle.

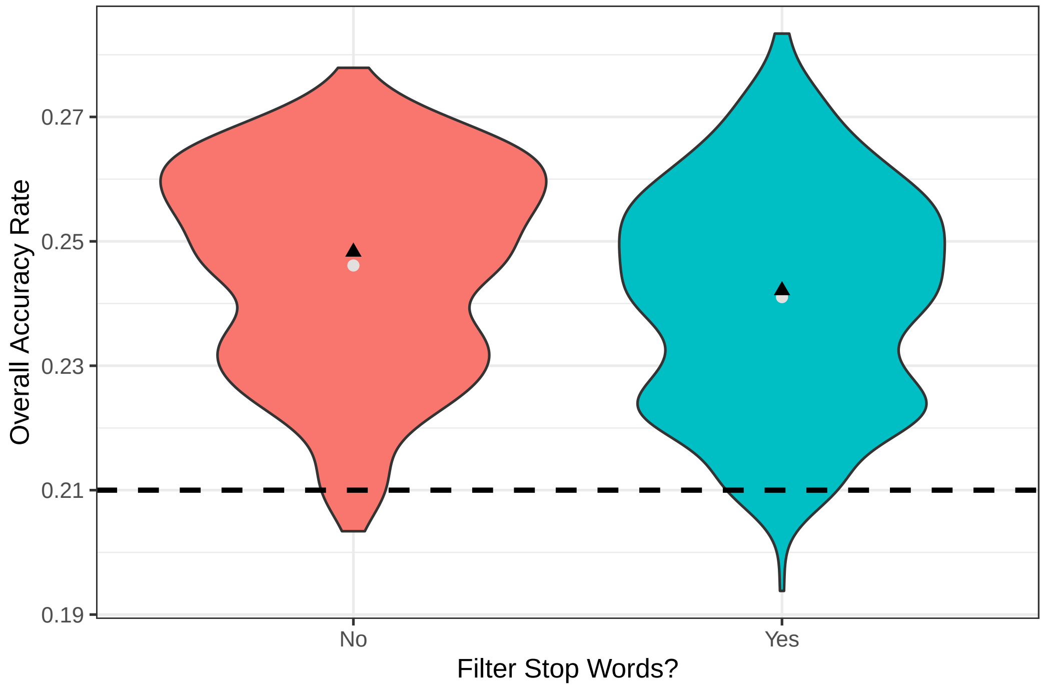

Figure B.9: S, F-ratio = 249.807. The effect of filtering stop words on the overall accuracy. The median is a black triangle, and the mean is a grey circle.

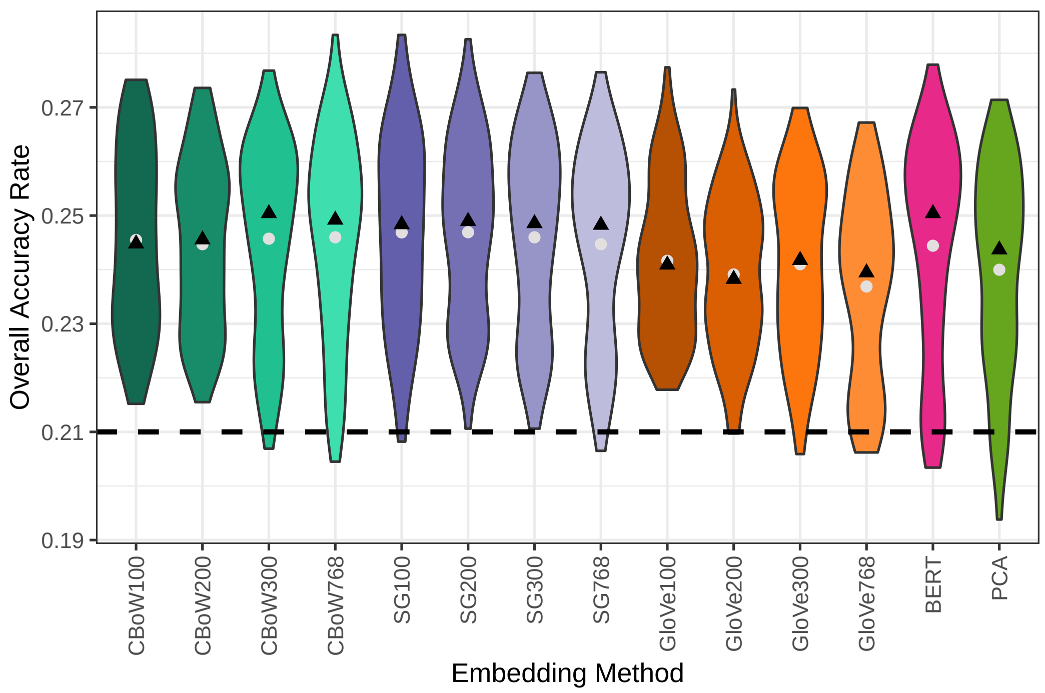

Figure B.10: E, F-ratio = 31.29. The effect of embedding method on overall accuracy. Word2Vec Continuous Bag of Words are in the blue-green shades, and Word2Vec Skip-Gram are in the blue-purple shades. GloVe is in the brown-orange shades. BERT is in pink, and PCA is in green. The median is a black triangle, and the mean is a grey circle.

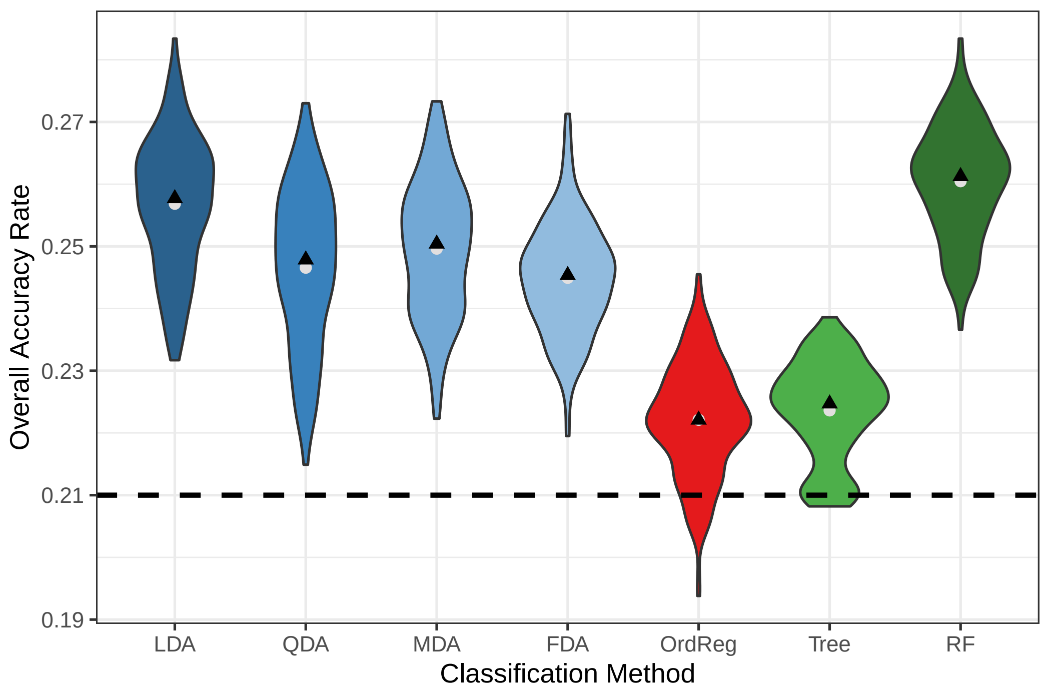

Figure B.11: C, F-ratio = 1369.341. The effect of classification method on overall accuracy. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

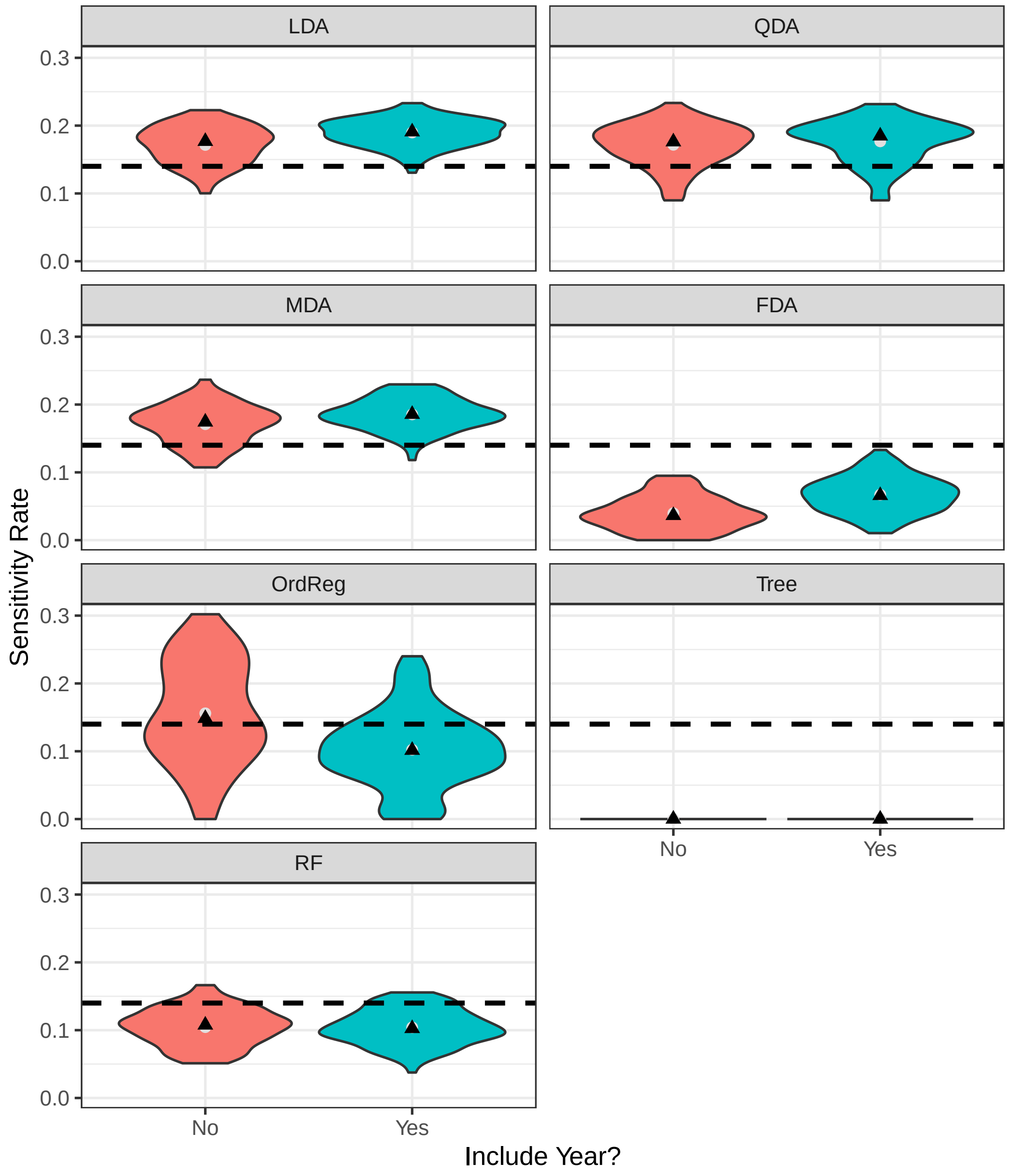

Figure B.12: $C \times R$, F-ratio = 49.388. The effect of including year as a predictor by classification method on the sensitivity of a true label. The median is a black triangle, and the mean is a grey circle.

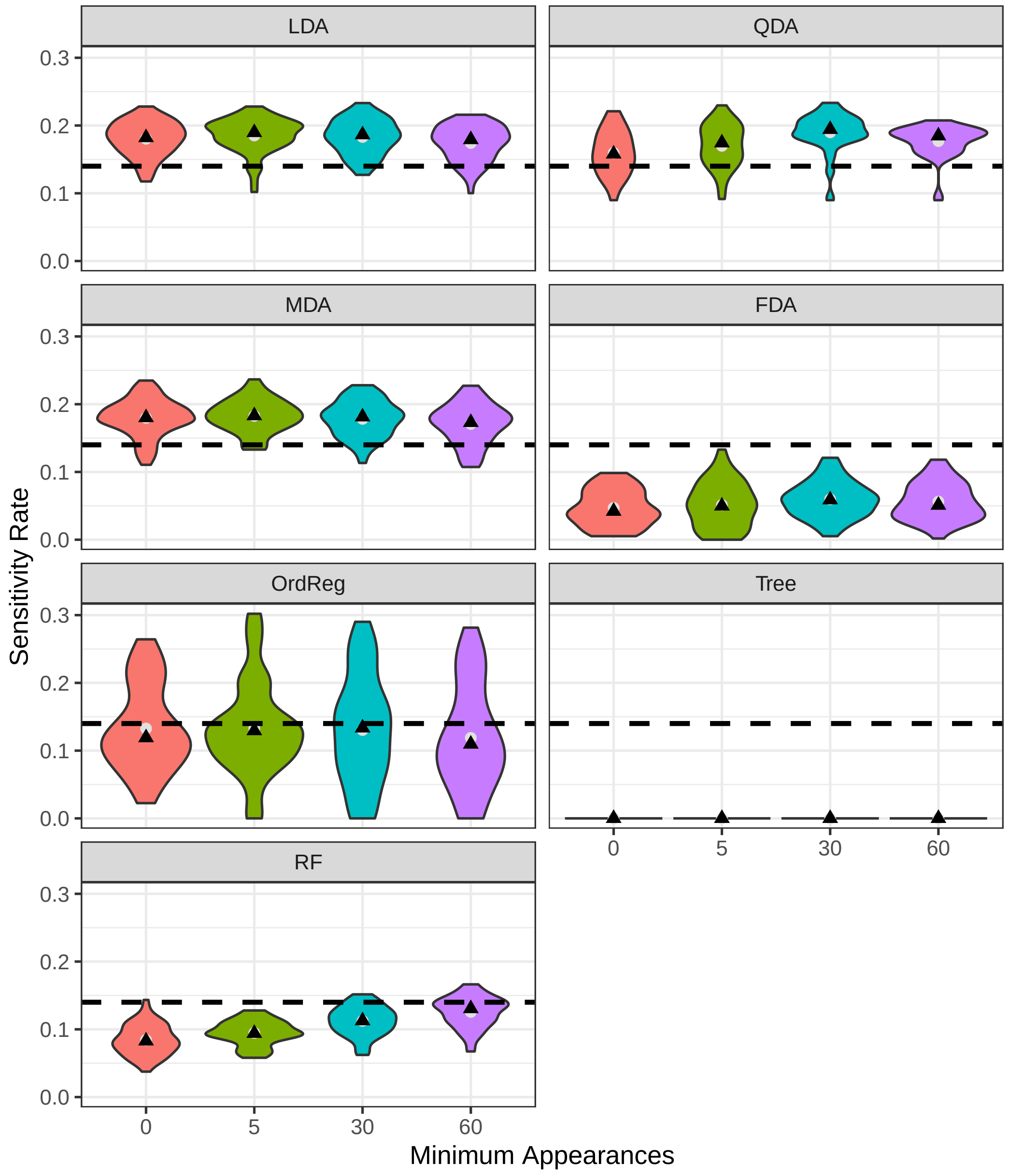

Figure B.13: $C \times F$, F-ratio = 5.371. The effect of filtering threshold by classification method on the sensitivity of a true label. The median is a black triangle, and the mean is a grey circle.

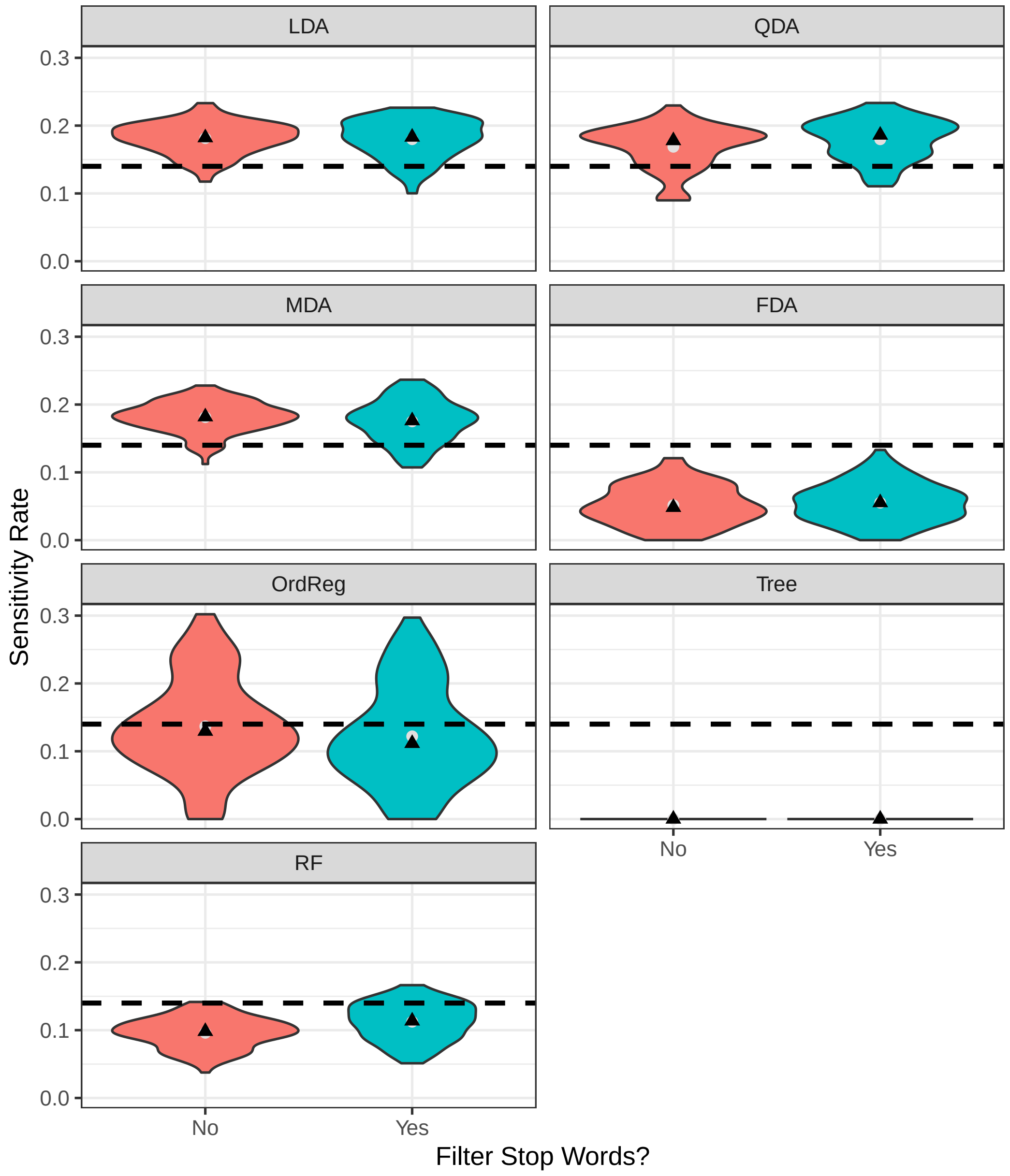

Figure B.14: $C \times S$, F-ratio = 7.727. The effect of filtering stop words by classification method on the sensitivity of a true label. The median is a black triangle, and the mean is a grey circle.

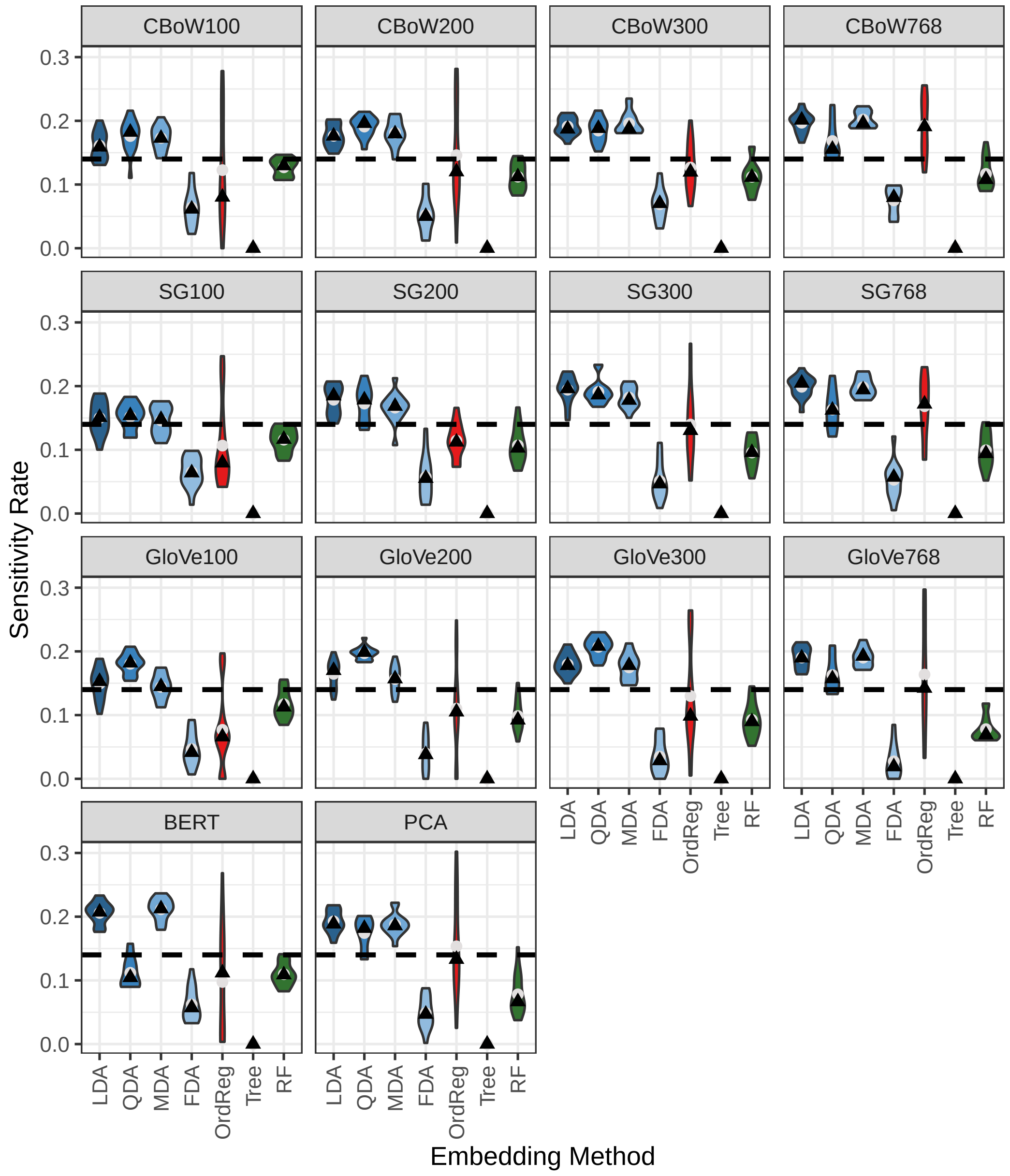

Figure B.15: $C \times E$, F-ratio = 7.095. The effect of classification method by embedding method on the sensitivity of a true label. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

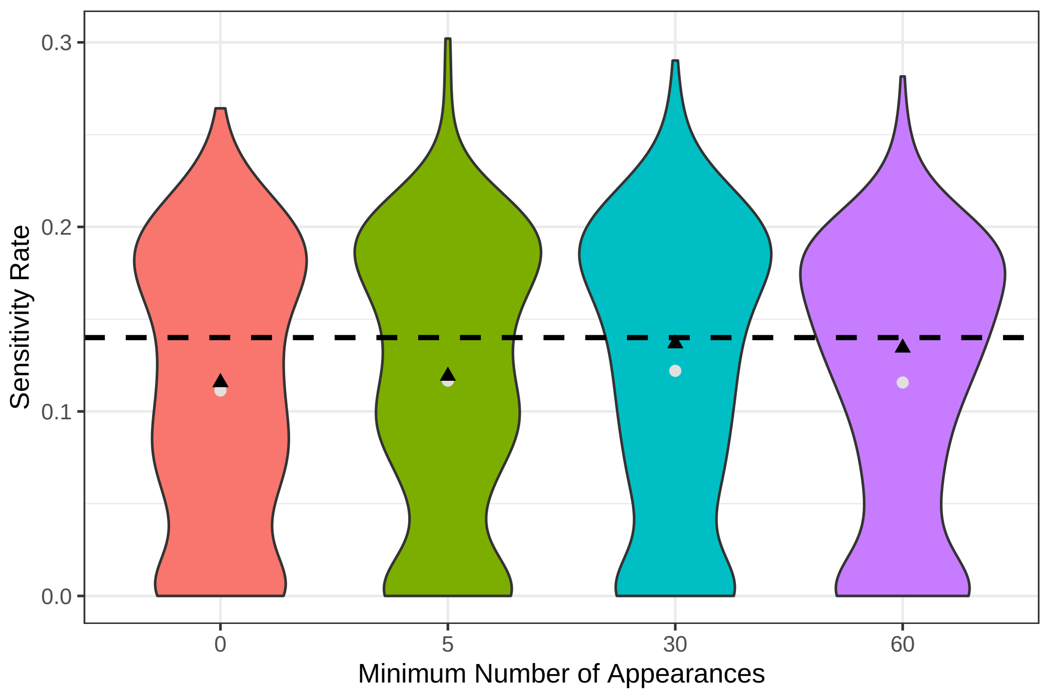

Figure B.16: F, F-ratio = 9.608. The effect of filtering threshold on the sensitivity of a true label. The median is a black triangle, and the mean is a grey circle.

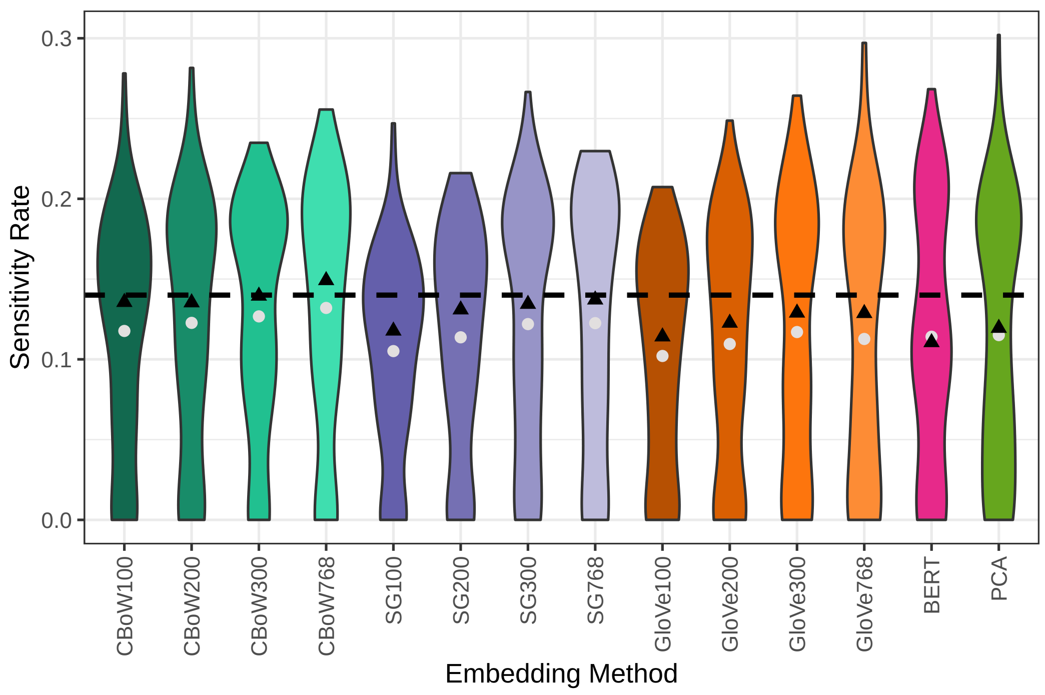

Figure B.17: E, F-ratio = 10.451. The effect of embedding method on the sensitivity of a true label. Word2Vec Continuous Bag of Words are in the blue-green shades, and Word2Vec Skip-Gram are in the blue-purple shades. GloVe is in the brown-orange shades. BERT is in pink, and PCA is in green. The median is a black triangle, and the mean is a grey circle.

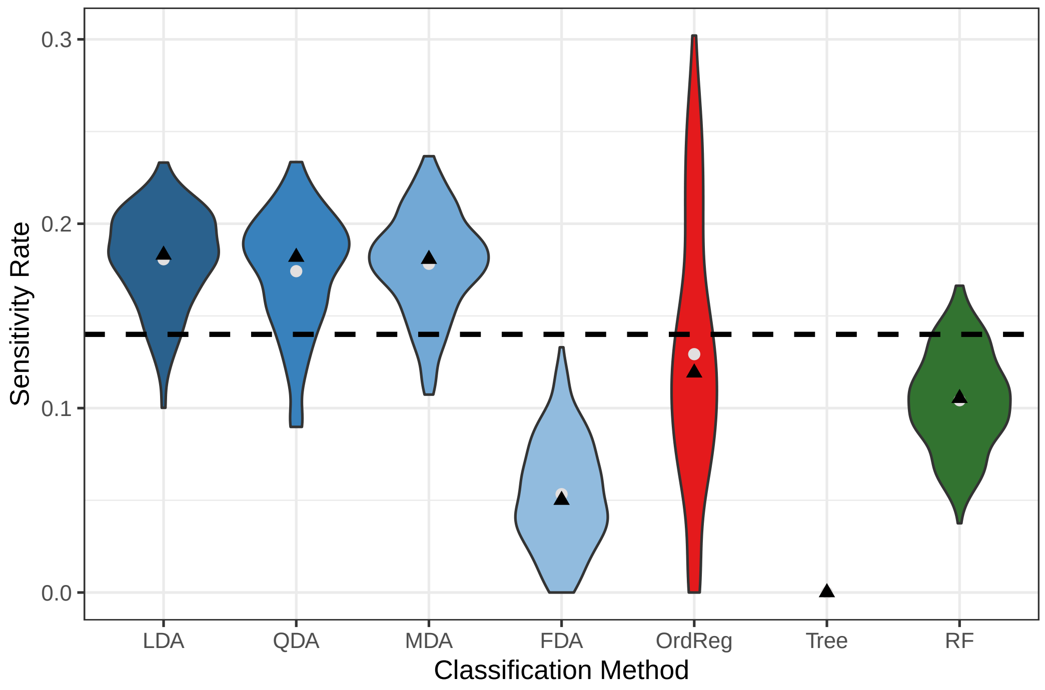

Figure B.18: C, F-ratio = 1443.107. The effect of classification method on the sensitivity of a true label. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

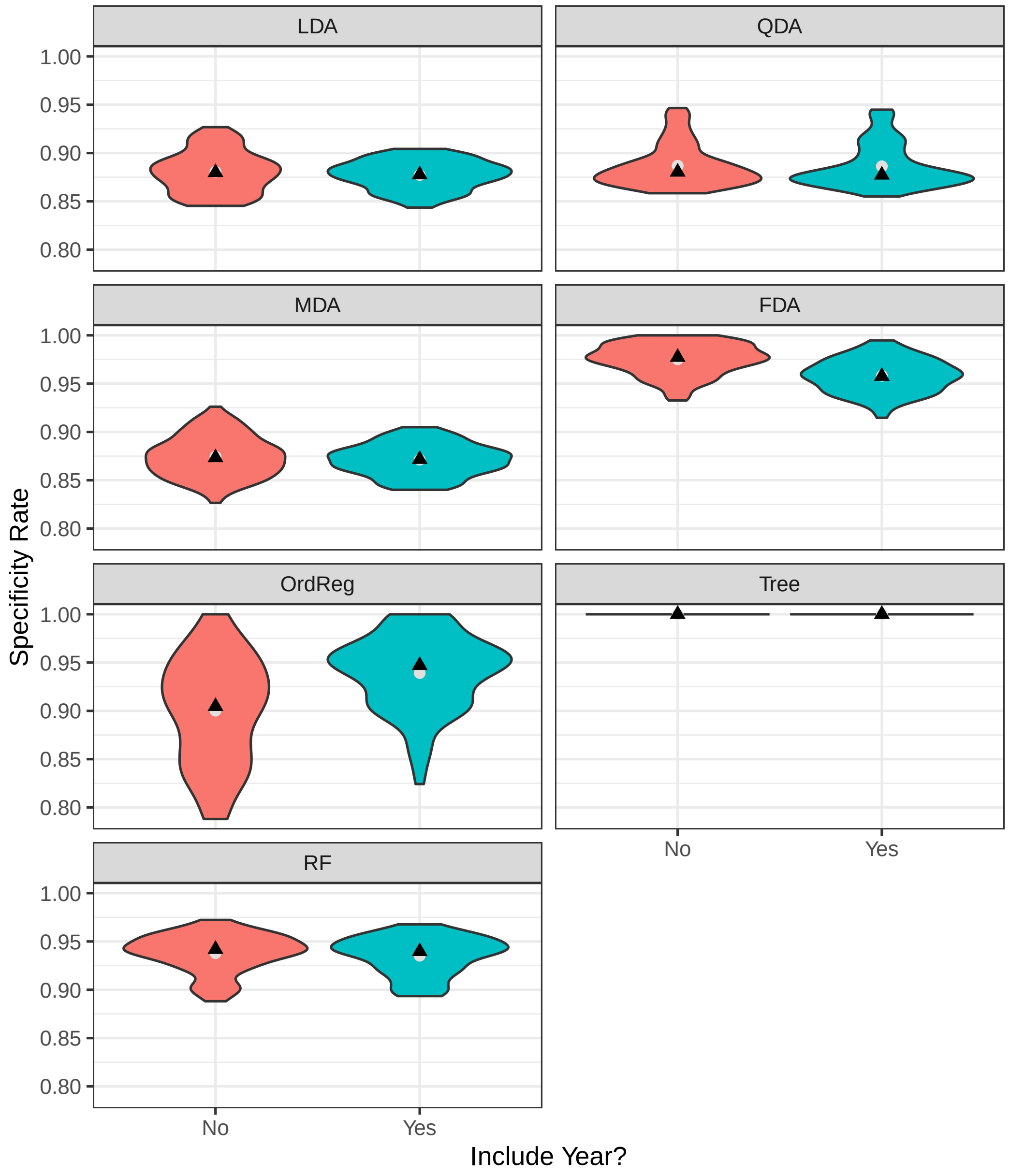

Figure B.19: $C \times R$, F-ratio $= 47.17$. The effect of including year as a predictor by classification method on the specificity of a true label. The median is a black triangle, and the mean is a grey circle.

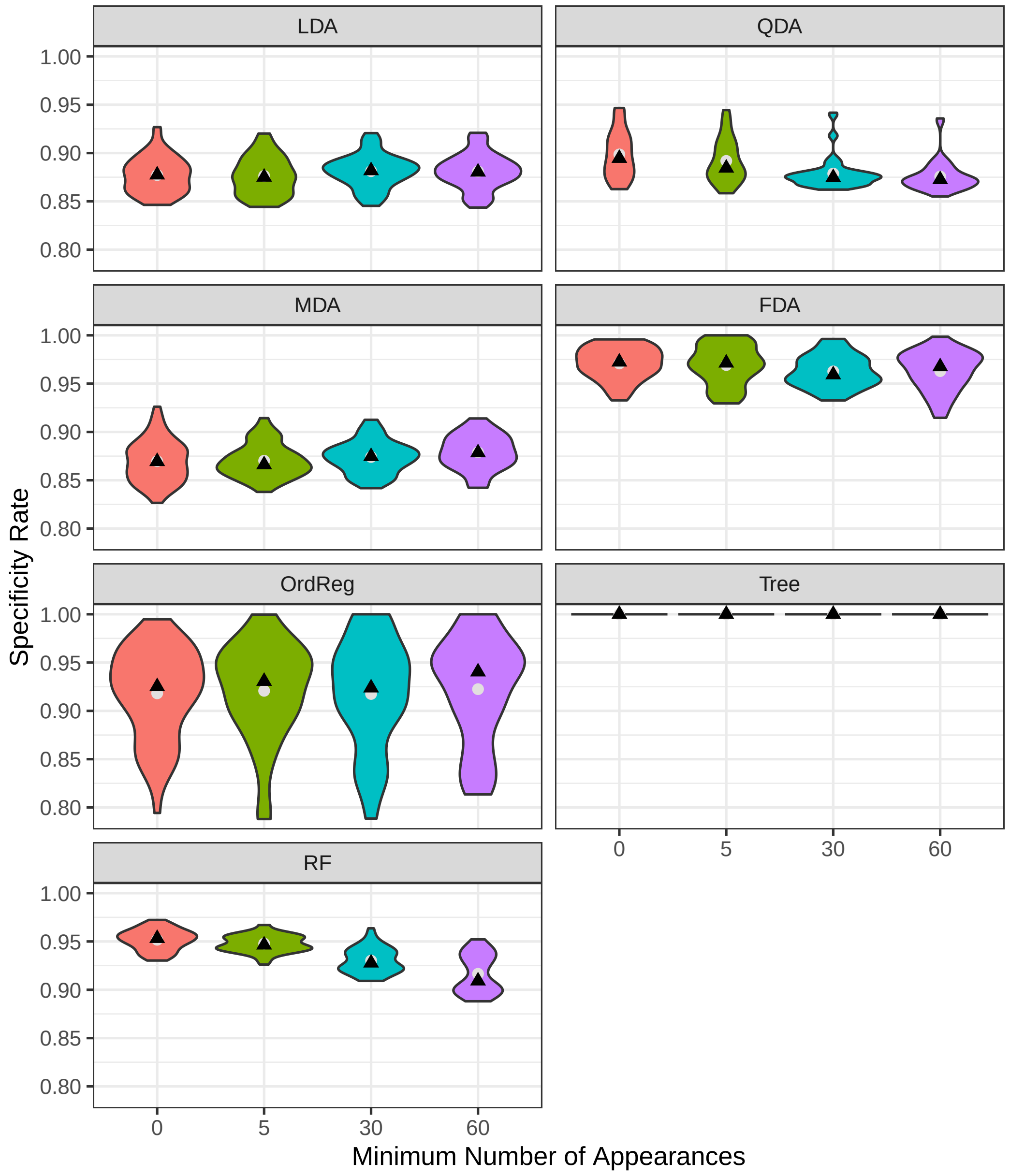

Figure B.20: $C \times F$, F-ratio = 8.112. The effect of filtering threshold by classification method on the specificity of a true label. The median is a black triangle, and the mean is a grey circle.

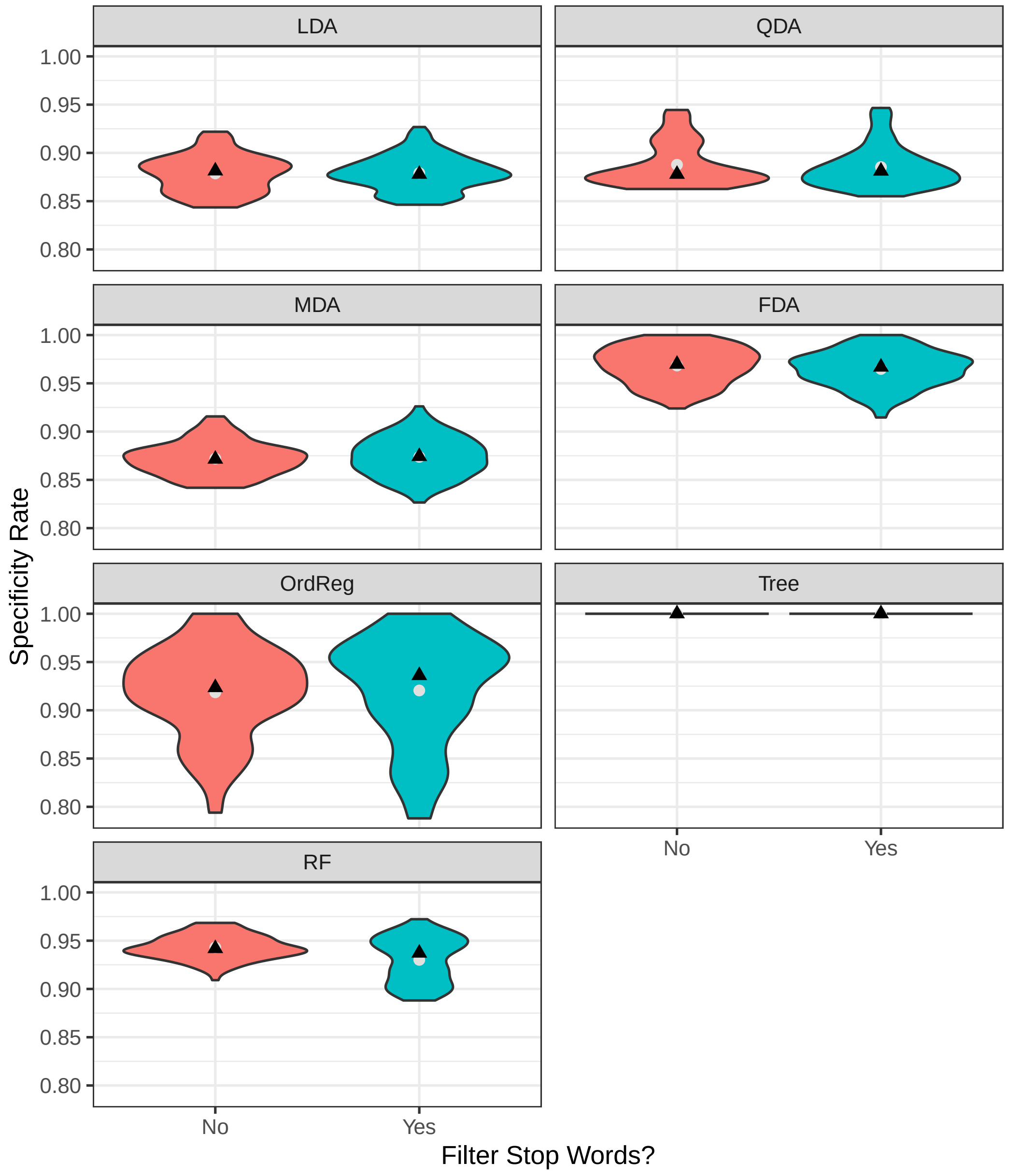

Figure B.21: $C \times S$, F-ratio = 4.565. The effect of filtering stop words by classification method on the specificity of a true label. The median is a black triangle, and the mean is a grey circle.

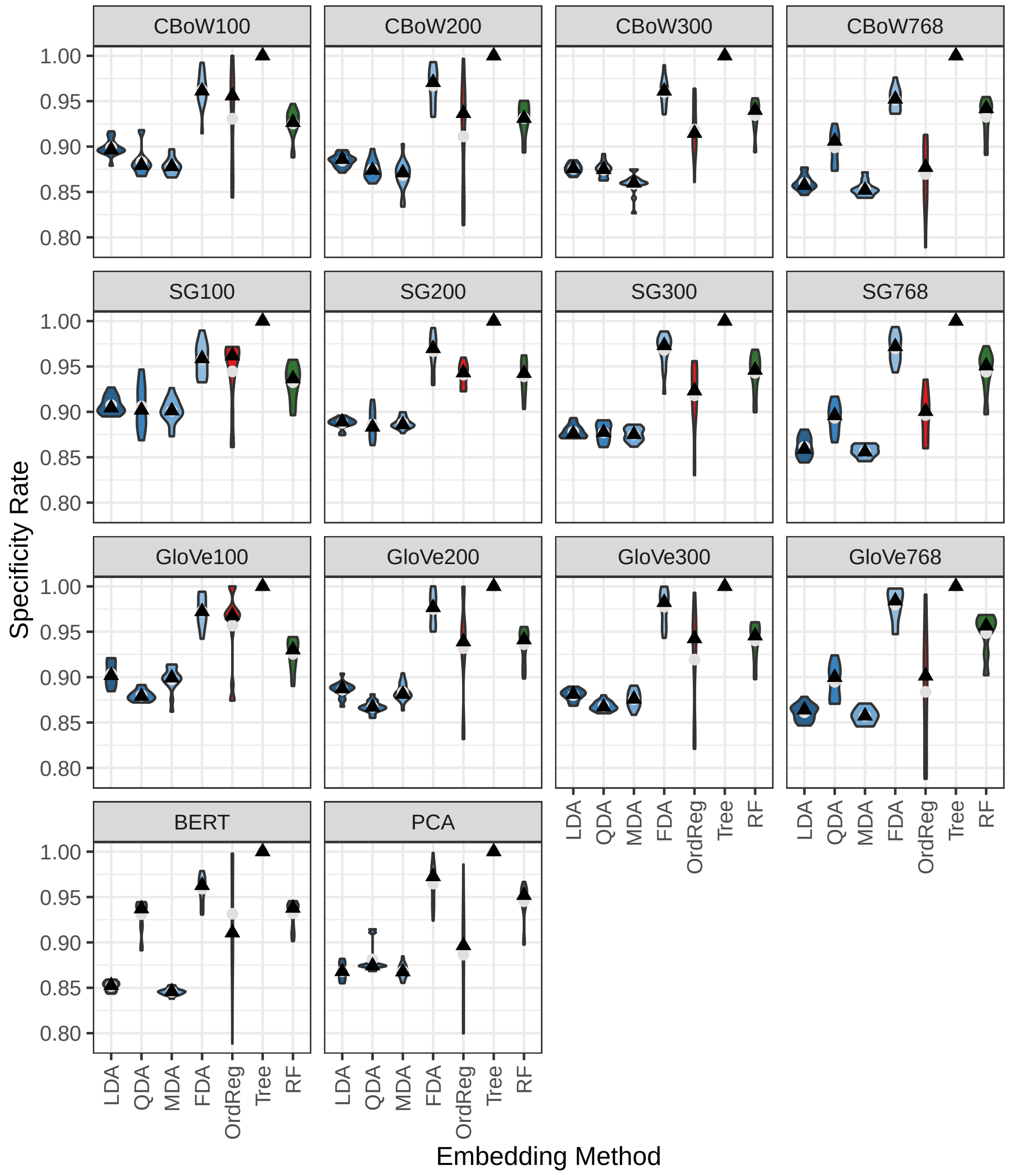

Figure B.22: $C \times E$, F-ratio = 9.182. The effect of classification method by embedding method on the specificity of a true label. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

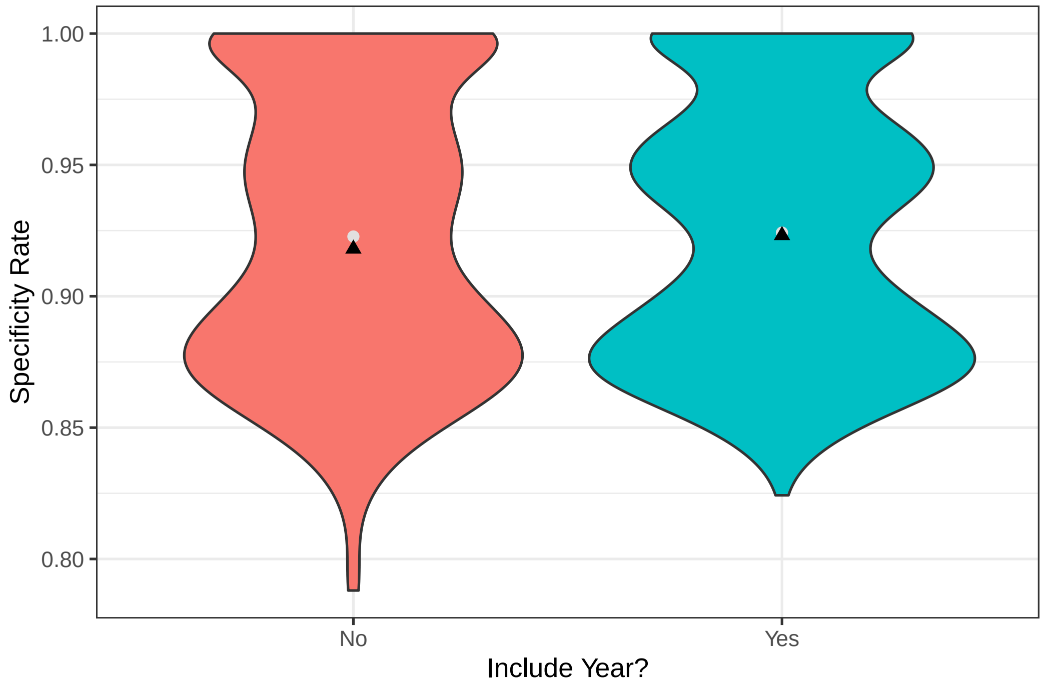

Figure B.23: R, F-ratio = 2.606. The effect of including year as a predictor on the specificity of a true label. The median is a black triangle, and the mean is a grey circle.

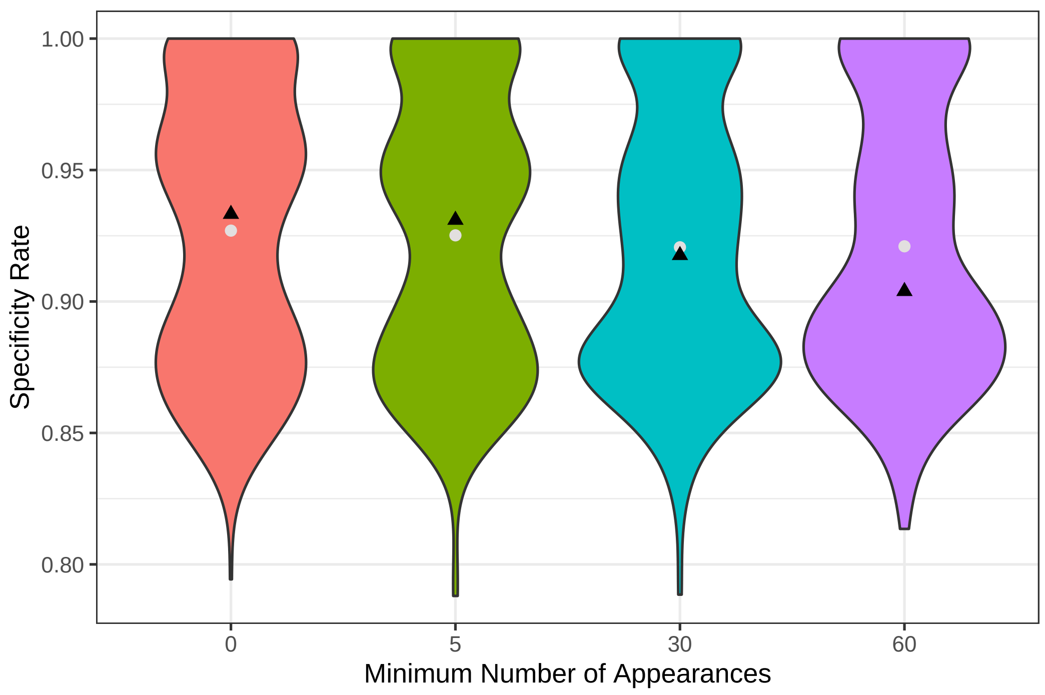

Figure B.24: F, F-ratio = 17.417. The effect of filtering threshold on the specificity of a true label. The median is a black triangle, and the mean is a grey circle.

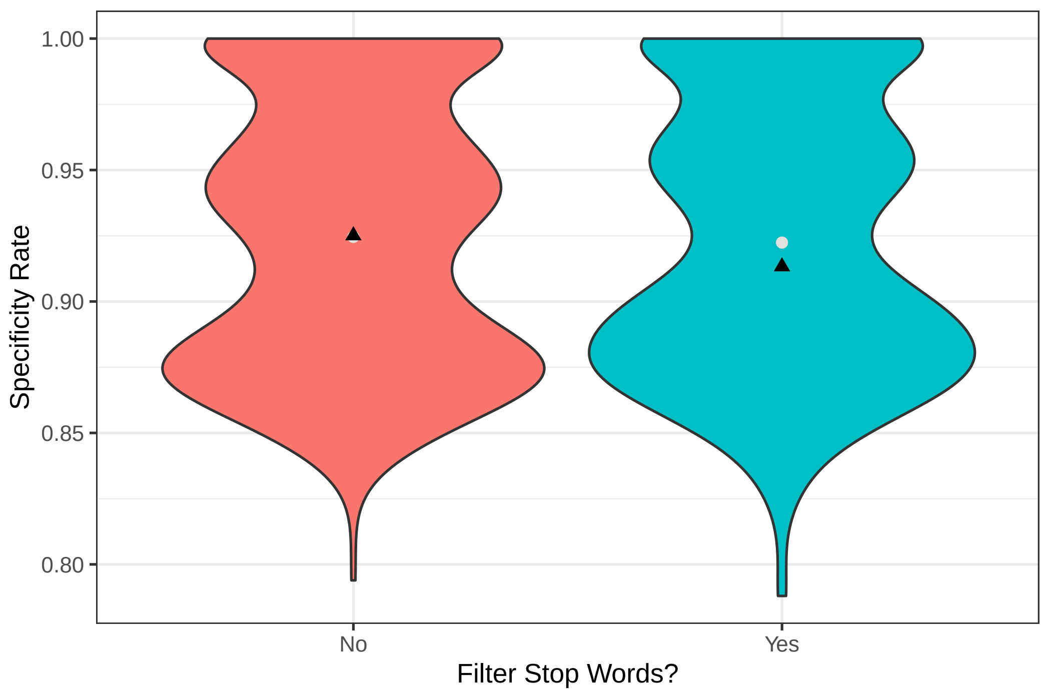

Figure B.25: S, F-ratio = 4.427. The effect of filtering stop words on the specificity of a true label. The median is a black triangle, and the mean is a grey circle.

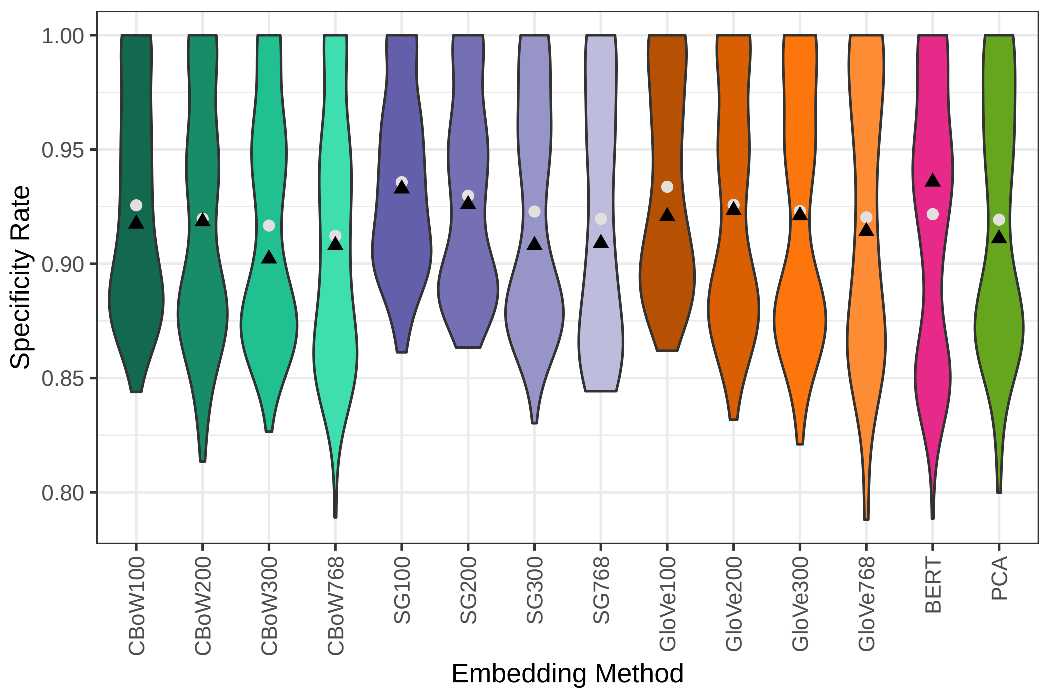

Figure B.26: E, F-ratio = 15.516. The effect of embedding method on the specificity of a true label. Word2Vec Continuous Bag of Words are in the blue-green shades, and Word2Vec Skip-Gram are in the blue-purple shades. GloVe is in the brown-orange shades. BERT is in pink, and PCA is in green. The median is a black triangle, and the mean is a grey circle.

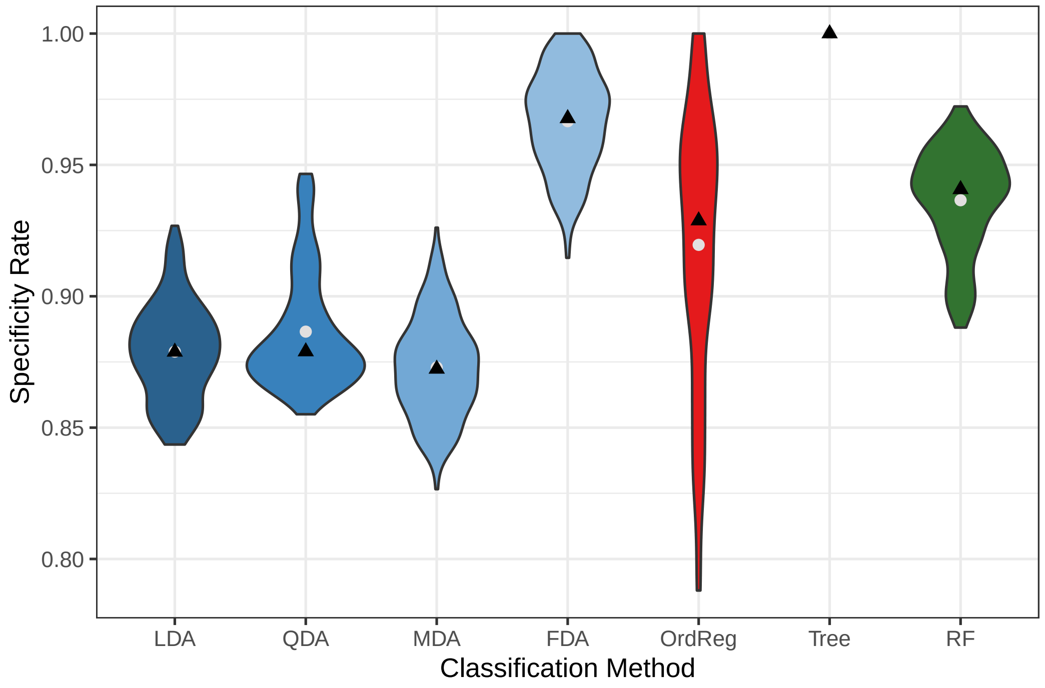

Figure B.27: C, F-ratio = 1507.22. The effect of classification method on the specificity of a true label. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

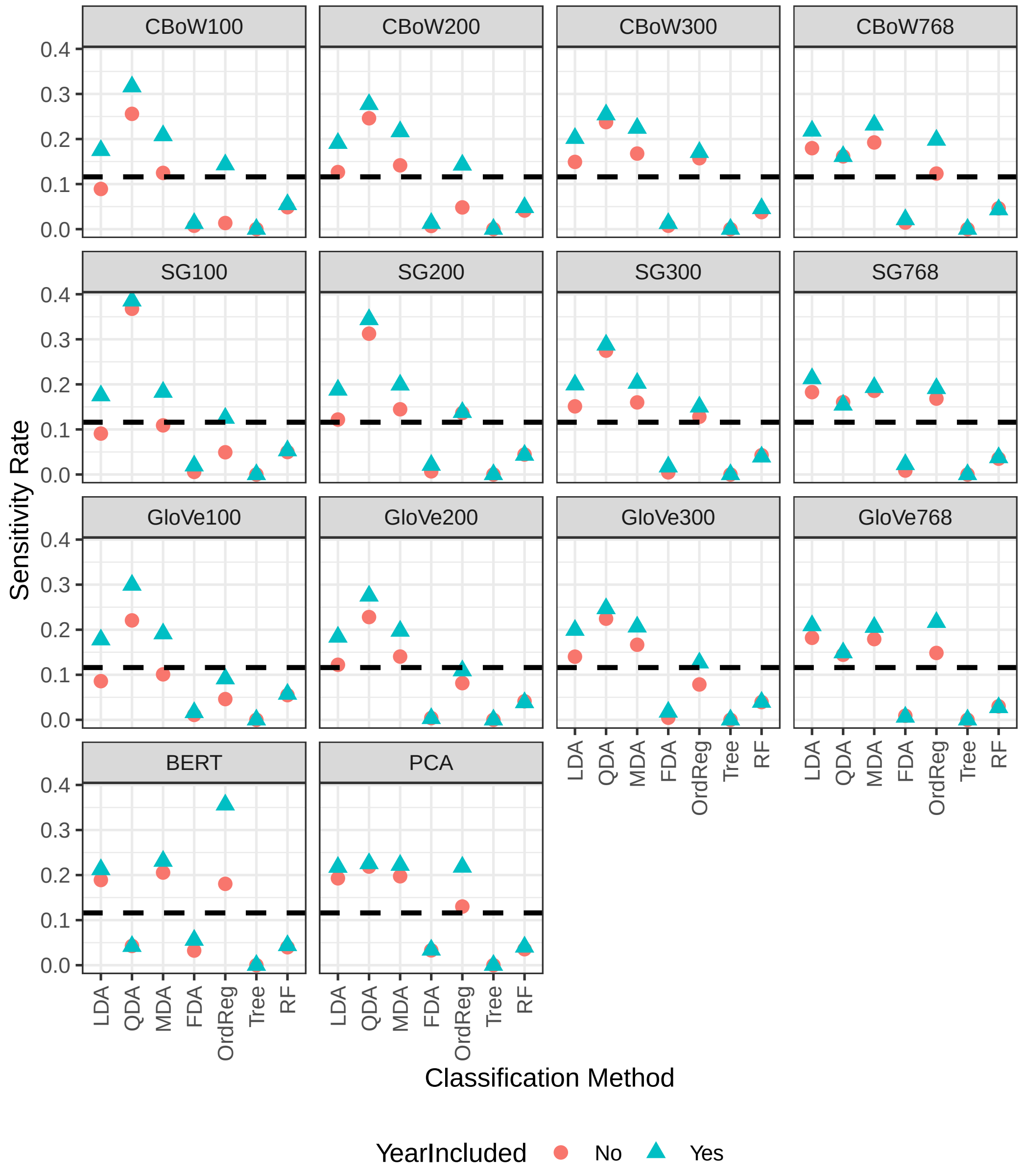

Figure B.28: $C \times E \times R$, F-ratio = 2.353. The effect of classification method, embedding method, and inclusion of year as a predictor on the sensitivity of a pants-on-fire label. Only the means are plotted here. Violin plots are not used due to the unbalanced number of combinations between the levels since there were computational issues. There are fewer combinations observed when year is included as a predictor.

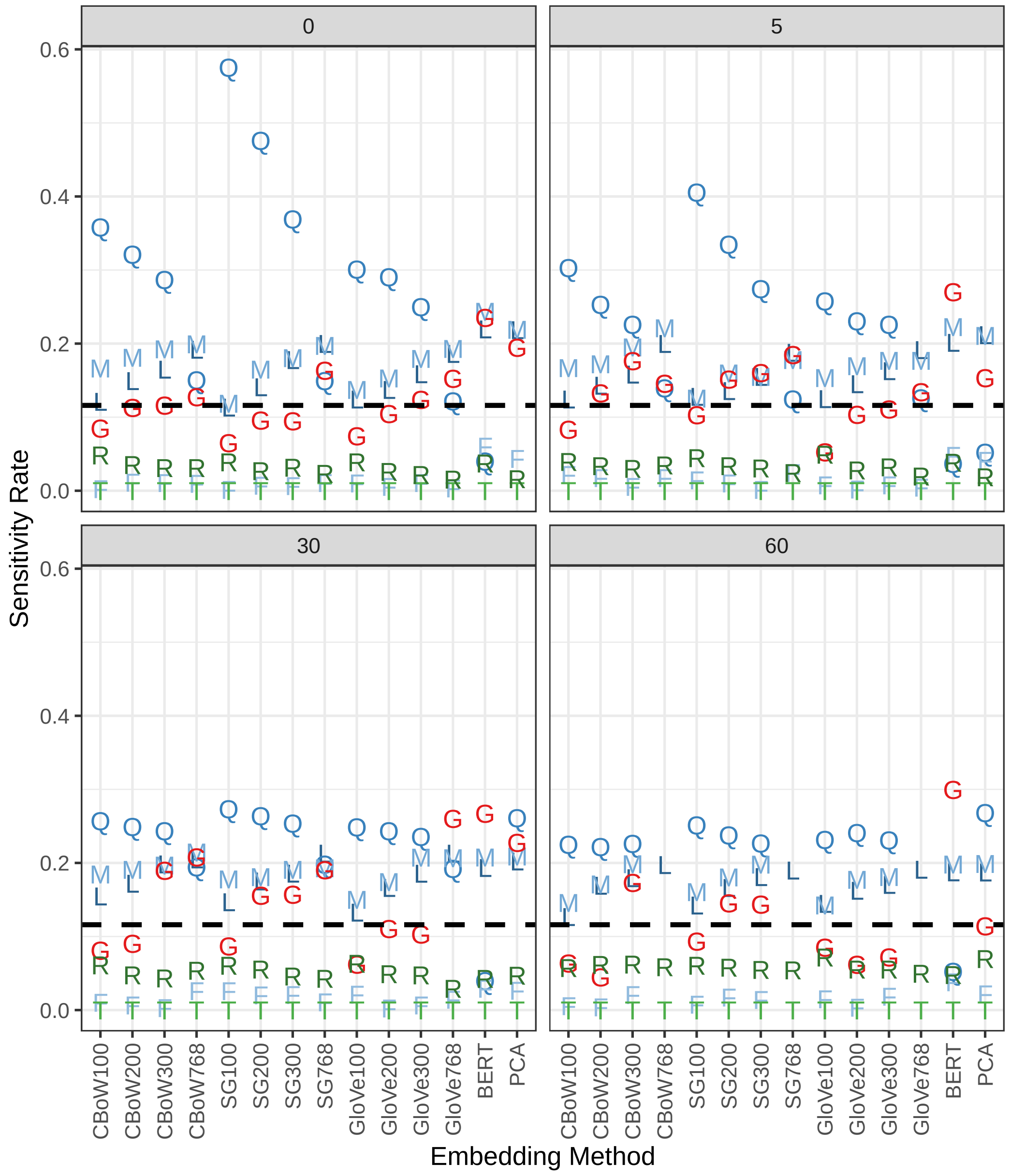

Figure B.29: $C \times E \times F$, F-ratio = 2.84. The effect of classification method, embedding method, and filtering threshold on the sensitivity of a pants-on-fire label. Each classification method is represented by a letter. T is for classification tree, and G is for ordinal regression. All others are based on the first letter of the method name. Only the means are plotted here. Violin plots are not used due to the number of levels of filtering threshold.

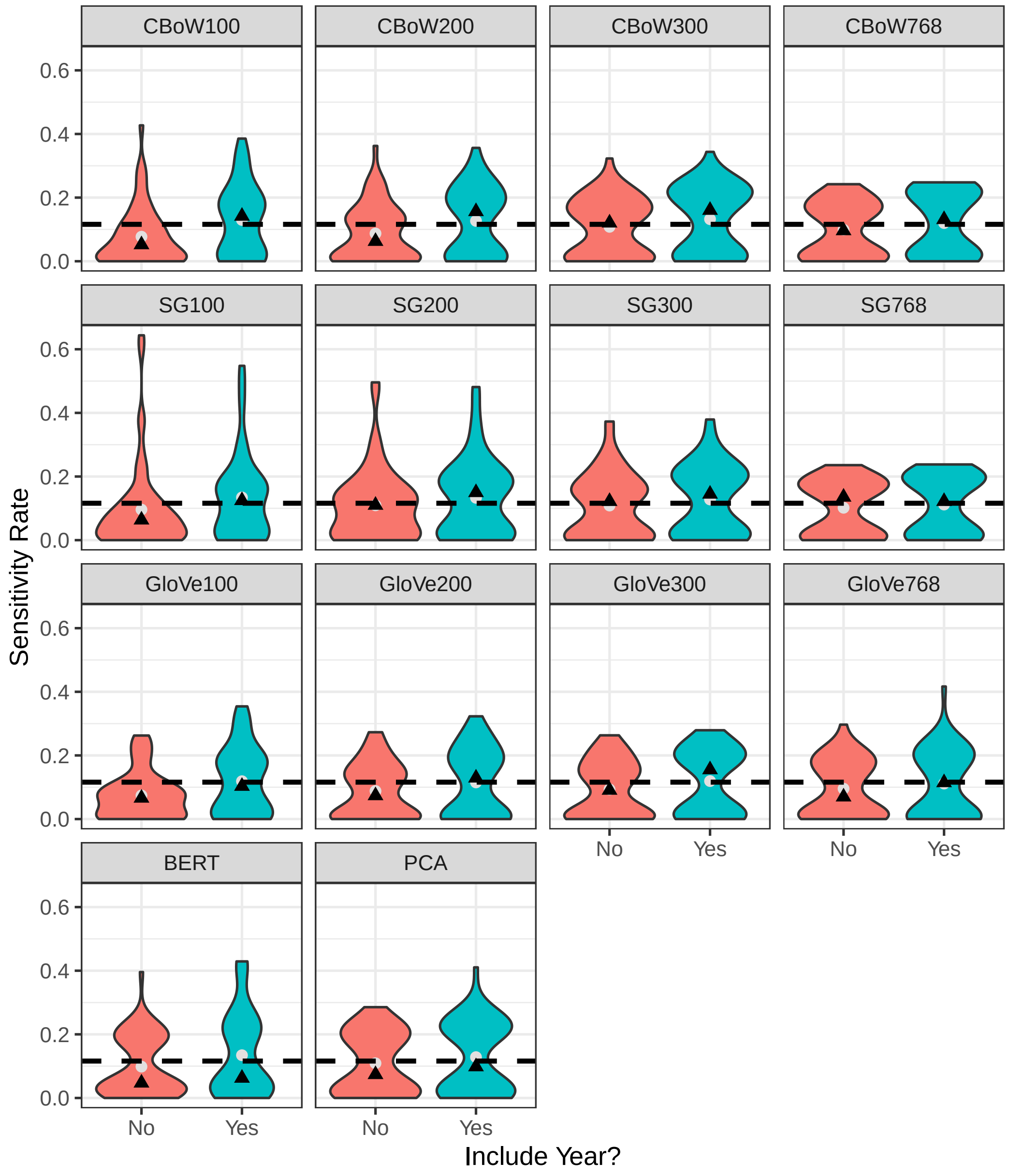

Figure B.30: $E \times R$, F-ratio $= 4.217$. The effect of including year as a predictor by embedding method on the sensitivity of a pants-on-fire label. The median is a black triangle, and the mean is a grey circle.

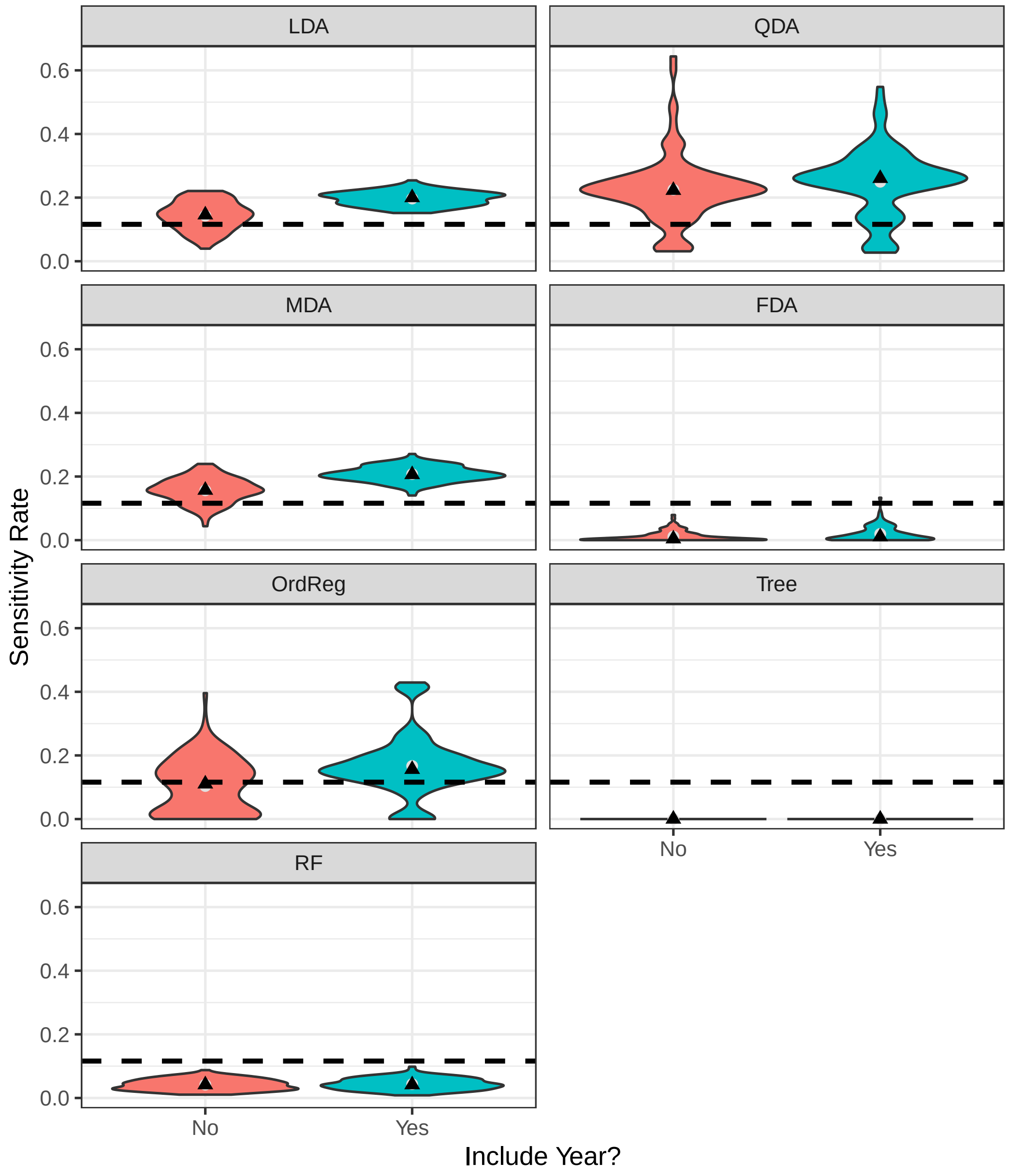

Figure B.31: $C \times R$, F-ratio $= 4.217$. The effect of including year as a predictor by classification method on the sensitivity of a pants-on-fire label. The median is a black triangle, and the mean is a grey circle.

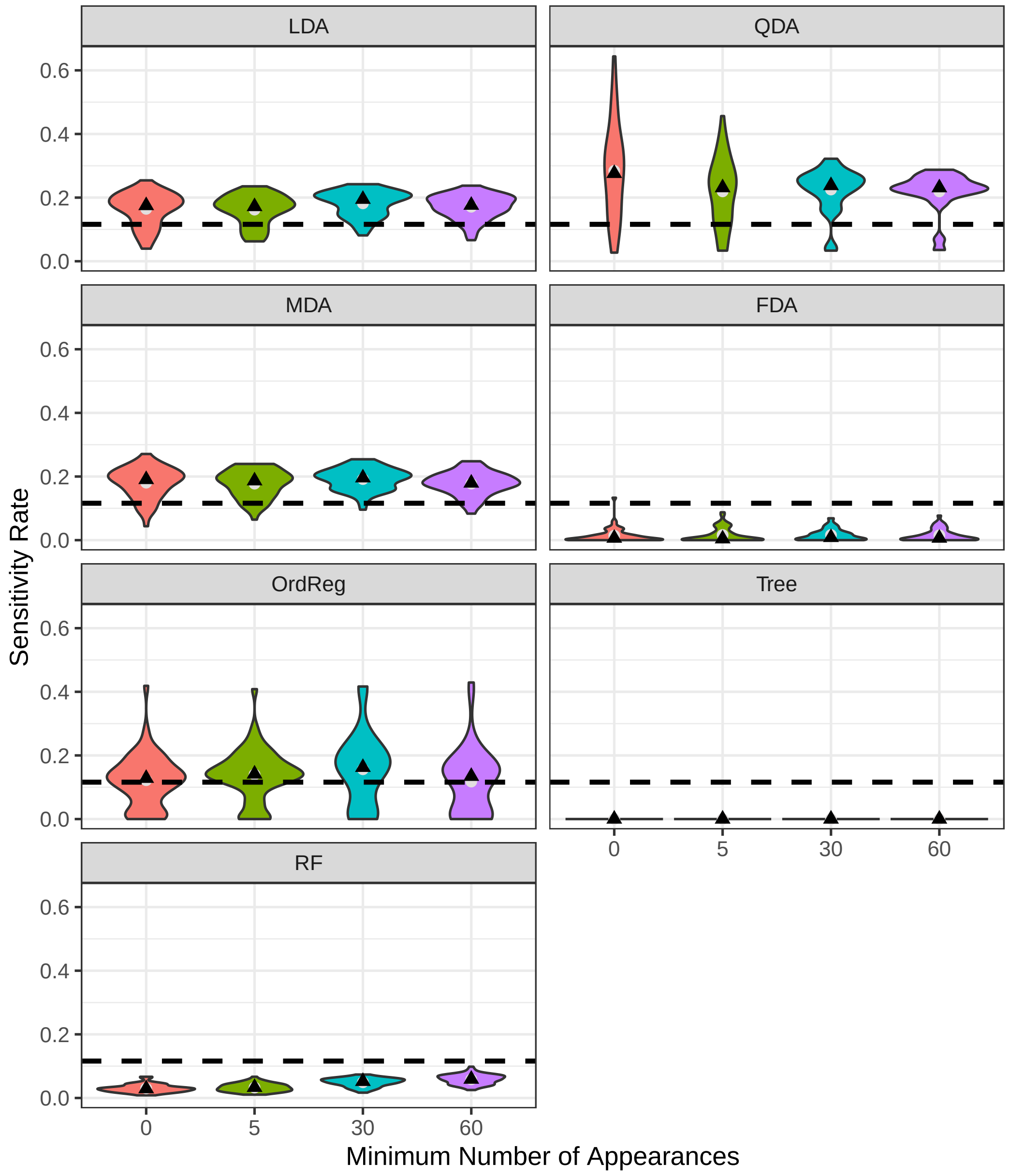

Figure B.32: $C \times F$, F-ratio = 16.803. The effect of filtering threshold by classification method on the sensitivity of a pants-on-fire label. The median is a black triangle, and the mean is a grey circle.

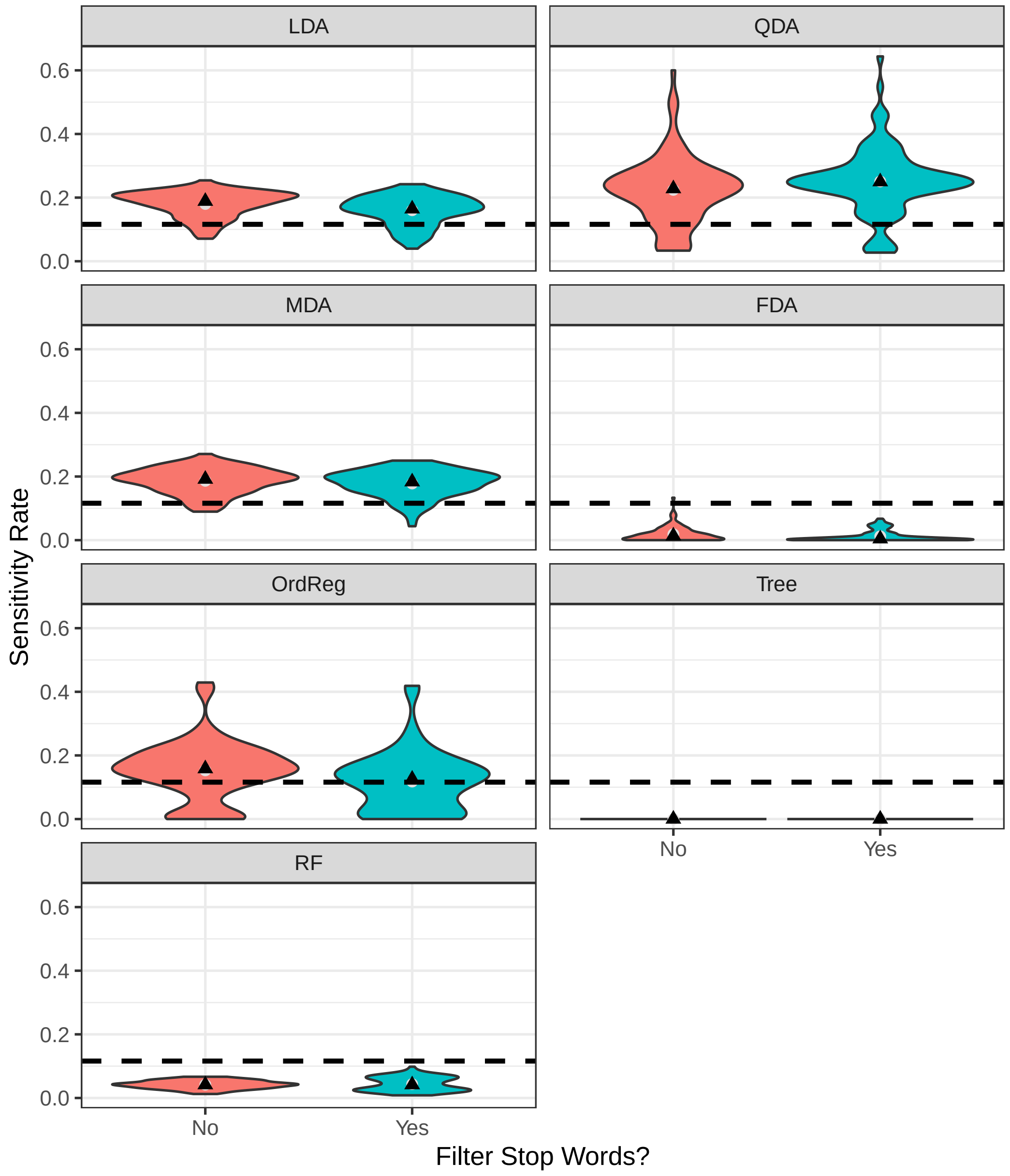

Figure B.33: $C \times S$, F-ratio = 20.903. The effect of filtering stop words by classification method on the sensitivity of a pants-on-fire label. The median is a black triangle, and the mean is a grey circle.

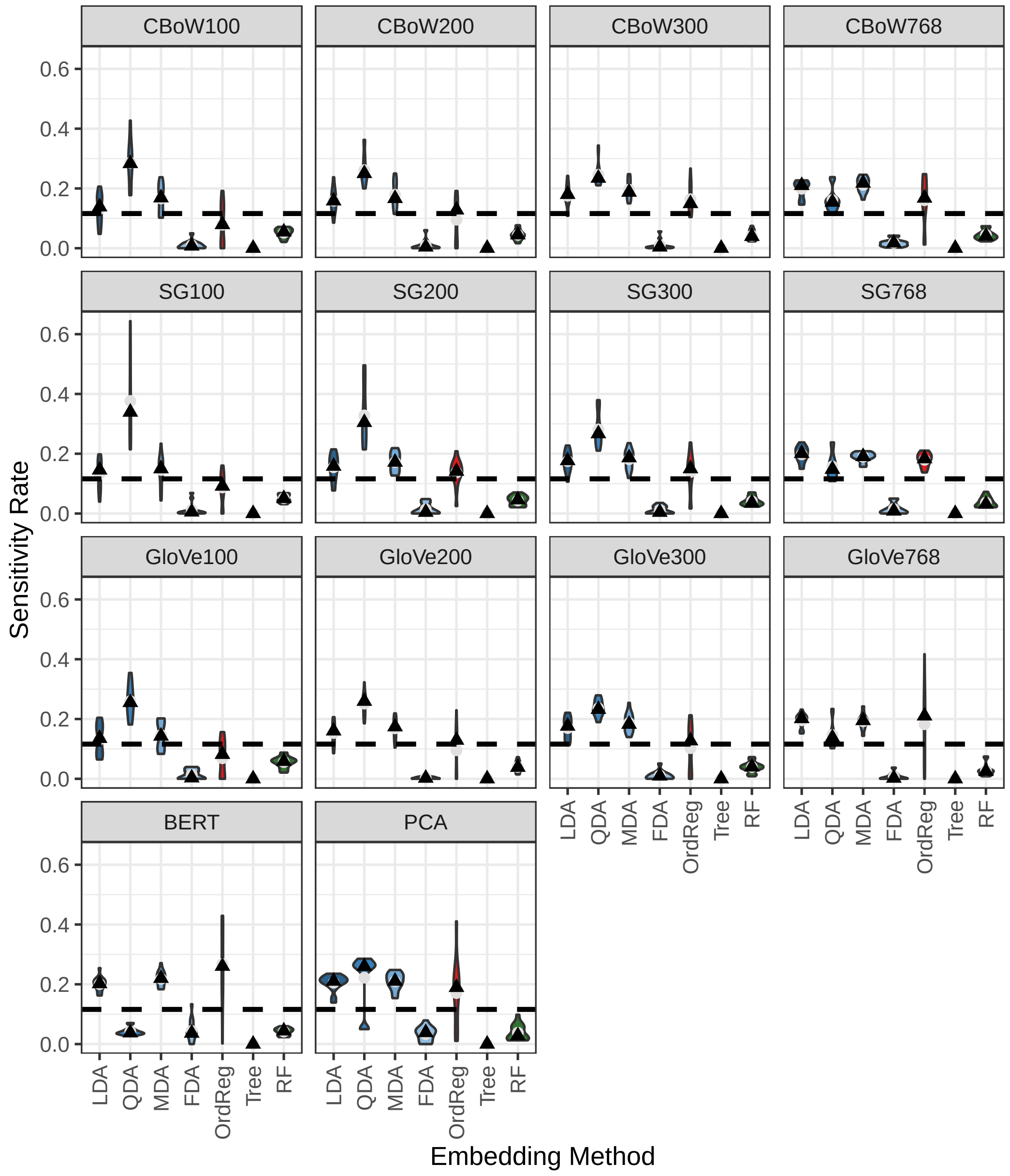

Figure B.34: $C \times E$, F-ratio = 32.143. The effect of classification method by embedding method on the sensitivity of the pants-on-fire label. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

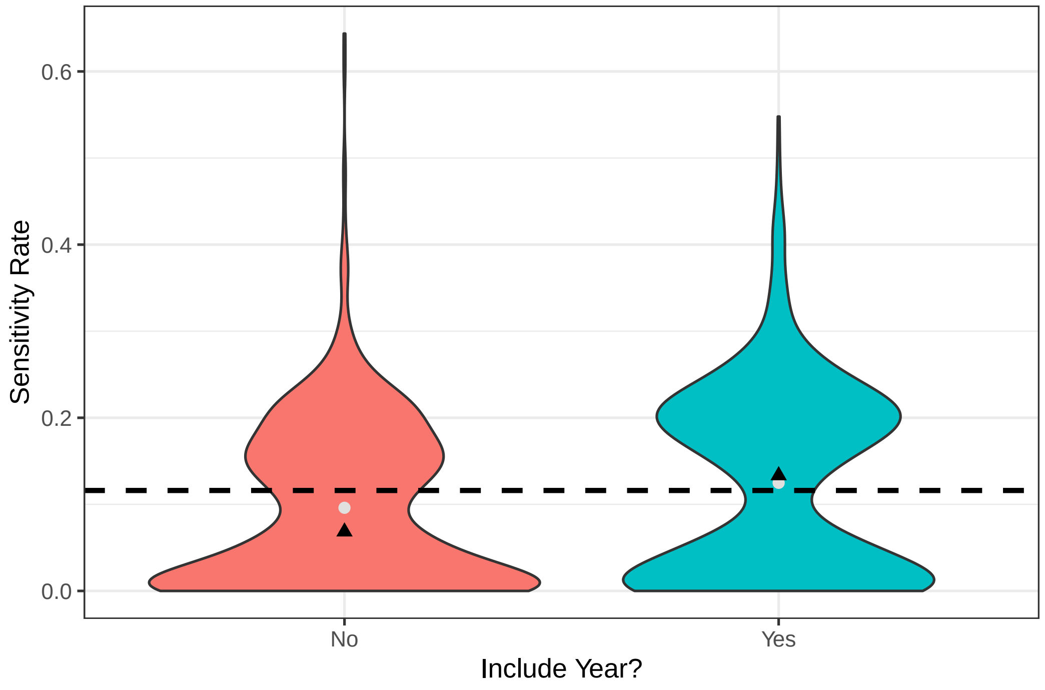

Figure B.35: R, F-ratio = 355.736. The effect of including year as a predictor on the sensitivity of the pants-on-fire label. The median is a black triangle, and the mean is a grey circle.

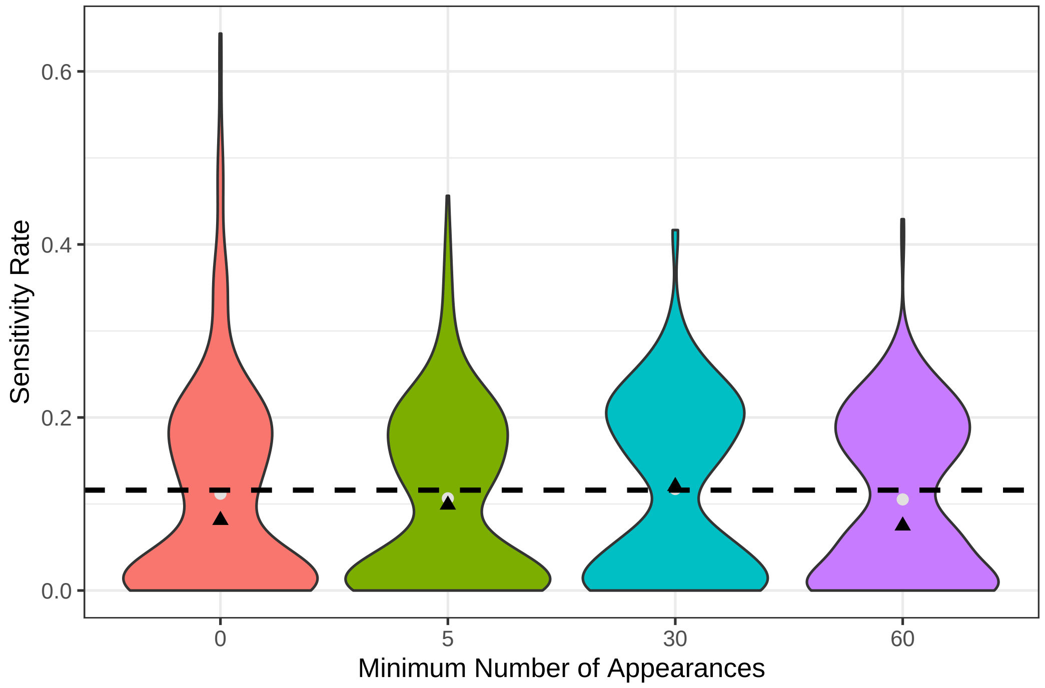

Figure B.36: F, F-ratio = 9.366. The effect of filtering threshold on the sensitivity of the pants-on-fire label. The median is a black triangle, and the mean is a grey circle.

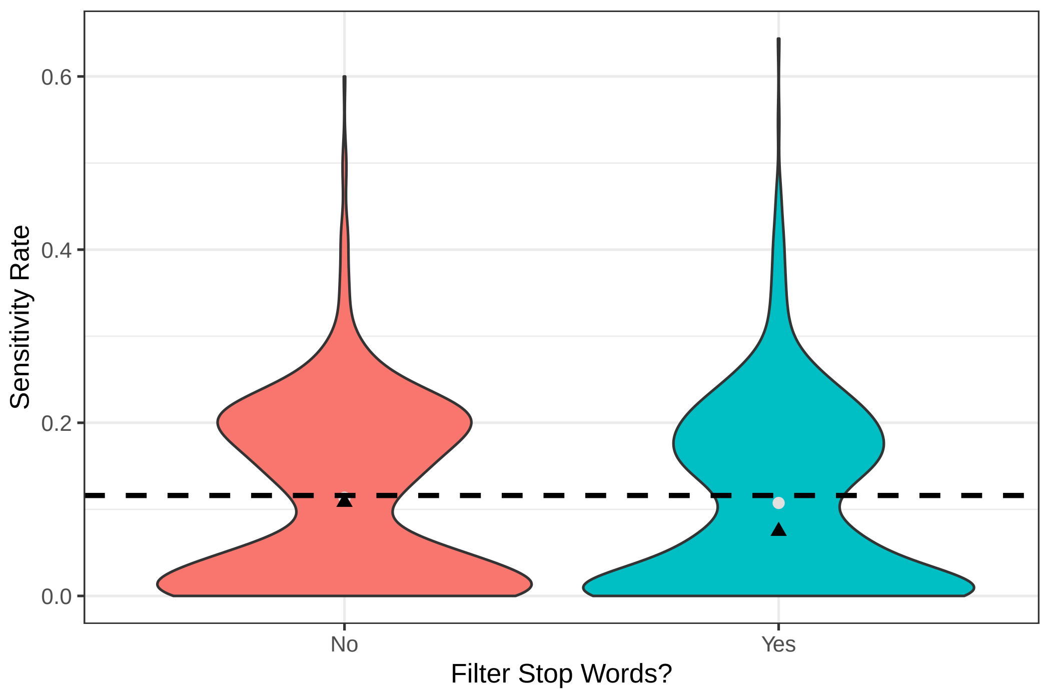

Figure B.37: S, F-ratio = 19.809. The effect of filtering stop words on the sensitivity of the pants-on-fire label. The median is a black triangle, and the mean is a grey circle.

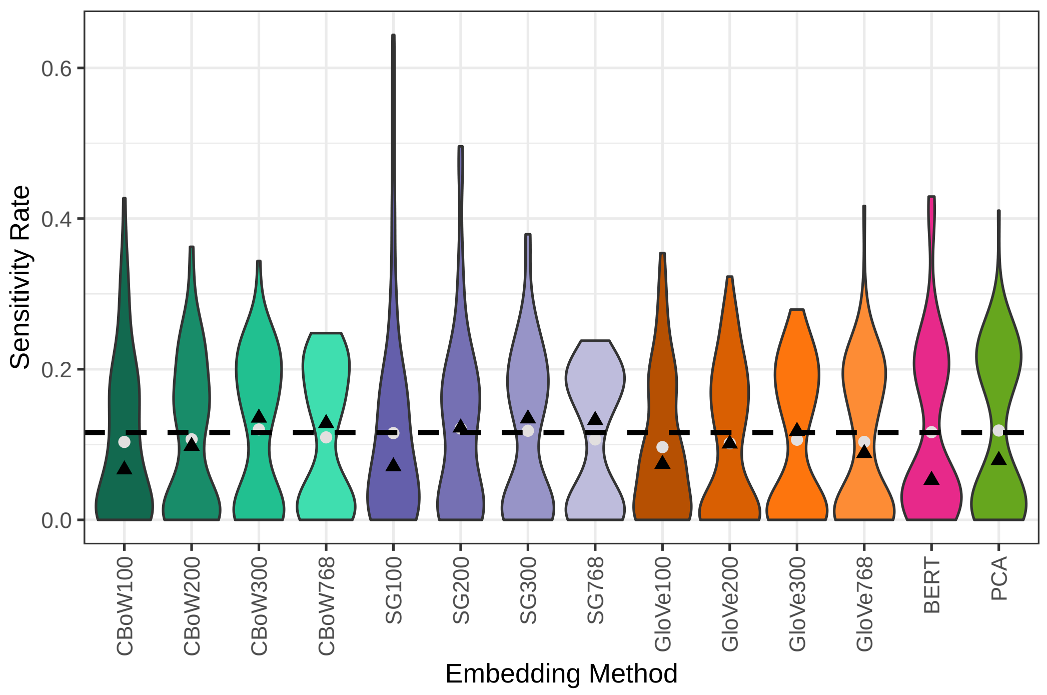

Figure B.38: E, F-ratio = 9.524. The effect of embedding method on the sensitivity of a pants-on-fire label. Word2Vec Continuous Bag of Words are in the blue-green shades, and Word2Vec Skip-Gram are in the blue-purple shades. GloVe is in the brown-orange shades. BERT is in pink, and PCA is in green. The median is a black triangle, and the mean is a grey circle.

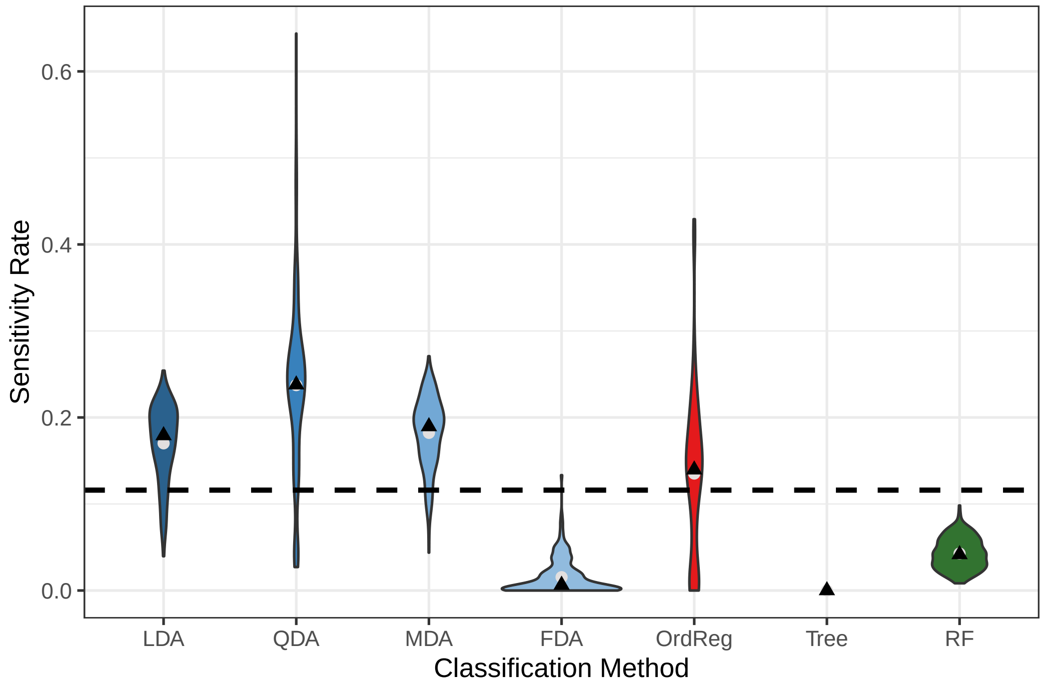

Figure B.39: C, F-ratio = 2080.956. The effect of classification method on the sensitivity of a pants-on-fire label. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

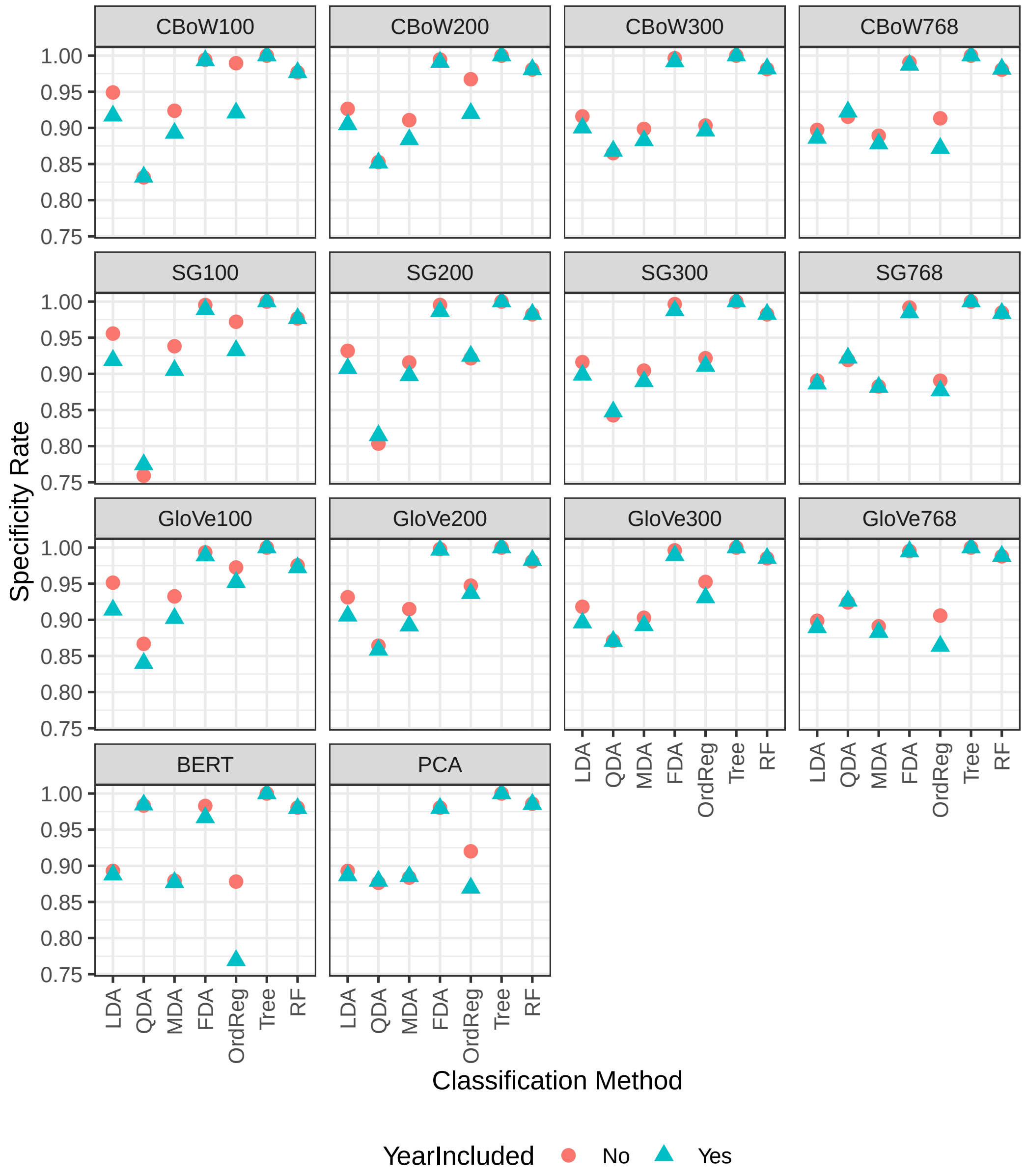

Figure B.40: $C \times E \times R$, F-ratio = 2.029. The effect of classification method, embedding method, and inclusion of year as a predictor on the specificity of a pants-on-fire label. Only the means are plotted here. Violin plots are not used due to the unbalanced number of combinations between the levels since there were computational issues. There are fewer combinations observed when year is included as a predictor.

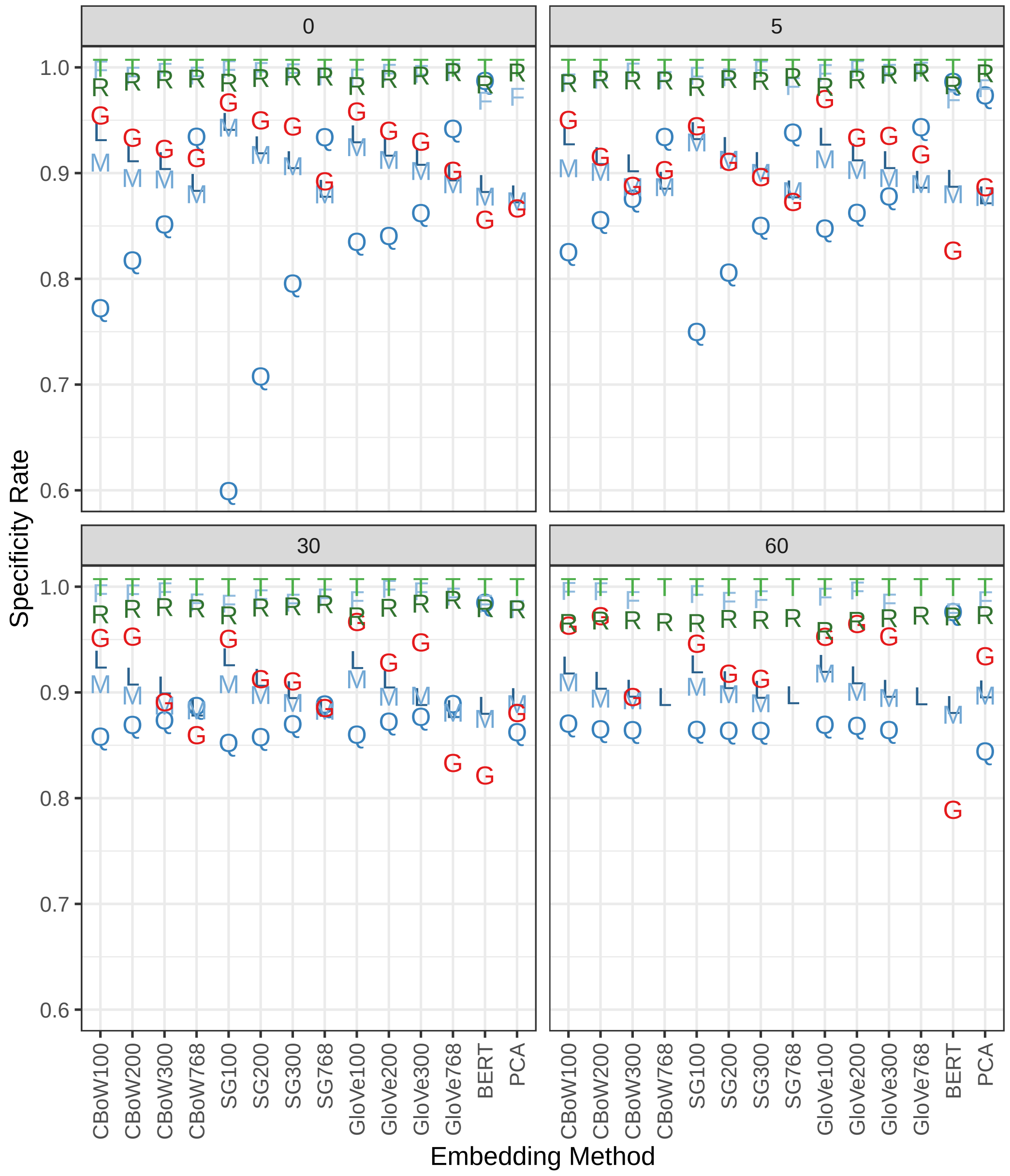

Figure B.41: $C \times E \times F$, F-ratio = 3.754. The effect of classification method, embedding method, and filtering threshold on the specificity of a pants-on-fire label. Each classification method is represented by a letter. T is for classification tree, and G is for ordinal regression. All others are based on the first letter of the method name. Only the means are plotted here. Violin plots are not used due to the number of levels of filtering threshold.

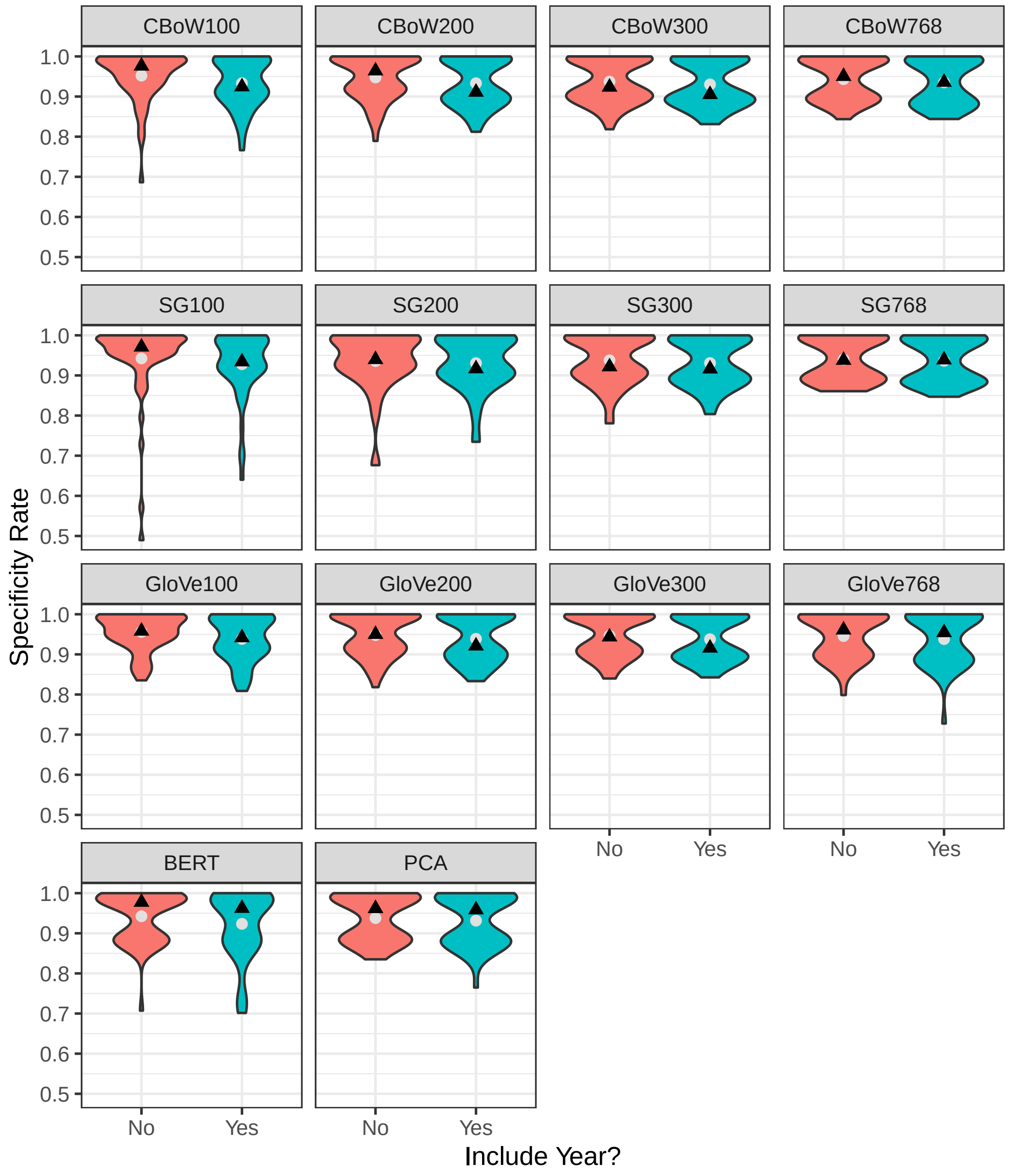

Figure B.42: $E \times R$, F-ratio $= 2.166$. The effect of including year as a predictor by embedding method on the specificity of a pants-on-fire label. The median is a black triangle, and the mean is a grey circle.

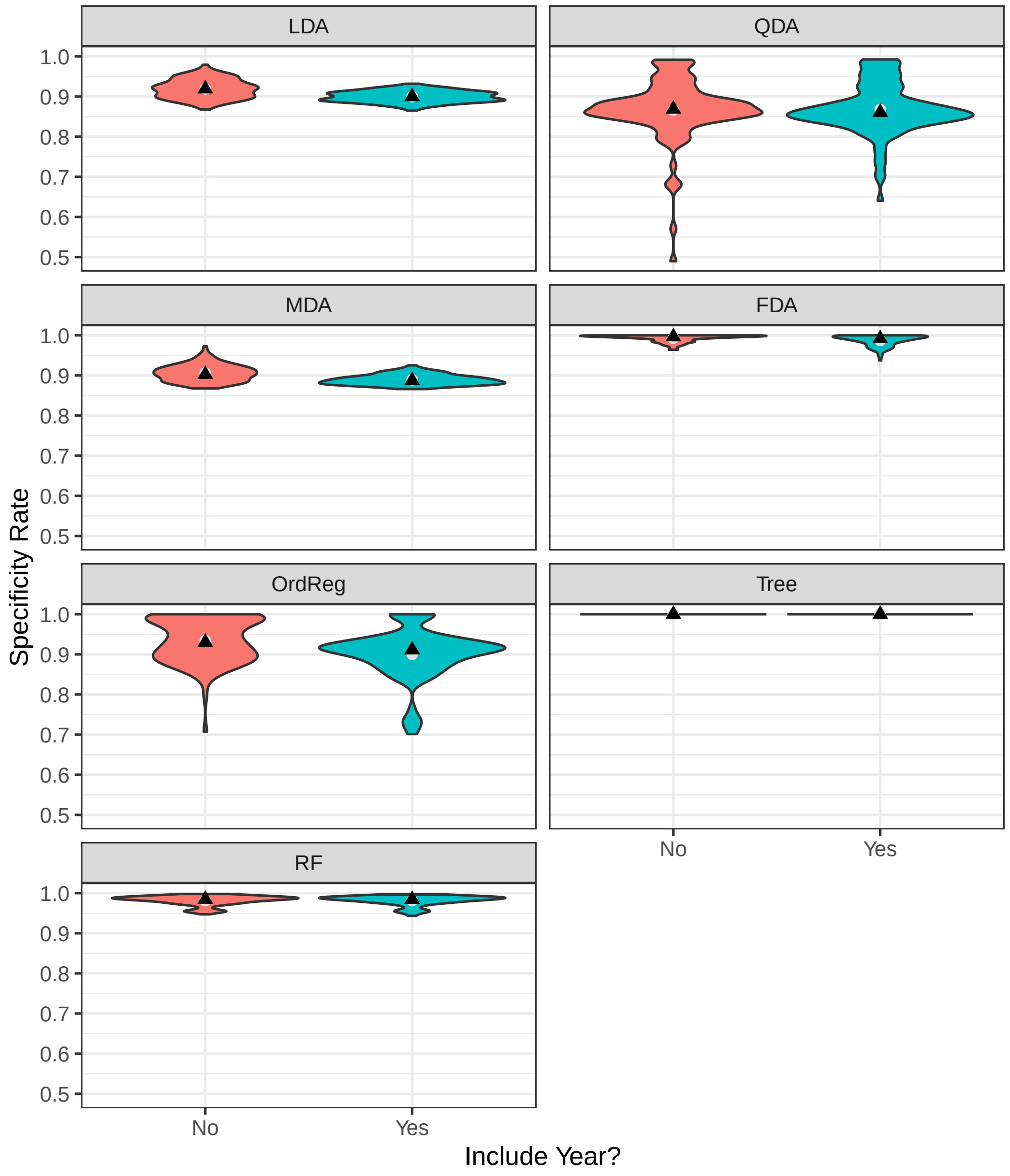

Figure B.43: $C \times R$, F-ratio = 27.138. The effect of including year as a predictor by classification method on the specificity of a pants-on-fire label. The median is a black triangle, and the mean is a grey circle.

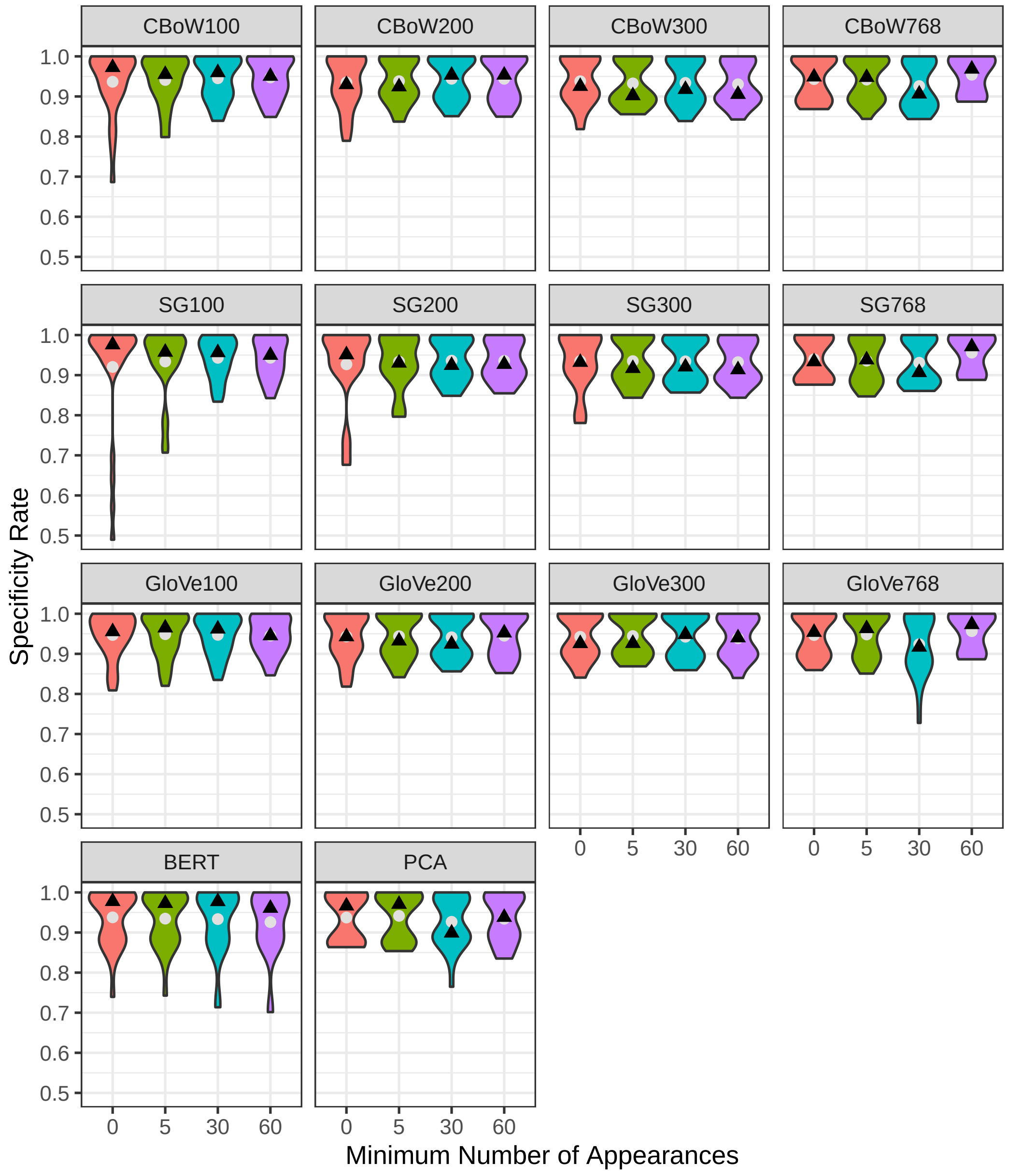

Figure B.44: $E \times F$, F-ratio = 2.828. The effect of filtering threshold by embedding method on the specificity of a pants-on-fire label. The median is a black triangle, and the mean is a grey circle.

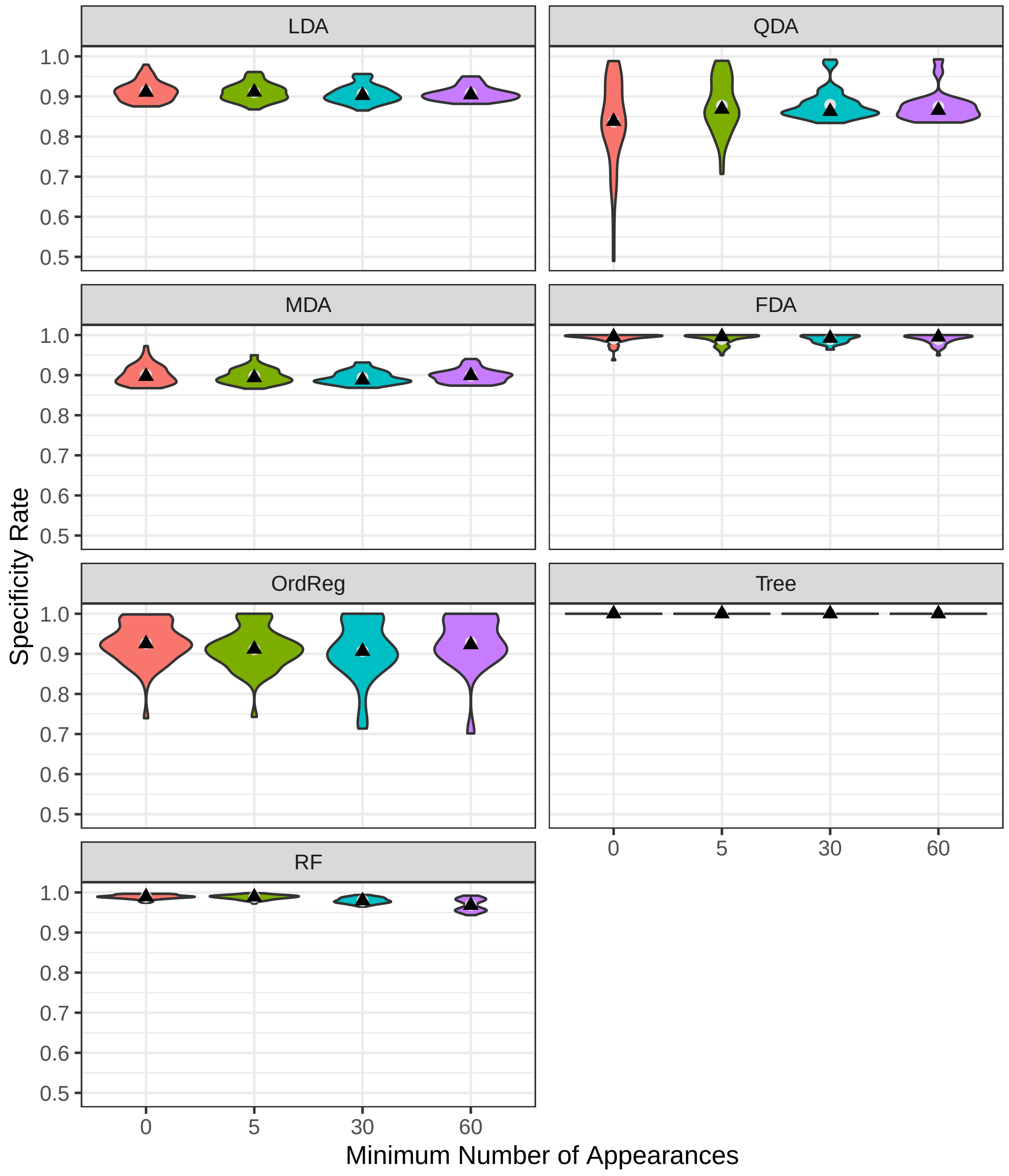

Figure B.45: $C \times F$, F-ratio = 17.852. The effect of filtering threshold by classification method on the specificity of a pants-on-fire label. The median is a black triangle, and the mean is a grey circle.

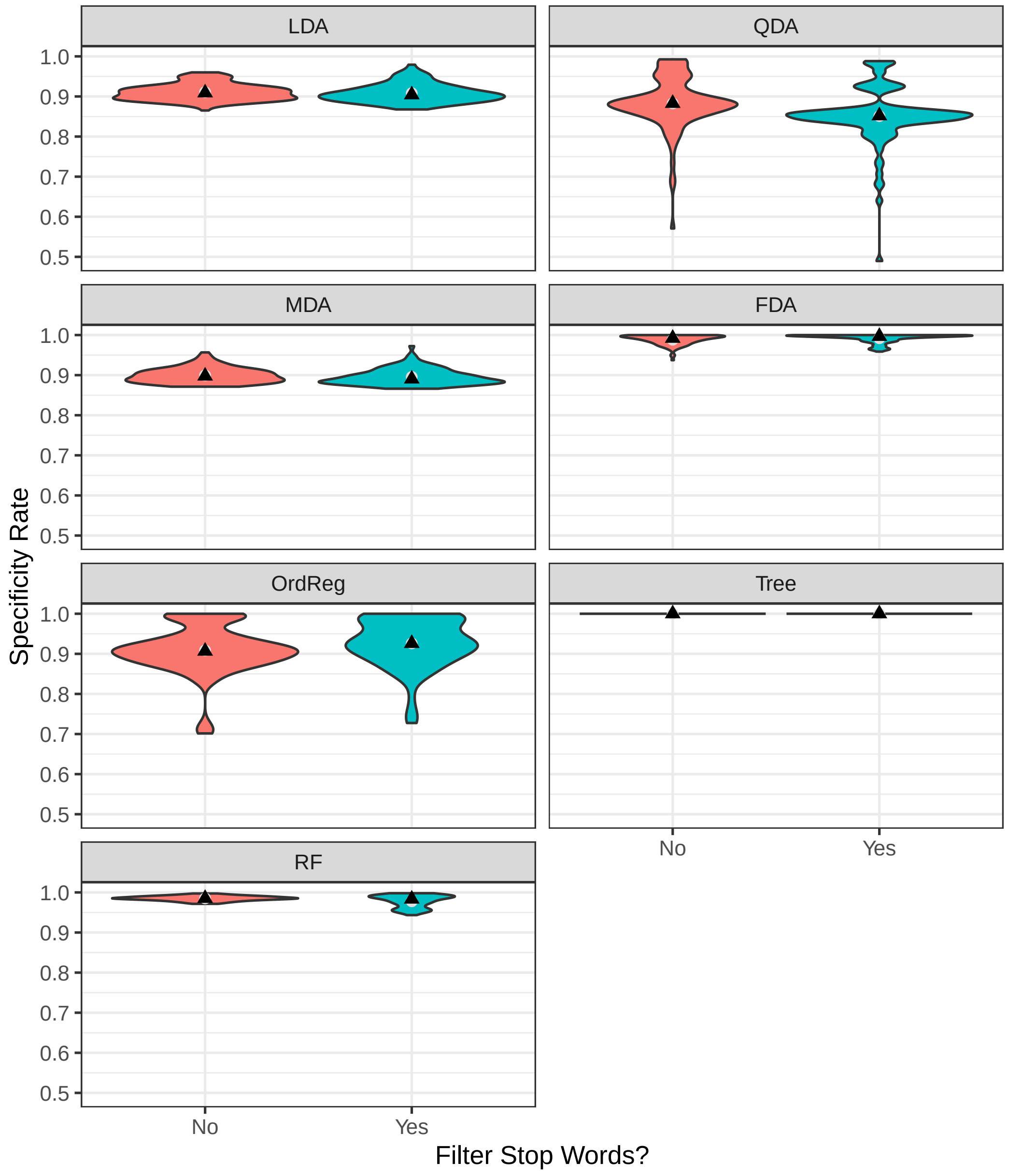

Figure B.46: $C \times S$, F-ratio = 32.135. The effect of filtering stop words by classification method on the specificity of a pants-on-fire label. The median is a black triangle, and the mean is a grey circle.

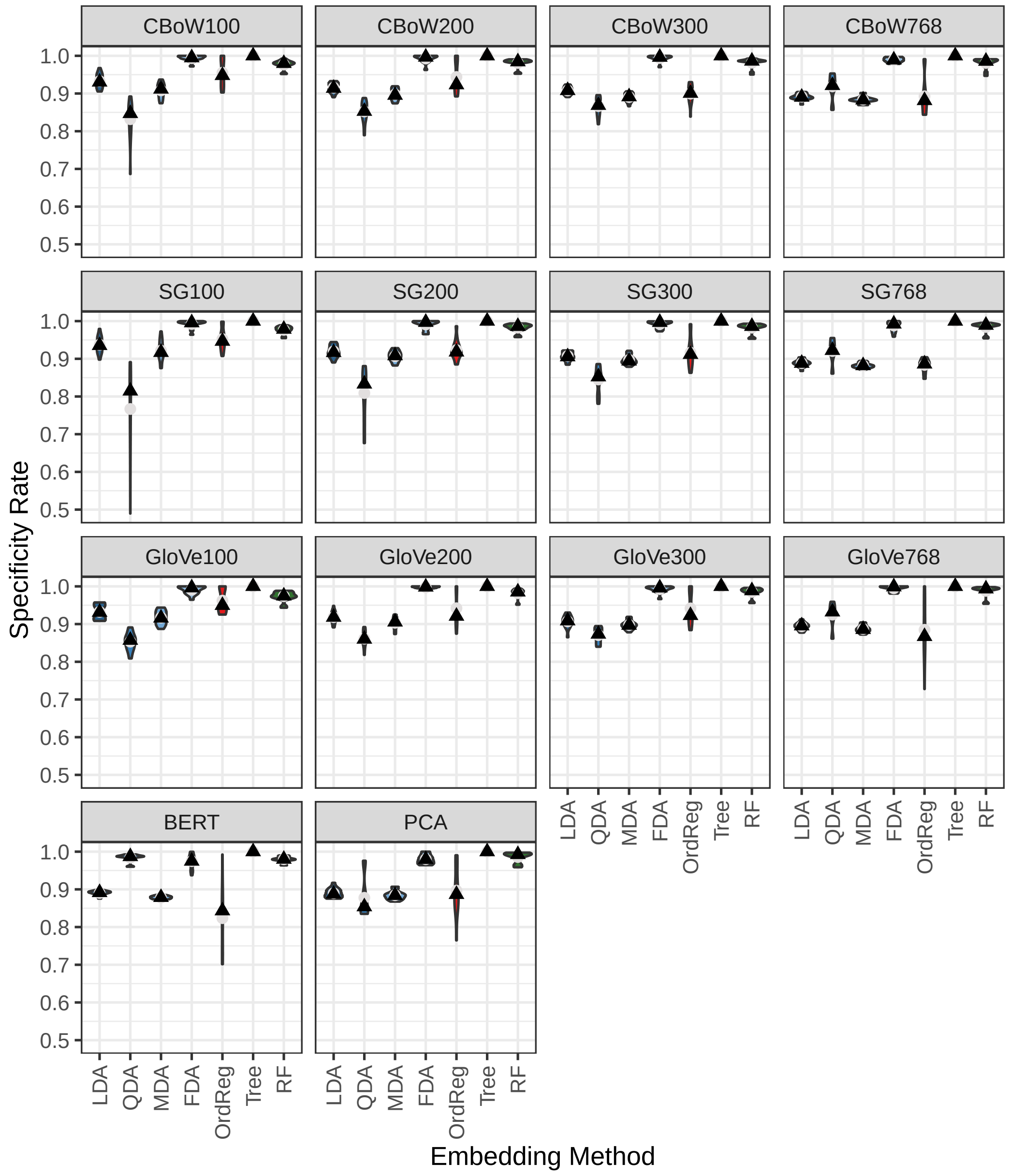

Figure B.47: $C \times E$, F-ratio = 35.297. The effect of classification method by embedding method on the specificity of the pants-on-fire label. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

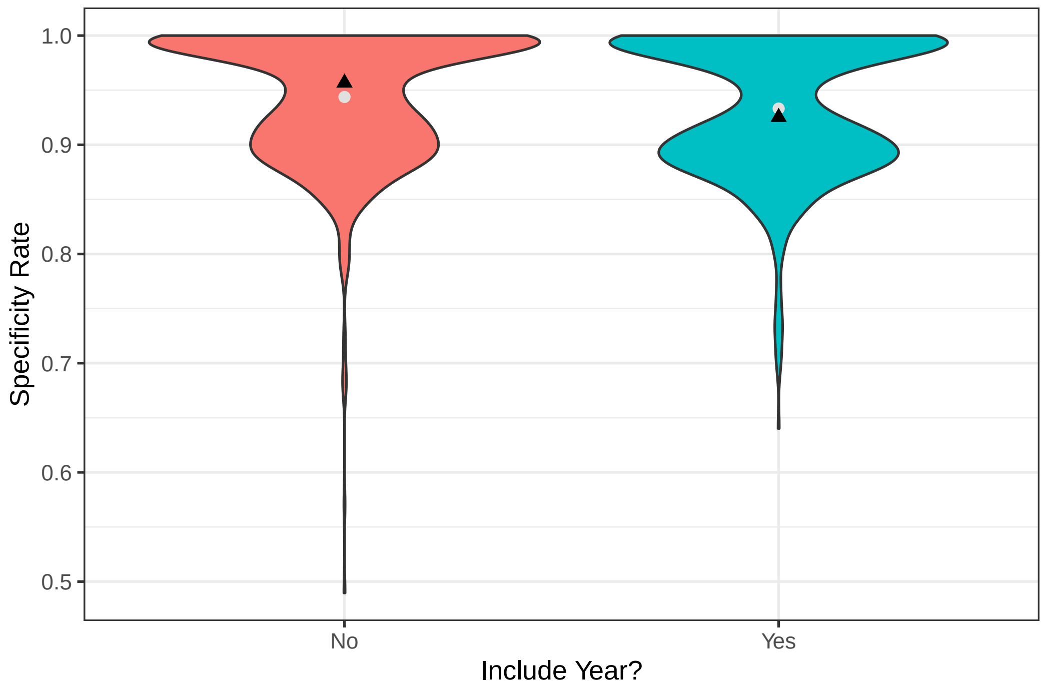

Figure B.48: *R*, F-ratio = 122.45. The effect of including year as a predictor on the specificity of the pants-on-fire label. The median is a black triangle, and the mean is a grey circle.

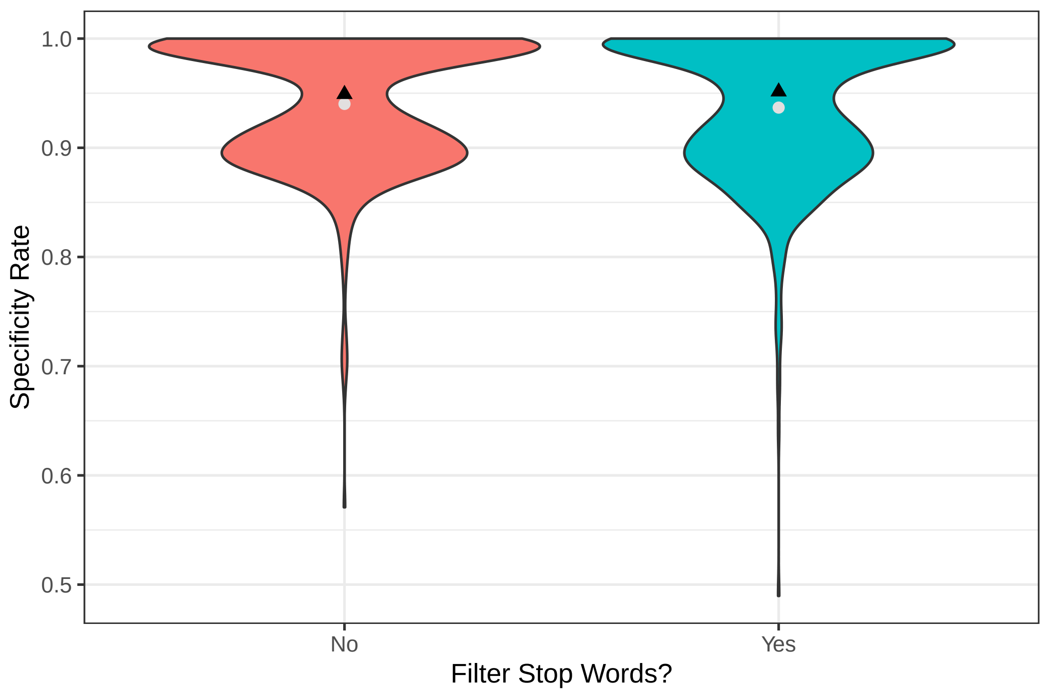

Figure B.49: S, F-ratio = 9.214. The effect of filtering stop words on the specificity of the pants-on-fire label. The median is a black triangle, and the mean is a grey circle.

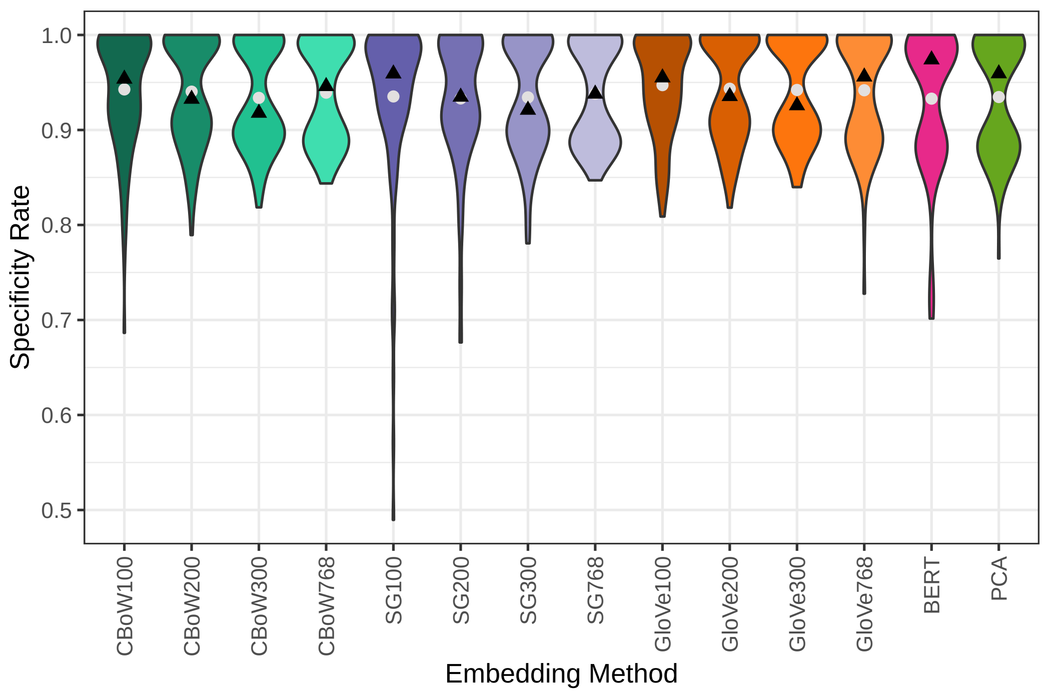

Figure B.50: E, F-ratio = 8.3. The effect of embedding method on the specificity of a pants-on-fire label. Word2Vec Continuous Bag of Words are in the blue-green shades, and Word2Vec Skip-Gram are in the blue-purple shades. GloVe is in the brown-orange shades. BERT is in pink, and PCA is in green. The median is a black triangle, and the mean is a grey circle.

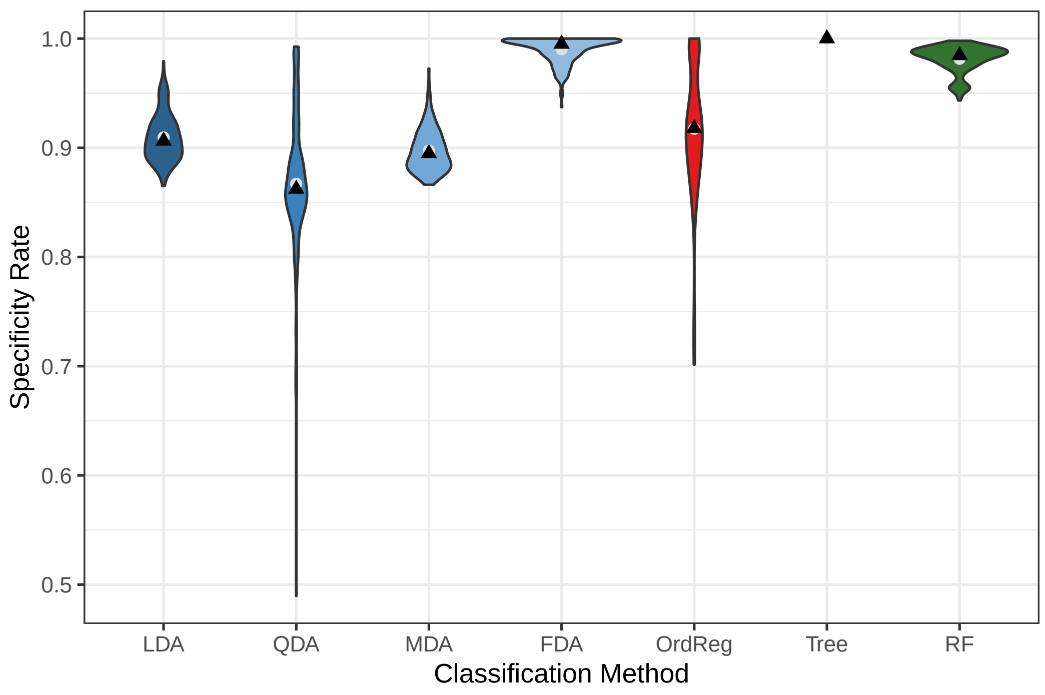

Figure B.51: C, F-ratio = 2080.956. The effect of classification method on the specificity of a pants-on-fire label. Discriminant analyses are in blue. Logistic regression is red, and tree-based methods are in green. The median is a black triangle, and the mean is a grey circle.

Appendix C

Code

The following code was ran using the Crane server at the Holland Computing Center, located at the University of Nebraska-Lincoln.

Binary Classification

Bag of Words

```
library(tidyverse)
library(tidytext)
library(textdata)
library(text2vec)
library(keras)
library(uwot)
library(tensorflow)
library(nomclust)
library(ggplot2)
library(caret)
library(doParallel)

set.seed(121565)

############## COMBINING DATA SETS ###########################
```

```
rawdata_train <- read.csv(file = "/work/wtp/jhaus4/fnn_train.csv",
                          header = T,
                          stringsAsFactors = F)
rawdata_test <- read.csv(file = "/work/wtp/jhaus4/fnn_test.csv",
                         header = T,
                         stringsAsFactors = F)
rawdata_valid <- read.csv(file = "/work/wtp/jhaus4/fnn_dev.csv",
                         header = T,
                         stringsAsFactors = F)
rawdata <- rbind(rawdata_train, rawdata_valid, rawdata_test)

data <- rawdata %>%
  mutate(newsTextClean=gsub('[[:punct:]]+', '',
                       gsub('\\\\n|\\.|\\,|\\;',' ',
                       gsub("[^[:alnum:]]", " ",
                       (substr(statement,1,nchar(statement)-1))))))
#For all lowercase
#tolower(substr(statement,1,nchar(statement)-1))

news_tokens <- data %>% select(id, label_fnn, newsTextClean) %>%
  unnest_tokens(word, newsTextClean, to_lower = FALSE) %>%
  #Use just "[a-z']+" for lower case letters
  #Change to "[[:alnum:]]+" for numbers to be included
  #Change to "[A-z']+" for all cases
  mutate(word = str_extract(word, "[[:alnum:]]+"))
news_tokens <- news_tokens %>% drop_na(word)
#news_tokens <- news_tokens %>% anti_join(stop_words)

news_new <- news_tokens %>% group_by(word) %>%
  mutate(token_freq = n()) %>%
  filter(token_freq >=60) %>%
  group_by(id, label_fnn) %>%
  summarise(newsTextClean = str_c(word, collapse = " "))

#Getting the labels
y <- news_new %>% select(label_fnn) %>% pull() %>% as.array()
y <- as.factor(y)
levels(y)
y <- ifelse(y == "fake", 0, 1)

it <- itoken(news_new$newsTextClean,
                  tokenizer = word_tokenizer,
                  ids = news_new$id,
                  progressbar = T, tolower = FALSE)
vocab <- create_vocabulary(it)
vocab <- prune_vocabulary(vocab)
vectorizer <- vocab_vectorizer(vocab)
news_new <- news_new %>% select(newsTextClean) %>% pull()

#Vectorize tokens - token receiving a unique integer
tokenizer <- text_tokenizer(lower = FALSE) %>%
              fit_text_tokenizer(news_new)

#put integers in a sequence
sequences<- texts_to_sequences(tokenizer, news_new)

##########################Document-term matrix all########################
un.words <- length(tokenizer$word_index)
docs <- length(sequences)
dtm <- matrix(0, nrow = docs, ncol = un.words)
for (i in 1:docs){
  word.vec <- sequences[[i]]
  c <- length(sequences[[i]])
  a <- 0
  for (j in 1:c){
    a <- a + 1
    b <- word.vec[a]
    dtm[i,b] <- dtm[i,b] + 1
  }
}

dtm_y <- cbind(dtm, y)
dtm_y <- as.data.frame(dtm_y)

words <- unlist(tokenizer$index_word, use.names = TRUE)
words <- c(words, "PolitiFactRating")

colnames(dtm_y) <- words



trainIndex <- createDataPartition(dtm_y$PolitiFactRating,
                                  p = .8, list = F, times = 1)

dtm_train_y <- dtm_y[trainIndex,]
dtm_test_y <- dtm_y[-trainIndex,]

mean(dtm_train_y$PolitiFactRating)
mean(dtm_test_y$PolitiFactRating)

dtm_train_y$PolitiFactRating <- as.factor(dtm_train_y$PolitiFactRating)
dtm_test_y$PolitiFactRating <- as.factor(dtm_test_y$PolitiFactRating)
```

```
###################### BoW CV ################

train.control <- trainControl(method = "cv", number = 10)
cl <- makePSOCKcluster(5)
registerDoParallel(cl)

#Logistic regression directly on DTM
start_time <- Sys.time()
model <- train(PolitiFactRating ~ ., data = dtm_train_y,
               trControl = train.control, method = "glm",
                family="binomial", control = list(maxit = 50))
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions, dtm_test_y$PolitiFactRating)

#Linear Discriminant Analysis directly on DTM
start_time <- Sys.time()
model <- train(PolitiFactRating ~ ., data = dtm_train_y,
               trControl = train.control, method = "lda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions, dtm_test_y$PolitiFactRating)

#Quadratic Discriminant Analysis directly on DTM
```

```
#start_time <- Sys.time()
#model <- train(PolitiFactRating ~ ., data = dtm_train_y,
#               trControl = train.control, method = "qda")
#end_time <- Sys.time()
#glm.time <- end_time - start_time
#glm.time
#model
#predictions <- model %>% predict(dtm_test_y)
#confusionMatrix(predictions, dtm_test_y$PolitiFactRating)

#Mixture Discriminant Analysis directly on DTM
library(mda)
#start_time <- Sys.time()
#model <- train(PolitiFactRating ~ ., data = dtm_train_y,
#               trControl = train.control, method = "mda")
#end_time <- Sys.time()
#glm.time <- end_time - start_time
#glm.time
#model
#predictions <- model %>% predict(dtm_test_y)
#confusionMatrix(predictions, dtm_test_y$PolitiFactRating)

#Flexible Discriminant Analysis directly on DTM
#library(earth)
#start_time <- Sys.time()
#model <- train(PolitiFactRating ~ ., data = dtm_train_y,
#               trControl = train.control, method = "fda")
#end_time <- Sys.time()
#glm.time <- end_time - start_time
#glm.time
#model
```

```
#predictions <- model %>% predict(dtm_test_y)
#confusionMatrix(predictions, dtm_test_y$PolitiFactRating)

#Classification trees directly on DTM
library(rpart)
start_time <- Sys.time()
cpGrid <- expand.grid(.cp=seq(0.01, 0.5,0.01))
model <- train(PolitiFactRating ~ ., data = dtm_train_y,
               trControl = train.control, method = "rpart",
               tuneGrid = cpGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions, dtm_test_y$PolitiFactRating)


### Random Forest
library(randomForest)
start_time <- Sys.time()
mtry <- floor(sqrt(ncol(dtm_train_y)-1))
if (mtry > 10){
  a <- mtry - 10
} else {a <- mtry}
b <- mtry + 10
mtryGrid <- expand.grid(.mtry=seq(a, b, 1))
model <- train(PolitiFactRating ~ ., data = dtm_train_y,
               trControl = train.control, method = "rf",
               tuneGrid = mtryGrid)
end_time <- Sys.time()
```

```
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions, dtm_test_y$PolitiFactRating)
```

BERT

```
library(tidyverse)
library(tidytext)
library(textdata)
library(text2vec)
library(keras)
library(uwot)
library(tensorflow)
library(nomclust)
library(ggplot2)
library(word2vec)
library(caret)
library(doParallel)

set.seed(121565)

rawdata_train <- read.csv(file = "/work/wtp/jhaus4/fnn_train.csv",
                          header = T,
                          stringsAsFactors = F)
rawdata_test <- read.csv(file = "/work/wtp/jhaus4/fnn_test.csv",
                         header = T,
                         stringsAsFactors = F)
rawdata_valid <- read.csv(file = "/work/wtp/jhaus4/fnn_dev.csv",
                          header = T,
                         stringsAsFactors = F)
rawdata <- rbind(rawdata_train, rawdata_valid, rawdata_test)

data <- rawdata %>%
  mutate(newsTextClean=gsub('[[:punct:]]+', '',
                       gsub('\\\\n|\\.|\\,|\\;',' ',
                       gsub("[^[:alnum:]]", " ",
                       (substr(statement,1,nchar(statement)-1))))))
#For all lowercase
#tolower(substr(statement,1,nchar(statement)-1))

news_tokens <- data %>% select(id, label_fnn, newsTextClean) %>%
  unnest_tokens(word, newsTextClean, to_lower = FALSE) %>%
  #Use just "[a-z']+" for lower case letters
  #Change to "[[:alnum:]]+" for numbers and cases to be included
  #Change to "[A-z']+" for all cases
  mutate(word = str_extract(word, "[[:alnum:]]+"))
news_tokens <- news_tokens %>% drop_na(word)
#news_tokens <- news_tokens %>% anti_join(stop_words)

news_new <- news_tokens %>% group_by(word) %>%
  mutate(token_freq = n()) %>%
  filter(token_freq >=60) %>%
  group_by(id, label_fnn) %>%
  summarise(newsTextClean = str_c(word, collapse = " "))

trainIndex <- createDataPartition(news_new$label_fnn,
                                  p = .8, list = F, times = 1)
train.control <- trainControl(method = "cv", number = 10)
cl <- makePSOCKcluster(5)
registerDoParallel(cl)
```

```
news_new_train <- news_new[trainIndex,]
news_new_test <- news_new[-trainIndex,]

#write.csv(news_new_train,
          "fnntrain60NumbersUpperStopwords.csv",
          row.names = FALSE)
#write.csv(news_new_test,
          "fnntest60NumbersUpperStopwords.csv",
          row.names = FALSE)

bert_train <- read.csv(file =
                      "BERT_Train_60_Numbers_Upper_Stopwords.csv",
                      header = F)
dim(bert_train)
bert_train <- bert_train[-1,]
dim(bert_train)

bert_test <- read.csv(file =
            "BERT_Test_60_Numbers_Upper_Stopwords.csv",
            header = F)
dim(bert_test)
bert_test <- bert_test[-1,]
dim(bert_test)

#Getting the labels
y_train <- news_new_train %>% select(label_fnn) %>% pull() %>% as.array()
y_test <- news_new_test %>% select(label_fnn) %>% pull() %>% as.array()
y_train <- as.factor(y_train)
y_test <- as.factor(y_test)
levels(y_train)
levels(y_test)
y_train <- ifelse(y_train == "fake", 0, 1)
y_test <- ifelse(y_test == "fake", 0, 1)

dtm_train_y <- cbind(bert_train, y_train)
dtm_train_y <- as.data.frame(dtm_train_y)
dtm_test_y <- cbind(bert_test, y_test)
dtm_test_y <- as.data.frame(dtm_test_y)

dtm_train_y$y_train <- as.factor(dtm_train_y$y_train)
dtm_test_y$y_test <- as.factor(dtm_test_y$y_test)

##### CV DIM#####



#Logistic regression directly on DTM
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "glm",
               family="binomial", control = list(maxit = 50))
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)



#Linear Discriminant Analysis directly on DTM
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "lda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)


#Quadratic Discriminant Analysis directly on DTM
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "qda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)


#Mixture Discriminant Analysis directly on DTM
library(mda)
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "mda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
```

```
confusionMatrix(predictions,dtm_test_y$y_test)


#Flexible Discriminant Analysis directly on DTM
library(earth)
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "fda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)


#Classification trees directly on DTM
library(rpart)
start_time <- Sys.time()
cpGrid <- expand.grid(.cp=seq(0.01, 0.5,0.01))
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "rpart",
               tuneGrid = cpGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

### Random Forest
```

```
library(randomForest)
start_time <- Sys.time()
mtry <- floor(sqrt(ncol(dtm_train_y)-1))
if (mtry > 10){
  a <- mtry - 10
} else {a <- mtry}
b <- mtry + 10
mtryGrid <- expand.grid(.mtry=seq(a, b, 1))
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "rf",
               tuneGrid = mtryGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)
```

BERT, Python portion (Reimer, 2022)

```
import tensorflow as tf
import torch
import pandas as pd

df_train = pd.read_csv("fnntrain60NumbersUpperStopwords.csv")
df_test = pd.read_csv("fnntest60NumbersUpperStopwords.csv")

#Get the lists of sentences and their labels.
sentences_train = df_train.newsTextClean.values
sentences_test = df_test.newsTextClean.values
```

```
from sentence_transformers import SentenceTransformer, models

#change to 'bert-base-cased' to tell difference between
#upper and lower case
word_embedding_model = models.Transformer('bert-base-uncased',
max_seq_length=120)
pooling_model = models.Pooling(word_embedding_model.
get_word_embedding_dimension())

sbert_model = SentenceTransformer(modules=[word_embedding_model,
pooling_model])

document_train = sbert_model.encode(sentences_train)
document_test = sbert_mode.encode(sentences_test)

df_train.to_csv("BERT_Train_60_Numbers_Upper_Stopwords.csv")
df_test.to_csv("BERT_Test_60_Numbers_Upper_Stopwords.csv")
```

GloVe

```
library(tidyverse)
library(tidytext)
library(textdata)
library(text2vec)
library(keras)
library(uwot)
library(tensorflow)
library(nomclust)
library(ggplot2)
library(word2vec)
library(caret)
library(doParallel)
library(pROC)

set.seed(121565)

############## COMBINING DATA SETS ############################

rawdata_train <- read.csv(file = "DataFakeNewsNets/fnn_train.csv",
                          header = T,
                          stringsAsFactors = F)
rawdata_test <- read.csv(file = "DataFakeNewsNets/fnn_test.csv",
                         header = T,
                         stringsAsFactors = F)
rawdata_valid <- read.csv(file = "DataFakeNewsNets/fnn_dev.csv",
                          header = T,
                          stringsAsFactors = F)
rawdata <- rbind(rawdata_train, rawdata_valid, rawdata_test)

data <- rawdata %>%
  mutate(newsTextClean=gsub('[[:punct:]]+', '',
                      gsub('\\\\n|\\.|\\,|\\;',' ',
                      gsub("[^[:alnum:]]", " ",
                      tolower(substr(statement,1,nchar(statement)-1))))))
data <- data %>% separate(date, c("Year", "Month", "Day","Junk"), sep = "-")
#For all lowercase
#tolower(substr(statement,1,nchar(statement)-1))

news_tokens <- data %>% select(id, label_fnn, Year, newsTextClean) %>%
  unnest_tokens(word, newsTextClean, to_lower = FALSE) %>%
  #Use just "[a-z']+" for lower case letters
  #Change to "[[:alnum:]]+" for numbers and cases to be included
  #Change to "[A-z']+" for all cases
  mutate(word = str_extract(word, "[a-z']+"))
news_tokens <- news_tokens %>% drop_na(word)
news_tokens <- news_tokens %>% anti_join(stop_words)

news_new <- news_tokens %>% group_by(word) %>%
  mutate(token_freq = n()) %>%
  filter(token_freq >= 5) %>%
  group_by(id, label_fnn, Year) %>%
  summarise(newsTextClean = str_c(word, collapse = " "))

max.length <- max(str_count(news_new$newsTextClean, "[[:alnum:]]+"))

trainIndex <- createDataPartition(news_new$label_fnn,
                                  p = .8, list = F, times = 1)
train.control <- trainControl(method = "cv", number = 10)

news_new_train <- news_new[trainIndex,]
news_new_test <- news_new[-trainIndex,]



#Getting the labels
y_train <- news_new_train %>% select(label_fnn) %>% pull() %>% as.array()
y_test <- news_new_test %>% select(label_fnn) %>% pull() %>% as.array()
y_train <- as.factor(y_train)
y_test <- as.factor(y_test)
levels(y_train)
levels(y_test)
y_train <- ifelse(y_train == "fake", 0, 1)
y_test <- ifelse(y_test == "fake", 0, 1)
```

```
year_train <- news_new_train %>% select(Year) %>% pull() %>% as.array()
year_test <- news_new_test %>% select(Year) %>% pull() %>% as.array()
year_train <- as.factor(year_train)
year_test <- as.factor(year_test)
levels(year_train)
levels(year_test)

it_train <- itoken(news_new_train$newsTextClean,
                   tokenizer = word_tokenizer,
                   ids = news_new$id,
                   progressbar = T, tolower = FALSE)
it_test <- itoken(news_new_test$newsTextClean,
                  tokenizer = word_tokenizer,
                  ids = news_new$id,
                  progressbar = T, tolower = FALSE)
vocab_train <- create_vocabulary(it_train)
vocab_test <- create_vocabulary(it_test)
vocab_train <- prune_vocabulary(vocab_train)
vocab_test <- prune_vocabulary(vocab_test)
vectorizer_train <- vocab_vectorizer(vocab_train)
vectorizer_test <- vocab_vectorizer(vocab_test)
news_new_train <- news_new_train %>% select(newsTextClean) %>% pull()
news_new_test <- news_new_test %>% select(newsTextClean) %>% pull()

#Vectorize tokens - token receiving a unique integer
tokenizer_train <- text_tokenizer(lower = FALSE) %>%
                    fit_text_tokenizer(news_new_train)
tokenizer_test <- text_tokenizer(lower = FALSE) %>%
                    fit_text_tokenizer(news_new_test)

#put integers in a sequence
```

```
sequences_train <- texts_to_sequences(tokenizer_train, news_new_train)
sequences_test <- texts_to_sequences(tokenizer_test, news_new_test)

dtm_train <- create_dtm(it_train, vectorizer_train)
dtm_train <- as.matrix(dtm_train)
dtm_test <- create_dtm(it_test, vectorizer_test)
dtm_test <- as.matrix(dtm_test)
tcm_train <- create_tcm(it_train, vectorizer_train,
                        skip_grams_window_context = c("symmetric"))
xmax <- max(tcm_train)

######################## GloVe Embedding Dim 300 ###############
glove_train <- GlobalVectors$new(rank = 300, x_max = xmax)
# This gets the embeddings for target words.
wv_tar_train <- glove_train$fit_transform(tcm_train, n_iter = 100)
dim(wv_tar_train)
# This gets the embedding for the context words.
wv_con_train <- glove_train$components
dim(wv_con_train)

word_vect_train <- wv_tar_train + t(wv_con_train)
dim(word_vect_train)
word_vect_train <- as.matrix(word_vect_train)

### Getting embeddings per document
intersection <- colnames(dtm_train) %in% rownames(word_vect_train)
dtm_train <- dtm_train[,intersection]
words <- colnames(dtm_train)
dtm_emb_train <- matrix(0, nrow = nrow(dtm_train),
                  ncol = ncol(word_vect_train))
for (i in 1:nrow(dtm_train)){
  for (j in 1:ncol(dtm_train)){
    if (dtm_train[i,j] != 0){
      dtm_emb_train[i,] <- dtm_emb_train[i,] + dtm_train[i,j] *
      word_vect_train[which(rownames(word_vect_train) == words[j]),]
    }
  }
}

int_test <- colnames(dtm_test) %in% rownames(word_vect_train)
dtm_test <- dtm_test[,int_test]
words_test <- colnames(dtm_test)
dtm_emb_test <- matrix(0, nrow = nrow(dtm_test),
                  ncol = ncol(word_vect_train))
for (i in 1:nrow(dtm_test)){
  for (j in 1:ncol(dtm_test)){
    if (dtm_test[i,j] != 0){
      dtm_emb_test[i,] <- dtm_emb_test[i,] + dtm_test[i,j] *
      word_vect_train[which(rownames(word_vect_train) == words_test[j]),]
    }
  }
}

dtm_train_y <- cbind(dtm_emb_train, y_train)
dtm_train_y <- as.data.frame(dtm_train_y)
dtm_test_y <- cbind(dtm_emb_test, y_test)
dtm_test_y <- as.data.frame(dtm_test_y)

dtm_train_y$y_train <- as.factor(dtm_train_y$y_train)
dtm_test_y$y_test <- as.factor(dtm_test_y$y_test)

train.control <- trainControl(method = "cv", number = 10)
```

```
cl <- makePSOCKcluster(5)
registerDoParallel(cl)

##### CV 300 DIM#####

#Logistic regression directly on DTM
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "glm",
               family="binomial", control = list(maxit = 50))
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Linear Discriminant Analysis directly on DTM
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "lda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Quadratic Discriminant Analysis directly on DTM
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "qda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Mixture Discriminant Analysis directly on DTM
library(mda)
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "mda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Flexible Discriminant Analysis directly on DTM
library(earth)
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "fda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)
```

```
#Classification trees directly on DTM
library(rpart)
start_time <- Sys.time()
cpGrid <- expand.grid(.cp=seq(0.01, 0.5,0.01))
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "rpart",
               tuneGrid = cpGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

### Random Forest
library(randomForest)
start_time <- Sys.time()
mtry <- floor(sqrt(ncol(dtm_train_y)-1))
if (mtry > 10){
  a <- mtry - 10
} else {a <- mtry}
b <- mtry + 10
mtryGrid <- expand.grid(.mtry=seq(a, b, 1))
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "rf",
               tuneGrid = mtryGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
```

```
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)
```

PCA

```
library(tidyverse)
library(tidytext)
library(textdata)
library(text2vec)
library(keras)
library(uwot)
library(tensorflow)
library(nomclust)
library(ggplot2)
library(caret)
library(doParallel)

set.seed(121565)

rawdata_train <- read.csv(file = "/work/wtp/jhaus4/fnn_train.csv",
                          header = T,
                          stringsAsFactors = F)
rawdata_test <- read.csv(file = "/work/wtp/jhaus4/fnn_test.csv",
                         header = T,
                         stringsAsFactors = F)
rawdata_valid <- read.csv(file = "/work/wtp/jhaus4/fnn_dev.csv",
                          header = T,
                          stringsAsFactors = F)
rawdata <- rbind(rawdata_train, rawdata_valid, rawdata_test)

data <- rawdata %>%
  mutate(newsTextClean=gsub('[[:punct:]]+', '',
                     gsub('\\\\n|\\.|\\,|\\;',' ',
                     gsub("[^[:alnum:]]", " ",
                     (substr(statement,1,nchar(statement)-1))))))
#For all lowercase
#tolower(substr(statement,1,nchar(statement)-1))

news_tokens <- data %>% select(id, label_fnn, newsTextClean) %>%
  unnest_tokens(word, newsTextClean, to_lower = F) %>%
  #Use just "[a-z']+" for lower case letters
  #Change to "[[:alnum:]]+" for numbers and cases to be included
  #Change to "[A-z']+" for all cases
  mutate(word = str_extract(word, "[[:alnum:]]+"))
news_tokens <- news_tokens %>% drop_na(word)
#news_tokens <- news_tokens %>% anti_join(stop_words)

news_new <- news_tokens %>% group_by(word) %>%
  mutate(token_freq = n()) %>%
  filter(token_freq >= 60) %>%
  group_by(id, label_fnn) %>%
  summarise(newsTextClean = str_c(word, collapse = " "))

#Getting the labels
y <- news_new %>% select(label_fnn) %>% pull() %>% as.array()
y <- as.factor(y)
levels(y)
y <- ifelse(y == "fake", 0, 1)

it <- itoken(news_new$newsTextClean,
             tokenizer = word_tokenizer,
             ids = news_new$id,
             progressbar = T, tolower = F)
vocab <- create_vocabulary(it)
vocab <- prune_vocabulary(vocab)
vectorizer <- vocab_vectorizer(vocab)
news_new <- news_new %>% select(newsTextClean) %>% pull()

#Vectorize tokens - token receiving a unique integer
tokenizer <- text_tokenizer(lower = F) %>% fit_text_tokenizer(news_new)

#put integers in a sequence
sequences<- texts_to_sequences(tokenizer, news_new)

library(MASS)

#Document-term matrix all
un.words <- length(tokenizer$word_index)
docs <- length(sequences)
dtm <- matrix(0, nrow = docs, ncol = un.words)
for (i in 1:docs){
  word.vec <- sequences[[i]]
  c <- length(sequences[[i]])
  a <- 0
  for (j in 1:c){
    a <- a + 1
    b <- word.vec[a]
    dtm[i,b] <- dtm[i,b] + 1
  }
}

dtm_y <- cbind(y, dtm)
dtm_y <- as.data.frame(dtm_y)
```

```
words <- unlist(tokenizer$index_word, use.names = TRUE)
words <- c("PolitiFactRating", words)

colnames(dtm_y) <- words

trainIndex <- createDataPartition(dtm_y$PolitiFactRating,
              p = .8, list = F, times = 1)

dtm_train_y <- dtm_y[trainIndex,]
dtm_test_y <- dtm_y[-trainIndex,]

mean(dtm_train_y$PolitiFactRating)
mean(dtm_test_y$PolitiFactRating)

dtm_train_y$PolitiFactRating <- as.factor(dtm_train_y$PolitiFactRating)
dtm_test_y$PolitiFactRating <- as.factor(dtm_test_y$PolitiFactRating)

dtm_train <- dtm_train_y[,-1]
pca_train <- prcomp(dtm_train, center = T)
pct_train <- (pca_train$sdev^2/sum(pca_train$sdev^2))*100
sum(pct_train[1:400])
embeds <- pca_train$rotation[,1:400]
pc_train <- as.matrix(dtm_train) %*% as.matrix(embeds)

pc_train_y <- cbind(dtm_train_y[,1],pc_train)
pc_train_y <- as.data.frame(pc_train_y)

pc_train_y$V1 <- ifelse(pc_train_y$V1 == 1, 0, 1)
pc_train_y$V1 <- as.factor(pc_train_y$V1)
```

```
pc_test <- as.matrix(dtm_test_y[,-1]) %*% as.matrix(embeds)

pc_test_y <- cbind( dtm_test_y[,1], pc_test)
pc_test_y <- as.data.frame(pc_test_y)

pc_test_y$V1 <- ifelse(pc_test_y$V1 == 1, 0, 1)
pc_test_y$V1 <- as.factor(pc_test_y$V1)
identical(pc_test_y$V1, dtm_test_y[,1])

###################### PCA CV ################

train.control <- trainControl(method = "cv", number = 10)
cl <- makePSOCKcluster(5)
registerDoParallel(cl)

#Logistic regression directly on DTM
start_time <- Sys.time()
model <- train(V1 ~ ., data = pc_train_y,
               trControl = train.control, method = "glm",
               family="binomial", control = list(maxit = 50))
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(pc_test_y)
confusionMatrix(predictions, pc_test_y$V1)

#Linear Discriminant Analysis directly on DTM
start_time <- Sys.time()
model <- train(V1 ~ ., data = pc_train_y,
               trControl = train.control, method = "lda")
```

```
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(pc_test_y)
confusionMatrix(predictions, pc_test_y$V1)

#Quadratic Discriminant Analysis directly on DTM
start_time <- Sys.time()
model <- train(V1 ~ ., data = pc_train_y,
               trControl = train.control, method = "qda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(pc_test_y)
confusionMatrix(predictions, pc_test_y$V1)

#Mixture Discriminant Analysis directly on DTM
library(mda)
start_time <- Sys.time()
model <- train(V1 ~ ., data = pc_train_y,
               trControl = train.control, method = "mda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(pc_test_y)
confusionMatrix(predictions, pc_test_y$V1)

#Flexible Discriminant Analysis directly on DTM
```

```
library(earth)
start_time <- Sys.time()
model <- train(V1 ~ ., data = pc_train_y,
               trControl = train.control, method = "fda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(pc_test_y)
confusionMatrix(predictions, pc_test_y$V1)

#Classification trees directly on DTM
library(rpart)
start_time <- Sys.time()
cpGrid <- expand.grid(.cp=seq(0.01, 0.5,0.01))
model <- train(V1 ~ ., data = pc_train_y,
               trControl = train.control, method = "rpart",
               tuneGrid = cpGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(pc_test_y)
confusionMatrix(predictions, pc_test_y$V1)

### Random Forest
library(randomForest)
start_time <- Sys.time()
mtry <- floor(sqrt(ncol(pc_train_y)-1))
if (mtry > 10){
  a <- mtry - 10
} else {a <- mtry}
b <- mtry + 10
mtryGrid <- expand.grid(.mtry=seq(a, b, 1))
model <- train(V1 ~ ., data = pc_train_y,
               trControl = train.control, method = "rf",
               tuneGrid = mtryGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(pc_test_y)
confusionMatrix(predictions, pc_test_y$V1)
```

TF-IDF

```
library(tidyverse)
library(tidytext)
library(textdata)
library(text2vec)
library(keras)
library(uwot)
library(tensorflow)
library(nomclust)
library(ggplot2)
library(caret)
library(doParallel)

set.seed(121565)

############## COMBINING DATA SETS ############################
```

```
rawdata_train <- read.csv(file = "/work/wtp/jhaus4/fnn_train.csv",
                          header = T,
                          stringsAsFactors = F)
rawdata_test <- read.csv(file = "/work/wtp/jhaus4/fnn_test.csv",
                         header = T,
                         stringsAsFactors = F)
rawdata_valid <- read.csv(file = "/work/wtp/jhaus4/fnn_dev.csv",
                          header = T,
                          stringsAsFactors = F)
rawdata <- rbind(rawdata_train, rawdata_valid, rawdata_test)

data <- rawdata %>%
  mutate(newsTextClean=gsub('[[:punct:]]+', '',
                       gsub('\\\\n|\\.|\\,|\\;',' ',
                       gsub("[^[:alnum:]]", " ",
                       (substr(statement,1,nchar(statement)-1))))))
#For all lowercase
#tolower(substr(statement,1,nchar(statement)-1))

news_tokens <- data %>% select(id, label_fnn, newsTextClean) %>%
  unnest_tokens(word, newsTextClean, to_lower = FALSE) %>%
  #Use just "[a-z']+" for lower case letters
  #Change to "[[:alnum:]]+" for numbers and cases to be included
  #Change to "[A-z']+" for all cases
  mutate(word = str_extract(word, "[[:alnum:]]+"))
news_tokens <- news_tokens %>% drop_na(word)
#news_tokens <- news_tokens %>% anti_join(stop_words)

news_new <- news_tokens %>% group_by(word) %>%
  mutate(token_freq = n()) %>%
  filter(token_freq >=60) %>%
  group_by(id, label_fnn) %>%
  summarise(newsTextClean = str_c(word, collapse = " "))

#Getting the labels
y <- news_new %>% select(label_fnn) %>% pull() %>% as.array()
y <- as.factor(y)
levels(y)
y <- ifelse(y == "fake", 0, 1)

it <- itoken(news_new$newsTextClean,
             tokenizer = word_tokenizer,
             ids = news_new$id,
             progressbar = T, tolower = FALSE)
vocab <- create_vocabulary(it)
vocab <- prune_vocabulary(vocab)
vectorizer <- vocab_vectorizer(vocab)
news_new <- news_new %>% select(newsTextClean) %>% pull()

#Vectorize tokens - token receiving a unique integer
tokenizer <- text_tokenizer(lower = FALSE) %>%
                fit_text_tokenizer(news_new)

#put integers in a sequence
sequences<- texts_to_sequences(tokenizer, news_new)

library(MASS)

#Document-term matrix all
un.words <- length(tokenizer$word_index)
docs <- length(sequences)
dtm <- matrix(0, nrow = docs, ncol = un.words)
```

```
for (i in 1:docs){
  word.vec <- sequences[[i]]
  c <- length(sequences[[i]])
  a <- 0
  for (j in 1:c){
    a <- a + 1
    b <- word.vec[a]
    dtm[i,b] <- dtm[i,b] + 1
  }
}

dtm_y <- cbind(y, dtm)
dtm_y <- as.data.frame(dtm_y)

words <- unlist(tokenizer$index_word, use.names = TRUE)
words <- c("PolitiFactRating", words)

colnames(dtm_y) <- words

trainIndex <- createDataPartition(dtm_y$PolitiFactRating,
                p = .8, list = F, times = 1)

dtm_train_y <- dtm_y[trainIndex,]
dtm_test_y <- dtm_y[-trainIndex,]

mean(dtm_train_y$PolitiFactRating)
mean(dtm_test_y$PolitiFactRating)

dtm_train_y$PolitiFactRating <- as.factor(dtm_train_y$PolitiFactRating)
dtm_test_y$PolitiFactRating <- as.factor(dtm_test_y$PolitiFactRating)
```

```
dim(dtm_train_y)

#Term Frequency
tf_train <- t(apply(dtm_train_y[,-1],1,function(x) x/sum(x)))
tf_test <- t(apply(dtm_test_y[,-1],1,function(x) x/sum(x)))

#IDF
idf_train <- t(apply(dtm_train_y[,-1], 2, function(x)
  log(nrow(dtm_train_y)/sum(x != 0))))

#TF-IDF
tf.idf.train <- t(apply(tf_train, 1, function(x) x*idf_train))
tf.idf.test <- t(apply(tf_test, 1, function(x) x*idf_train))

dtm_train_y <- cbind(tf.idf.train, dtm_train_y[,1])
dtm_train_y <- as.data.frame(dtm_train_y)

dtm_test_y <- cbind(tf.idf.test, dtm_test_y[,1])
dtm_test_y <- as.data.frame(dtm_test_y)

words <- unlist(tokenizer$index_word, use.names = TRUE)
words <- c(words, "PolitiFactRating")

colnames(dtm_train_y) <- words
colnames(dtm_test_y) <- words

dtm_train_y$PolitiFactRating <- as.factor(dtm_train_y$PolitiFactRating)
dtm_test_y$PolitiFactRating <- as.factor(dtm_test_y$PolitiFactRating)
levels(dtm_train_y$PolitiFactRating)

dtm_train_y$PolitiFactRating <-
```

```
          ifelse(dtm_train_y$PolitiFactRating == 1, 0, 1)
dtm_train_y$PolitiFactRating <- as.factor(dtm_train_y$PolitiFactRating)

dtm_test_y$PolitiFactRating <-
          ifelse(dtm_test_y$PolitiFactRating == 1, 0, 1)
dtm_test_y$PolitiFactRating <- as.factor(dtm_test_y$PolitiFactRating)

levels(dtm_train_y$PolitiFactRating)

###################### TF-IDF CV ################

train.control <- trainControl(method = "cv", number = 10)
cl <- makePSOCKcluster(5)
registerDoParallel(cl)

#Logistic regression directly on DTM
start_time <- Sys.time()
model <- train(PolitiFactRating ~ ., data = dtm_train_y,
               trControl = train.control, method = "glm",
               family="binomial", control = list(maxit = 50))
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$PolitiFactRating)

#Linear Discriminant Analysis directly on DTM
start_time <- Sys.time()
model <- train(PolitiFactRating ~ ., data = dtm_train_y,
               trControl = train.control, method = "lda")
```

```
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$PolitiFactRating)

#Quadratic Discriminant Analysis directly on DTM
#start_time <- Sys.time()
#model <- train(PolitiFactRating ~ ., data = dtm_train_y,
#               trControl = train.control, method = "qda")
#end_time <- Sys.time()
#glm.time <- end_time - start_time
#glm.time
#model
#predictions <- model %>% predict(dtm_test_y)
#confusionMatrix(predictions,dtm_test_y$PolitiFactRating)


#Mixture Discriminant Analysis directly on DTM
library(mda)
#start_time <- Sys.time()
#model <- train(PolitiFactRating ~ ., data = dtm_train_y,
#               trControl = train.control, method = "mda")
#end_time <- Sys.time()
#glm.time <- end_time - start_time
#glm.time
#model
#predictions <- model %>% predict(dtm_test_y)
#confusionMatrix(predictions,dtm_test_y$PolitiFactRating)
```

```
#Flexible Discriminant Analysis directly on DTM
library(earth)
#start_time <- Sys.time()
#model <- train(PolitiFactRating ~ ., data = dtm_train_y,
#               trControl = train.control, method = "fda")
#end_time <- Sys.time()
#glm.time <- end_time - start_time
#glm.time
#model
#predictions <- model %>% predict(dtm_test_y)
#confusionMatrix(predictions,dtm_test_y$PolitiFactRating)

#Classification trees directly on DTM
library(rpart)
start_time <- Sys.time()
cpGrid <- expand.grid(.cp=seq(0.01, 0.5,0.01))
model <- train(PolitiFactRating ~ ., data = dtm_train_y,
               trControl = train.control, method = "rpart",
               tuneGrid = cpGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$PolitiFactRating)


### Random Forest
library(randomForest)
start_time <- Sys.time()
```

```
mtry <- floor(sqrt(ncol(dtm_train_y)-1))
if (mtry > 10){
  a <- mtry - 10
} else {a <- mtry}
b <- mtry + 10
mtryGrid <- expand.grid(.mtry=seq(a, b, 1))
model <- train(PolitiFactRating ~ ., data = dtm_train_y,
               trControl = train.control, method = "rf",
               tuneGrid = mtryGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$PolitiFactRating)
```

Word2Vec Continuous Bag of Words

```
library(tidyverse)
library(tidytext)
library(textdata)
library(text2vec)
library(keras)
library(uwot)
library(tensorflow)
library(nomclust)
library(ggplot2)
library(word2vec)
library(caret)
library(doParallel)
```

```
set.seed(121565)

rawdata_train <- read.csv(file = "/work/wtp/jhaus4/fnn_train.csv",
                          header = T,
                          stringsAsFactors = F)
rawdata_test <- read.csv(file = "/work/wtp/jhaus4/fnn_test.csv",
                         header = T,
                         stringsAsFactors = F)
rawdata_valid <- read.csv(file = "/work/wtp/jhaus4/fnn_dev.csv",
                          header = T,
                          stringsAsFactors = F)
rawdata <- rbind(rawdata_train, rawdata_valid, rawdata_test)

data <- rawdata %>%
  mutate(newsTextClean=gsub('[[:punct:]]+', '',
                       gsub('\\\\n|\\.|\\,|\\;',' ',
                       gsub("[^[:alnum:]]", " ",
                       (substr(statement,1,nchar(statement)-1))))))

news_tokens <- data %>% select(id, label_fnn, newsTextClean) %>%
  unnest_tokens(word, newsTextClean, to_lower = FALSE) %>%
  #Use just "[a-z']+" for lower case letters
  #Change to "[[:alnum:]]+" for numbers to be included
  #"[A-z']+"
  mutate(word = str_extract(word, "[[:alnum:]]+"))
news_tokens <- news_tokens %>% drop_na(word)
#news_tokens <- news_tokens %>% anti_join(stop_words)

news_new <- news_tokens %>% group_by(word) %>%
  mutate(token_freq = n()) %>%
  filter(token_freq >= 60) %>%
  group_by(id, label_fnn) %>%
  summarise(newsTextClean = str_c(word, collapse = " "))

#Use just "[a-z']+" for lower case letters
#Change to "[[:alnum:]]+" for numbers to be included
max.length <- max(str_count(news_new$newsTextClean, "[[:alnum:]]+"))

trainIndex <- createDataPartition(news_new$label_fnn,
                p = .8, list = F, times = 1)
train.control <- trainControl(method = "cv", number = 10)

news_new_train <- news_new[trainIndex,]
news_new_test <- news_new[-trainIndex,]

#Getting the labels
y_train <- news_new_train %>% select(label_fnn) %>% pull() %>% as.array()
y_test <- news_new_test %>% select(label_fnn) %>% pull() %>% as.array()
y_train <- as.factor(y_train)
y_test <- as.factor(y_test)
levels(y_train)
levels(y_test)
y_train <- ifelse(y_train == "fake", 0, 1)
y_test <- ifelse(y_test == "fake", 0, 1)

it_train <- itoken(news_new_train$newsTextClean,
             tokenizer = word_tokenizer,
             ids = news_new$id,
             progressbar = T, tolower = FALSE)
it_test <- itoken(news_new_test$newsTextClean,
                   tokenizer = word_tokenizer,
                   ids = news_new$id,
                   progressbar = T, tolower = FALSE)
vocab_train <- create_vocabulary(it_train)
vocab_test <- create_vocabulary(it_test)
vocab_train <- prune_vocabulary(vocab_train)
vocab_test <- prune_vocabulary(vocab_test)
vectorizer_train <- vocab_vectorizer(vocab_train)
vectorizer_test <- vocab_vectorizer(vocab_test)
news_new_train <- news_new_train %>% select(newsTextClean) %>% pull()
news_new_test <- news_new_test %>% select(newsTextClean) %>% pull()

#Vectorize tokens - token receiving a unique integer
tokenizer_train <- text_tokenizer(lower = FALSE) %>%
                    fit_text_tokenizer(news_new_train)
tokenizer_test <- text_tokenizer(lower = FALSE) %>%
                    fit_text_tokenizer(news_new_test)

#put integers in a sequence
sequences_train <- texts_to_sequences(tokenizer_train, news_new_train)
sequences_test <- texts_to_sequences(tokenizer_test, news_new_test)

dtm_train <- create_dtm(it_train, vectorizer_train)
dtm_train <- as.matrix(dtm_train)
dtm_test <- create_dtm(it_test, vectorizer_test)
dtm_test <- as.matrix(dtm_test)

############WORD2VEC WITH 300 DIM##################
model_train <- word2vec(x = news_new_train, dim = 300, iter = 100,
                min_count = 0L)
model_train$success
emb_train <- as.matrix(model_train)
emb_train[1:5, 1:5]
```

```
emb_train_ord <- emb_train[order(rownames(emb_train)),]
emb_train_ord[1:5, 1:5]

### Getting embeddings per document
intersection <- colnames(dtm_train) %in% rownames(emb_train_ord)
dtm_train <- dtm_train[,intersection]
words <- colnames(dtm_train)
dtm_emb_train <- matrix(0, nrow = nrow(dtm_train),
                  ncol = ncol(emb_train_ord))
for (i in 1:nrow(dtm_train)){
 for (j in 1:ncol(dtm_train)){
   if (dtm_train[i,j] != 0){
     dtm_emb_train[i,] <- dtm_emb_train[i,] + dtm_train[i,j] *
       emb_train_ord[which(rownames(emb_train_ord) == words[j]),]
   }
 }
}
dtm_emb_train[1:5, 1:5]
max(dtm_emb_train)

int_test <- colnames(dtm_test) %in% rownames(emb_train_ord)
dtm_test <- dtm_test[,int_test]
words_test <- colnames(dtm_test)
dtm_emb_test <- matrix(0, nrow = nrow(dtm_test),
                ncol = ncol(emb_train_ord))
for (i in 1:nrow(dtm_test)){
 for (j in 1:ncol(dtm_test)){
   if (dtm_test[i,j] != 0){
     dtm_emb_test[i,] <- dtm_emb_test[i,] + dtm_test[i,j] *
       emb_train_ord[which(rownames(emb_train_ord) == words_test[j]),]
   }
 }
}
dtm_emb_test[1:5, 1:5]
max(dtm_emb_test)

dtm_train_y <- cbind(dtm_emb_train, y_train)
dtm_train_y <- as.data.frame(dtm_train_y)
dtm_test_y <- cbind(dtm_emb_test, y_test)
dtm_test_y <- as.data.frame(dtm_test_y)

dtm_train_y$y_train <- as.factor(dtm_train_y$y_train)
dtm_test_y$y_test <- as.factor(dtm_test_y$y_test)

train.control <- trainControl(method = "cv", number = 10)
cl <- makePSOCKcluster(5)
registerDoParallel(cl)

 ##### CV 300 DIM#####

# #Logistic regression directly on DTM
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "glm",
              family="binomial", control = list(maxit = 50))
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)
```

```
#Linear Discriminant Analysis directly on DTM
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "lda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Quadratic Discriminant Analysis directly on DTM
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "qda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

# #Mixture Discriminant Analysis directly on DTM
library(mda)
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "mda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
```

```
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

# #Flexible Discriminant Analysis directly on DTM
library(earth)
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "fda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

# #Classification trees directly on DTM
library(rpart)
start_time <- Sys.time()
cpGrid <- expand.grid(.cp=seq(0.01, 0.5,0.01))
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "rpart",
              tuneGrid = cpGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

# ### Random Forest
library(randomForest)
```

```
start_time <- Sys.time()
mtry <- floor(sqrt(ncol(dtm_train_y)-1))
if (mtry > 10){
 a <- mtry - 10
} else {a <- mtry}
b <- mtry + 10
mtryGrid <- expand.grid(.mtry=seq(a, b, 1))
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "rf",
              tuneGrid = mtryGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)
```

Word2Vec Skip-Gram

```
library(tidyverse)
library(tidytext)
library(textdata)
library(text2vec)
library(keras)
library(uwot)
library(tensorflow)
library(nomclust)
library(ggplot2)
library(word2vec)
library(caret)
library(doParallel)
```

```
set.seed(121565)

rawdata_train <- read.csv(file = "/work/wtp/jhaus4/fnn_train.csv",
                          header = T,
                          stringsAsFactors = F)
rawdata_test <- read.csv(file = "/work/wtp/jhaus4/fnn_test.csv",
                          header = T,
                         stringsAsFactors = F)
rawdata_valid <- read.csv(file = "/work/wtp/jhaus4/fnn_dev.csv",
                          header = T,
                          stringsAsFactors = F)
rawdata <- rbind(rawdata_train, rawdata_valid, rawdata_test)

data <- rawdata %>%
  mutate(newsTextClean=gsub('[[:punct:]]+', '',
                       gsub('\\\\n|\\.|\\,|\\;',' ',
                       gsub("[^[:alnum:]]", " ",
                       (substr(statement,1,nchar(statement)-1))))))

news_tokens <- data %>% select(id, label_fnn, newsTextClean) %>%
  unnest_tokens(word, newsTextClean, to_lower = FALSE) %>%
  #Use just "[a-z']+" for lower case letters
  #Change to "[[:alnum:]]+" for numbers to be included
  #"[A-z']+"
  mutate(word = str_extract(word, "[[:alnum:]]+"))
news_tokens <- news_tokens %>% drop_na(word)
#news_tokens <- news_tokens %>% anti_join(stop_words)

news_new <- news_tokens %>% group_by(word) %>%
  mutate(token_freq = n()) %>%
  filter(token_freq >= 60) %>%
  group_by(id, label_fnn) %>%
  summarise(newsTextClean = str_c(word, collapse = " "))

#Use just "[a-z']+" for lower case letters
#Change to "[[:alnum:]]+" for numbers to be included
max.length <- max(str_count(news_new$newsTextClean, "[[:alnum:]]+"))

trainIndex <- createDataPartition(news_new$label_fnn,
                p = .8, list = F, times = 1)
train.control <- trainControl(method = "cv", number = 10)
cl <- makePSOCKcluster(5)
registerDoParallel(cl)

news_new_train <- news_new[trainIndex,]
news_new_test <- news_new[-trainIndex,]

#Getting the labels
y_train <- news_new_train %>% select(label_fnn) %>% pull() %>% as.array()
y_test <- news_new_test %>% select(label_fnn) %>% pull() %>% as.array()
y_train <- as.factor(y_train)
y_test <- as.factor(y_test)
levels(y_train)
levels(y_test)
y_train <- ifelse(y_train == "fake", 0, 1)
y_test <- ifelse(y_test == "fake", 0, 1)

it_train <- itoken(news_new_train$newsTextClean,
                   tokenizer = word_tokenizer,
                   ids = news_new$id,
                   progressbar = T, tolower = FALSE)
```

```
it_test <- itoken(news_new_test$newsTextClean,
                  tokenizer = word_tokenizer,
                  ids = news_new$id,
                  progressbar = T, tolower = FALSE)
vocab_train <- create_vocabulary(it_train)
vocab_test <- create_vocabulary(it_test)
vocab_train <- prune_vocabulary(vocab_train)
vocab_test <- prune_vocabulary(vocab_test)
vectorizer_train <- vocab_vectorizer(vocab_train)
vectorizer_test <- vocab_vectorizer(vocab_test)
news_new_train <- news_new_train %>% select(newsTextClean) %>% pull()
news_new_test <- news_new_test %>% select(newsTextClean) %>% pull()

#Vectorize tokens - token receiving a unique integer
tokenizer_train <- text_tokenizer(lower = FALSE) %>%
                    fit_text_tokenizer(news_new_train)
tokenizer_test <- text_tokenizer(lower = FALSE) %>%
                    fit_text_tokenizer(news_new_test)

#put integers in a sequence
sequences_train <- texts_to_sequences(tokenizer_train, news_new_train)
sequences_test <- texts_to_sequences(tokenizer_test, news_new_test)

dtm_train <- create_dtm(it_train, vectorizer_train)
dtm_train <- as.matrix(dtm_train)
dtm_test <- create_dtm(it_test, vectorizer_test)
dtm_test <- as.matrix(dtm_test)

############WORD2VEC SG WITH 300 DIM#################
model_train <- word2vec(x = news_new_train, type = "skip-gram",
                        dim = 300, iter = 100, min_count = 0L)
```

```
model_train$success
emb_train <- as.matrix(model_train)
emb_train_ord <- emb_train[order(rownames(emb_train)),]

### Getting embeddings per document
intersection <- colnames(dtm_train) %in% rownames(emb_train_ord)
dtm_train <- dtm_train[,intersection]
words <- colnames(dtm_train)
dtm_emb_train <- matrix(0, nrow = nrow(dtm_train),
                  ncol = ncol(emb_train_ord))
for (i in 1:nrow(dtm_train)){
  for (j in 1:ncol(dtm_train)){
    if (dtm_train[i,j] != 0){
      dtm_emb_train[i,] <- dtm_emb_train[i,] + dtm_train[i,j] *
        emb_train_ord[which(rownames(emb_train_ord) == words[j]),]
    }
  }
}

int_test <- colnames(dtm_test) %in% rownames(emb_train_ord)
dtm_test <- dtm_test[,int_test]
words_test <- colnames(dtm_test)
dtm_emb_test <- matrix(0, nrow = nrow(dtm_test),
                  ncol = ncol(emb_train_ord))
for (i in 1:nrow(dtm_test)){
  for (j in 1:ncol(dtm_test)){
    if (dtm_test[i,j] != 0){
      dtm_emb_test[i,] <- dtm_emb_test[i,] + dtm_test[i,j] *
        emb_train_ord[which(rownames(emb_train_ord) == words_test[j]),]
    }
  }
}

dtm_train_y <- cbind(dtm_emb_train, y_train)
dtm_train_y <- as.data.frame(dtm_train_y)
dtm_test_y <- cbind(dtm_emb_test, y_test)
dtm_test_y <- as.data.frame(dtm_test_y)

dtm_train_y$y_train <- as.factor(dtm_train_y$y_train)
dtm_test_y$y_test <- as.factor(dtm_test_y$y_test)

##### CV 300 DIM#####

#Logistic regression directly on DTM
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "glm",
               family="binomial", control = list(maxit = 50))
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Linear Discriminant Analysis directly on DTM
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "lda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
```

```
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Quadratic Discriminant Analysis directly on DTM
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "qda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Mixture Discriminant Analysis directly on DTM
library(mda)
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "mda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Flexible Discriminant Analysis directly on DTM
library(earth)
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "fda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Classification trees directly on DTM
library(rpart)
start_time <- Sys.time()
cpGrid <- expand.grid(.cp=seq(0.01, 0.5,0.01))
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "rpart",
               tuneGrid = cpGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

### Random Forest
library(randomForest)
start_time <- Sys.time()
mtry <- floor(sqrt(ncol(dtm_train_y)-1))
if (mtry > 10){
  a <- mtry - 10
} else {a <- mtry}
b <- mtry + 10
mtryGrid <- expand.grid(.mtry=seq(a, b, 1))
```

```
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "rf",
               tuneGrid = mtryGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)
```

Multi-Class Classification

BERT

```
library(tidyverse)
library(tidytext)
library(textdata)
library(text2vec)
library(keras)
library(uwot)
library(tensorflow)
library(nomclust)
library(ggplot2)
library(caret)
library(doParallel)
library(nnet)

set.seed(121565)

fnn_train <- read.csv(file = "/work/wtp/jhaus4/fnn_train.csv",
                      header = T,
                      stringsAsFactors = F)
```

```
fnn_test <- read.csv(file = "/work/wtp/jhaus4/fnn_test.csv",
                      header = T,
                     stringsAsFactors = F)
fnn_valid <- read.csv(file = "/work/wtp/jhaus4/fnn_dev.csv",
                      header = T,
                      stringsAsFactors = F)
fnn <- rbind(fnn_train, fnn_valid, fnn_test)

LIAR_train <- read.csv(file = "/work/wtp/jhaus4/liar_train.csv",
                        header = T,
                       stringsAsFactors = F)
LIAR_test <- read.csv(file = "/work/wtp/jhaus4/liar_test.csv",
                      header = T,
                      stringsAsFactors = F)
LIAR_valid <- read.csv(file = "/work/wtp/jhaus4/liar_dev.csv",
                        header = T,
                       stringsAsFactors = F)
LIAR <- rbind(LIAR_train, LIAR_valid, LIAR_test)

common <- intersect(fnn$id, LIAR$id)
rawdata <- LIAR[LIAR$id %in% common, ]

data <- rawdata %>%
  mutate(newsTextClean=gsub('[[:punct:]]+', '',
                       gsub('\\\\n|\\.|\\,|\\;',' ',
                       gsub("[^[:alnum:]]", " ",
                       tolower(substr(statement,1,nchar(statement)-1))))))

data <- data %>% separate(date,
                  c("Year", "Month", "Day","Junk"), sep = "-")
#For all lowercase
```

```
#tolower(substr(statement,1,nchar(statement)-1))

news_tokens <- data %>% select(id, label.liar, Year, newsTextClean) %>%
  unnest_tokens(word, newsTextClean, to_lower = FALSE) %>%
  #Use just "[a-z']+" for lower case letters
  mutate(word = str_extract(word, "[a-z']+"))
news_tokens <- news_tokens %>% drop_na(word)
#news_tokens <- news_tokens %>% anti_join(stop_words)

news_new <- news_tokens %>% group_by(word) %>%
  mutate(token_freq = n()) %>%
  filter(token_freq >= 60) %>%
  group_by(id, label.liar, Year) %>%
  summarise(newsTextClean = str_c(word, collapse = " "))

trainIndex <- createDataPartition(news_new$label.liar,
                                  p = .8, list = F, times = 1)
train.control <- trainControl(method = "cv", number = 10)
cl <- makePSOCKcluster(5)
registerDoParallel(cl)

news_new_train <- news_new[trainIndex,]
news_new_test <- news_new[-trainIndex,]

#write.csv(news_new_train, "liartrain60Stopwords.csv", row.names = FALSE)
#write.csv(news_new_test, "liartest60Stopwords.csv", row.names = FALSE)

bert_train <- read.csv(file = "BERT_LIAR_Train_60_Stopwords.csv",
                       header = F)
dim(bert_train)
bert_train <- bert_train[-1,]
```

```
dim(bert_train)


bert_test <- read.csv(file = "BERT_LIAR_Test_60_Stopwords.csv",
                      header = F)
dim(bert_test)
bert_test <- bert_test[-1,]
dim(bert_test)


#Getting the labels
y_train <- news_new_train %>% select(label.liar) %>%
  pull() %>% as.array()
y_test <- news_new_test %>% select(label.liar) %>%
  pull() %>% as.array()
y_train <- as.factor(y_train)
y_test <- as.factor(y_test)
levels(y_train)
levels(y_test)
levels(y_train) <- c("true", "mostly-true", "half-true",
                     "barely-true", "false", "pants-fire")
levels(y_test) <- c("true", "mostly-true", "half-true",
                    "barely-true", "false", "pants-fire")


#year_train <- news_new_train %>% select(Year) %>% pull() %>% as.array()
#year_test <- news_new_test %>% select(Year) %>% pull() %>% as.array()
#year_train <- as.factor(year_train)
#year_test <- as.factor(year_test)
#levels(year_train)
#levels(year_test)


dtm_train_y <- cbind(bert_train, y_train)
dtm_train_y <- as.data.frame(dtm_train_y)
```

```
dtm_test_y <- cbind(bert_test, y_test)
dtm_test_y <- as.data.frame(dtm_test_y)
#dtm_test_y <- dtm_test_y %>% rename("year_train"="year_test")

dtm_train_y$y_train <- as.factor(dtm_train_y$y_train)
dtm_test_y$y_test <- as.factor(dtm_test_y$y_test)
#dtm_train_y$year_train <- as.factor(dtm_train_y$year_train)
#dtm_test_y$year_train <- as.factor(dtm_test_y$year_train)

##### CV DIM#####

#Ordinal Logistic regression
library(MASS)
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "polr",
               control = list(maxit = 100))
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Linear Discriminant Analysis directly on DTM
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "lda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
```

```
model

predictions <- model %>% predict(dtm_test_y)

confusionMatrix(predictions,dtm_test_y$y_test)


#Quadratic Discriminant Analysis directly on DTM

start_time <- Sys.time()

model <- train(y_train ~ ., data = dtm_train_y,

               trControl = train.control, method = "qda")

end_time <- Sys.time()

glm.time <- end_time - start_time

glm.time

model

predictions <- model %>% predict(dtm_test_y)

confusionMatrix(predictions,dtm_test_y$y_test)


#Mixture Discriminant Analysis directly on DTM

library(mda)

start_time <- Sys.time()

model <- train(y_train ~ ., data = dtm_train_y,

               trControl = train.control, method = "mda")

end_time <- Sys.time()

glm.time <- end_time - start_time

glm.time

model

predictions <- model %>% predict(dtm_test_y)

confusionMatrix(predictions,dtm_test_y$y_test)


#Flexible Discriminant Analysis directly on DTM

library(earth)

start_time <- Sys.time()

model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "fda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Classification trees directly on DTM
library(rpart)
start_time <- Sys.time()
cpGrid <- expand.grid(.cp=seq(0.01, 0.5,0.01))
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "rpart",
               tuneGrid = cpGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

### Random Forest
library(randomForest)
start_time <- Sys.time()
mtry <- floor(sqrt(ncol(dtm_train_y)-1))
if (mtry > 10){
  a <- mtry - 10
} else {a <- mtry}
b <- mtry + 10
mtryGrid <- expand.grid(.mtry=seq(a, b, 1))
```

```
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "rf",
               tuneGrid = mtryGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)
```

GloVe

```
library(tidyverse)
library(tidytext)
library(textdata)
library(text2vec)
library(keras)
library(uwot)
library(tensorflow)
library(nomclust)
library(ggplot2)
library(word2vec)
library(caret)
library(doParallel)
library(nnet)

set.seed(121565)

############## COMBINING DATA SETS ############################

fnn_train <- read.csv(file = "/work/wtp/jhaus4/fnn_train.csv",
                      header = T,
                      stringsAsFactors = F)
fnn_test <- read.csv(file = "/work/wtp/jhaus4/fnn_test.csv",
                      header = T,
                     stringsAsFactors = F)
fnn_valid <- read.csv(file = "/work/wtp/jhaus4/fnn_dev.csv",
                      header = T,
                      stringsAsFactors = F)
fnn <- rbind(fnn_train, fnn_valid, fnn_test)

LIAR_train <- read.csv(file = "/work/wtp/jhaus4/liar_train.csv",
                        header = T,
                       stringsAsFactors = F)
LIAR_test <- read.csv(file = "/work/wtp/jhaus4/liar_test.csv",
                      header = T,
                      stringsAsFactors = F)
LIAR_valid <- read.csv(file = "/work/wtp/jhaus4/liar_dev.csv",
                        header = T,
                       stringsAsFactors = F)
LIAR <- rbind(LIAR_train, LIAR_valid, LIAR_test)

common <- intersect(fnn$id, LIAR$id)
rawdata <- LIAR[LIAR$id %in% common, ]



data <- rawdata %>%
  mutate(newsTextClean=gsub('[[:punct:]]+', '',
                       gsub('\\\\n|\\.|\\,|\\;',' ',
                       gsub("[^[:alnum:]]", " ",
                       tolower(substr(statement,1,nchar(statement)-1))))))
```

```
data <- data %>% separate(date,
                 c("Year", "Month", "Day","Junk"), sep = "-")

news_tokens <- data %>% select(id, label.liar, Year, newsTextClean) %>%
  unnest_tokens(word, newsTextClean, to_lower = FALSE) %>%
  #Use just "[a-z']+" for lower case letters
  mutate(word = str_extract(word, "[a-z']+"))
news_tokens <- news_tokens %>% drop_na(word)
#news_tokens <- news_tokens %>% anti_join(stop_words)

news_new <- news_tokens %>% group_by(word) %>%
  mutate(token_freq = n()) %>%
  filter(token_freq >= 30) %>%
  group_by(id, label.liar, Year) %>%
  summarise(newsTextClean = str_c(word, collapse = " "))

max.length <- max(str_count(news_new$newsTextClean, "[a-z']+"))

trainIndex <- createDataPartition(news_new$label.liar,
                p = .8, list = F, times = 1)
train.control <- trainControl(method = "cv", number = 10)
cl <- makePSOCKcluster(5)
registerDoParallel(cl)

news_new_train <- news_new[trainIndex,]
news_new_test <- news_new[-trainIndex,]

#Getting the labels
y_train <- news_new_train %>% select(label.liar) %>% pull() %>% as.array()
y_test <- news_new_test %>% select(label.liar) %>% pull() %>% as.array()
y_train <- as.factor(y_train)
```

```
y_test <- as.factor(y_test)
levels(y_train)
levels(y_test)
#y_train <- relevel(y_train, ref = "true")
#y_test <- relevel(y_test, ref = "true")
levels(y_train) <- c("true", "mostly-true", "half-true",
          "barely-true", "false", "pants-fire")
levels(y_test) <- c("true", "mostly-true", "half-true",
          "barely-true", "false", "pants-fire")

#year_train <- news_new_train %>% select(Year) %>% pull() %>% as.array()
#year_test <- news_new_test %>% select(Year) %>% pull() %>% as.array()
#year_train <- as.factor(year_train)
#year_test <- as.factor(year_test)
#levels(year_train)
#levels(year_test)

it_train <- itoken(news_new_train$newsTextClean,
                   tokenizer = word_tokenizer,
                   ids = news_new$id,
                   progressbar = T, tolower = FALSE)
it_test <- itoken(news_new_test$newsTextClean,
                  tokenizer = word_tokenizer,
                  ids = news_new$id,
                  progressbar = T, tolower = FALSE)
vocab_train <- create_vocabulary(it_train)
vocab_test <- create_vocabulary(it_test)
vocab_train <- prune_vocabulary(vocab_train)
vocab_test <- prune_vocabulary(vocab_test)
vectorizer_train <- vocab_vectorizer(vocab_train)
vectorizer_test <- vocab_vectorizer(vocab_test)
```

```
news_new_train <- news_new_train %>% select(newsTextClean) %>% pull()
news_new_test <- news_new_test %>% select(newsTextClean) %>% pull()

#Vectorize tokens - token receiving a unique integer
tokenizer_train <- text_tokenizer(lower = FALSE) %>%
      fit_text_tokenizer(news_new_train)
tokenizer_test <- text_tokenizer(lower = FALSE) %>%
      fit_text_tokenizer(news_new_test)

#put integers in a sequence
sequences_train <- texts_to_sequences(tokenizer_train, news_new_train)
sequences_test <- texts_to_sequences(tokenizer_test, news_new_test)

dtm_train <- create_dtm(it_train, vectorizer_train)
dtm_train <- as.matrix(dtm_train)
dtm_test <- create_dtm(it_test, vectorizer_test)
dtm_test <- as.matrix(dtm_test)
tcm_train <- create_tcm(it_train, vectorizer_train,
                        skip_grams_window_context = c("symmetric"))
xmax <- max(tcm_train)

####################### GloVe Embedding Dim 300 ##############
glove_train <- GlobalVectors$new(rank = 300, x_max = xmax)
# This gets the embeddings for target words.
wv_tar_train <- glove_train$fit_transform(tcm_train, n_iter = 100)
dim(wv_tar_train)
# This gets the embedding for the context words.
wv_con_train <- glove_train$components
dim(wv_con_train)

word_vect_train <- wv_tar_train + t(wv_con_train)
```

```
dim(word_vect_train)
word_vect_train <- as.matrix(word_vect_train)

### Getting embeddings per document
intersection <- colnames(dtm_train) %in% rownames(word_vect_train)
dtm_train <- dtm_train[,intersection]
words <- colnames(dtm_train)
dtm_emb_train <- matrix(0, nrow = nrow(dtm_train),
      ncol = ncol(word_vect_train))
for (i in 1:nrow(dtm_train)){
  for (j in 1:ncol(dtm_train)){
    if (dtm_train[i,j] != 0){
      dtm_emb_train[i,] <- dtm_emb_train[i,] + dtm_train[i,j] *
        word_vect_train[which(rownames(word_vect_train) == words[j]),]
    }
  }
}

int_test <- colnames(dtm_test) %in% rownames(word_vect_train)
dtm_test <- dtm_test[,int_test]
words_test <- colnames(dtm_test)
dtm_emb_test <- matrix(0, nrow = nrow(dtm_test),
      ncol = ncol(word_vect_train))
for (i in 1:nrow(dtm_test)){
  for (j in 1:ncol(dtm_test)){
    if (dtm_test[i,j] != 0){
      dtm_emb_test[i,] <- dtm_emb_test[i,] + dtm_test[i,j] *
        word_vect_train[which(rownames(word_vect_train) == words_test[j]),]
    }
  }
}
```

```
dtm_train_y <- cbind(dtm_emb_train, y_train)

dtm_train_y <- as.data.frame(dtm_train_y)

dtm_test_y <- cbind(dtm_emb_test, y_test)

dtm_test_y <- as.data.frame(dtm_test_y)

#dtm_test_y <- dtm_test_y %>% rename("year_train"="year_test")


dtm_train_y$y_train <- as.factor(dtm_train_y$y_train)

dtm_test_y$y_test <- as.factor(dtm_test_y$y_test)

#dtm_train_y$year_train <- as.factor(dtm_train_y$year_train)

#dtm_test_y$year_test <- as.factor(dtm_test_y$year_test)


##### CV 300 DIM#####


#Ordinal Logistic regression

library(MASS)

start_time <- Sys.time()

model <- train(y_train ~ ., data = dtm_train_y,

               trControl = train.control, method = "polr",

               control = list(maxit = 100))

end_time <- Sys.time()

glm.time <- end_time - start_time

glm.time

model

predictions <- model %>% predict(dtm_test_y)

confusionMatrix(predictions,dtm_test_y$y_test)


#Linear Discriminant Analysis directly on DTM

start_time <- Sys.time()

model <- train(y_train ~ ., data = dtm_train_y,

               trControl = train.control, method = "lda")
```

```
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Quadratic Discriminant Analysis directly on DTM
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "qda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Mixture Discriminant Analysis directly on DTM
library(mda)
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "mda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Flexible Discriminant Analysis directly on DTM
```

```
library(earth)
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "fda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Classification trees directly on DTM
library(rpart)
start_time <- Sys.time()
cpGrid <- expand.grid(.cp=seq(0.01, 0.5,0.01))
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "rpart",
               tuneGrid = cpGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

### Random Forest
library(randomForest)
start_time <- Sys.time()
mtry <- floor(sqrt(ncol(dtm_train_y)-1))
if (mtry > 10){
  a <- mtry - 10
} else {a <- mtry}
b <- mtry + 10
mtryGrid <- expand.grid(.mtry=seq(a, b, 1))
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "rf",
               tuneGrid = mtryGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)
```

PCA

```
library(tidyverse)
library(tidytext)
library(textdata)
library(text2vec)
library(keras)
library(uwot)
library(tensorflow)
library(nomclust)
library(ggplot2)
library(caret)
library(doParallel)
library(nnet)

set.seed(121565)

fnn_train <- read.csv(file = "/work/wtp/jhaus4/fnn_train.csv",
                      header = T,
                      stringsAsFactors = F)
fnn_test <- read.csv(file = "/work/wtp/jhaus4/fnn_test.csv",
                      header = T,
                     stringsAsFactors = F)
fnn_valid <- read.csv(file = "/work/wtp/jhaus4/fnn_dev.csv",
                      header = T,
                      stringsAsFactors = F)
fnn <- rbind(fnn_train, fnn_valid, fnn_test)

LIAR_train <- read.csv(file = "/work/wtp/jhaus4/liar_train.csv",
                      header = T,
                       stringsAsFactors = F)
LIAR_test <- read.csv(file = "/work/wtp/jhaus4/liar_test.csv",
                      header = T,
                      stringsAsFactors = F)
LIAR_valid <- read.csv(file = "/work/wtp/jhaus4/liar_dev.csv",
                      header = T,
                       stringsAsFactors = F)
LIAR <- rbind(LIAR_train, LIAR_valid, LIAR_test)

common <- intersect(fnn$id, LIAR$id)
rawdata <- LIAR[LIAR$id %in% common, ]



data <- rawdata %>%
  mutate(newsTextClean=gsub('[[:punct:]]+', '',
                       gsub('\\\\n|\\.|\\,|\\;',' ',
                       gsub("[^[:alnum:]]", " ",
                       tolower(substr(statement,1,nchar(statement)-1))))))
```

```
data <- data %>% separate(date,
          c("Year", "Month", "Day","Junk"), sep = "-")

news_tokens <- data %>% select(id, label.liar, Year, newsTextClean) %>%
  unnest_tokens(word, newsTextClean, to_lower = FALSE) %>%
  #Use just "[a-z']+" for lower case letters
  mutate(word = str_extract(word, "[a-z']+"))
news_tokens <- news_tokens %>% drop_na(word)
#news_tokens <- news_tokens %>% anti_join(stop_words)

news_new <- news_tokens %>% group_by(word) %>%
  mutate(token_freq = n()) %>%
  #filter(token_freq >= 60) %>%
  group_by(id, label.liar, Year) %>%
  summarise(newsTextClean = str_c(word, collapse = " "))

#Getting the labels
y <- news_new %>% select(label.liar) %>% pull() %>% as.array()
y <- as.factor(y)
levels(y)
levels(y) <- c("true", "mostly-true", "half-true",
      "barely-true", "false", "pants-fire")

#year <- news_new %>% select(Year) %>% pull() %>% as.array()
#year <- as.factor(year)
#levels(year)

it <- itoken(news_new$newsTextClean,
             tokenizer = word_tokenizer,
             ids = news_new$id,
             progressbar = T, tolower = F)
```

```
vocab <- create_vocabulary(it)

vocab <- prune_vocabulary(vocab)

vectorizer <- vocab_vectorizer(vocab)

news_new <- news_new %>% select(newsTextClean) %>% pull()


#Vectorize tokens - token receiving a unique integer

tokenizer <- text_tokenizer(lower = F) %>% fit_text_tokenizer(news_new)


#put integers in a sequence

sequences<- texts_to_sequences(tokenizer, news_new)


library(MASS)


#Document-term matrix all

un.words <- length(tokenizer$word_index)

docs <- length(sequences)

dtm <- matrix(0, nrow = docs, ncol = un.words)

for (i in 1:docs){

  word.vec <- sequences[[i]]

  c <- length(sequences[[i]])

  a <- 0

  for (j in 1:c){

    a <- a + 1

    b <- word.vec[a]

    dtm[i,b] <- dtm[i,b] + 1

  }

}


dtm_y <- cbind(y, dtm)

dtm_y <- as.data.frame(dtm_y)
```

```
words <- unlist(tokenizer$index_word, use.names = TRUE)
words <- c("PolitiFactRating", words)

colnames(dtm_y) <- words

trainIndex <- createDataPartition(dtm_y$PolitiFactRating,
        p = .8, list = F, times = 1)

dtm_train_y <- dtm_y[trainIndex,]
dtm_test_y <- dtm_y[-trainIndex,]

dtm_train_y$PolitiFactRating <- as.factor(dtm_train_y$PolitiFactRating)
dtm_test_y$PolitiFactRating <- as.factor(dtm_test_y$PolitiFactRating)
#dtm_train_y$Year <- as.factor(dtm_train_y$Year)
#dtm_test_y$Year <- as.factor(dtm_test_y$Year)

dtm_train <- dtm_train_y[,-c(1)]
pca_train <- prcomp(dtm_train, center = T)
pct_train <- (pca_train$sdev^2/sum(pca_train$sdev^2))*100
sum(pct_train[1:2000])
embeds <- pca_train$rotation[,1:2000]
pc_train <- as.matrix(dtm_train) %*% as.matrix(embeds)

pc_train_y <- cbind(dtm_train_y[,c(1)],pc_train)
pc_train_y <- as.data.frame(pc_train_y)
names(pc_train_y)[names(pc_train_y) == 'V1'] <- 'PolitiFactRating'
pc_train_y$PolitiFactRating <- as.factor(pc_train_y$PolitiFactRating)

identical(pc_train_y$PolitiFactRating, dtm_train_y[,1])

pc_test <- as.matrix(dtm_test_y[,-c(1)]) %*% as.matrix(embeds)
```

```
pc_test_y <- cbind( dtm_test_y[,c(1)], pc_test)
pc_test_y <- as.data.frame(pc_test_y)
names(pc_test_y)[names(pc_test_y) == 'V1'] <- 'PolitiFactRating'
pc_test_y$PolitiFactRating <- as.factor(pc_test_y$PolitiFactRating)

identical(pc_test_y$PolitiFactRating, dtm_test_y[,1])

###################### PCA CV ################

train.control <- trainControl(method = "cv", number = 10)
cl <- makePSOCKcluster(5)
registerDoParallel(cl)

#Ordinal Logistic regression
#library(MASS)
#start_time <- Sys.time()
#model <- train(PolitiFactRating ~ ., data = pc_train_y,
#               trControl = train.control, method = "polr",
#               control = list(maxit = 100))
#end_time <- Sys.time()
#glm.time <- end_time - start_time
#glm.time
#model
#predictions <- model %>% predict(pc_test_y)
#confusionMatrix(predictions,pc_test_y$PolitiFactRating)

#Linear Discriminant Analysis directly on DTM
start_time <- Sys.time()
model <- train(PolitiFactRating ~ ., data = pc_train_y,
               trControl = train.control, method = "lda")
```

```
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(pc_test_y)
confusionMatrix(predictions, pc_test_y$PolitiFactRating)

#Quadratic Discriminant Analysis directly on DTM
#start_time <- Sys.time()
#model <- train(PolitiFactRating ~ ., data = pc_train_y,
#               trControl = train.control, method = "qda")
#end_time <- Sys.time()
#glm.time <- end_time - start_time
#glm.time
#model
#predictions <- model %>% predict(pc_test_y)
#confusionMatrix(predictions, pc_test_y$PolitiFactRating)

#Mixture Discriminant Analysis directly on DTM
library(mda)
start_time <- Sys.time()
model <- train(PolitiFactRating ~ ., data = pc_train_y,
               trControl = train.control, method = "mda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(pc_test_y)
confusionMatrix(predictions, pc_test_y$PolitiFactRating)

#Flexible Discriminant Analysis directly on DTM
```

```
library(earth)
start_time <- Sys.time()
model <- train(PolitiFactRating ~ ., data = pc_train_y,
               trControl = train.control, method = "fda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(pc_test_y)
confusionMatrix(predictions, pc_test_y$PolitiFactRating)

#Classification trees directly on DTM
library(rpart)
start_time <- Sys.time()
cpGrid <- expand.grid(.cp=seq(0.01, 0.5,0.01))
model <- train(PolitiFactRating ~ ., data = pc_train_y,
               trControl = train.control, method = "rpart",
               tuneGrid = cpGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(pc_test_y)
confusionMatrix(predictions, pc_test_y$PolitiFactRating)

### Random Forest
library(randomForest)
start_time <- Sys.time()
mtry <- floor(sqrt(ncol(pc_train_y)-1))
if (mtry > 10){
  a <- mtry - 10
} else {a <- mtry}
b <- mtry + 10
mtryGrid <- expand.grid(.mtry=seq(a, b, 1))
model <- train(PolitiFactRating ~ ., data = pc_train_y,
               trControl = train.control, method = "rf",
               tuneGrid = mtryGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(pc_test_y)
confusionMatrix(predictions, pc_test_y$PolitiFactRating)
```

Word2Vec Continuous Bag of Words

```
library(tidyverse)
library(tidytext)
library(textdata)
library(text2vec)
library(keras)
library(uwot)
library(tensorflow)
library(nomclust)
library(ggplot2)
library(word2vec)
library(caret)
library(doParallel)
library(nnet)

set.seed(121565)
```

```
fnn_train <- read.csv(file = "/work/wtp/jhaus4/fnn_train.csv",
                      header = T,
                      stringsAsFactors = F)
fnn_test <- read.csv(file = "/work/wtp/jhaus4/fnn_test.csv",
                      header = T,
                     stringsAsFactors = F)
fnn_valid <- read.csv(file = "/work/wtp/jhaus4/fnn_dev.csv",
                      header = T,
                      stringsAsFactors = F)
fnn <- rbind(fnn_train, fnn_valid, fnn_test)

LIAR_train <- read.csv(file = "/work/wtp/jhaus4/liar_train.csv",
                      header = T,
                       stringsAsFactors = F)
LIAR_test <- read.csv(file = "/work/wtp/jhaus4/liar_test.csv",
                      header = T,
                      stringsAsFactors = F)
LIAR_valid <- read.csv(file = "/work/wtp/jhaus4/liar_dev.csv",
                      header = T,
                       stringsAsFactors = F)
LIAR <- rbind(LIAR_train, LIAR_valid, LIAR_test)

common <- intersect(fnn$id, LIAR$id)
rawdata <- LIAR[LIAR$id %in% common, ]

data <- rawdata %>%
  mutate(newsTextClean=gsub('[[:punct:]]+', '',
                       gsub('\\\\n|\\.|\\,|\\;',' ',
                       gsub("[^[:alnum:]]", " ",
                       tolower(substr(statement,1,nchar(statement)-1))))))
```

```
data <- data %>% separate(date,
        c("Year", "Month", "Day","Junk"), sep = "-")

news_tokens <- data %>% select(id, label.liar, Year, newsTextClean) %>%
  unnest_tokens(word, newsTextClean, to_lower = FALSE) %>%
  #Use just "[a-z']+" for lower case letters
  mutate(word = str_extract(word, "[a-z']+"))
news_tokens <- news_tokens %>% drop_na(word)
#news_tokens <- news_tokens %>% anti_join(stop_words)

news_new <- news_tokens %>% group_by(word) %>%
  mutate(token_freq = n()) %>%
  filter(token_freq >= 30) %>%
  group_by(id, label.liar, Year) %>%
  summarise(newsTextClean = str_c(word, collapse = " "))

#Use just "[a-z']+" for lower case letters
max.length <- max(str_count(news_new$newsTextClean, "[a-z']+"))

trainIndex <- createDataPartition(news_new$label.liar,
      p = .8, list = F, times = 1)
train.control <- trainControl(method = "cv", number = 10)
cl <- makePSOCKcluster(5)
registerDoParallel(cl)

news_new_train <- news_new[trainIndex,]
news_new_test <- news_new[-trainIndex,]

#Getting the labels
y_train <- news_new_train %>% select(label.liar) %>% pull() %>% as.array()
y_test <- news_new_test %>% select(label.liar) %>% pull() %>% as.array()
```

```
y_train <- as.factor(y_train)
y_test <- as.factor(y_test)
levels(y_train)
levels(y_test)
levels(y_train) <- c("true", "mostly-true", "half-true",
    "barely-true", "false", "pants-fire")
levels(y_test) <- c("true", "mostly-true", "half-true",
    "barely-true", "false", "pants-fire")

#year_train <- news_new_train %>% select(Year) %>% pull() %>% as.array()
#year_test <- news_new_test %>% select(Year) %>% pull() %>% as.array()
#year_train <- as.factor(year_train)
#year_test <- as.factor(year_test)
#levels(year_train)
#levels(year_test)

it_train <- itoken(news_new_train$newsTextClean,
             tokenizer = word_tokenizer,
             ids = news_new$id,
             progressbar = T, tolower = FALSE)
it_test <- itoken(news_new_test$newsTextClean,
                   tokenizer = word_tokenizer,
                   ids = news_new$id,
                   progressbar = T, tolower = FALSE)
vocab_train <- create_vocabulary(it_train)
vocab_test <- create_vocabulary(it_test)
vocab_train <- prune_vocabulary(vocab_train)
vocab_test <- prune_vocabulary(vocab_test)
vectorizer_train <- vocab_vectorizer(vocab_train)
vectorizer_test <- vocab_vectorizer(vocab_test)
news_new_train <- news_new_train %>% select(newsTextClean) %>% pull()
```

```
news_new_test <- news_new_test %>% select(newsTextClean) %>% pull()

#Vectorize tokens - token receiving a unique integer
tokenizer_train <- text_tokenizer(lower = FALSE) %>%
    fit_text_tokenizer(news_new_train)
tokenizer_test <- text_tokenizer(lower = FALSE) %>%
    fit_text_tokenizer(news_new_test)

#put integers in a sequence
sequences_train <- texts_to_sequences(tokenizer_train, news_new_train)
sequences_test <- texts_to_sequences(tokenizer_test, news_new_test)

dtm_train <- create_dtm(it_train, vectorizer_train)
dtm_train <- as.matrix(dtm_train)
dtm_test <- create_dtm(it_test, vectorizer_test)
dtm_test <- as.matrix(dtm_test)

############WORD2VEC WITH 300 DIM##################
model_train <- word2vec(x = news_new_train, dim = 300, iter = 100,
    min_count = 0L)
model_train$success
emb_train <- as.matrix(model_train)
emb_train[1:5, 1:5]
emb_train_ord <- emb_train[order(rownames(emb_train)),]
emb_train_ord[1:5, 1:5]

### Getting embeddings per document
intersection <- colnames(dtm_train) %in% rownames(emb_train_ord)
dtm_train <- dtm_train[,intersection]
words <- colnames(dtm_train)
dtm_emb_train <- matrix(0, nrow = nrow(dtm_train),
    ncol = ncol(emb_train_ord))
for (i in 1:nrow(dtm_train)){
 for (j in 1:ncol(dtm_train)){
   if (dtm_train[i,j] != 0){
     dtm_emb_train[i,] <- dtm_emb_train[i,] + dtm_train[i,j] *
       emb_train_ord[which(rownames(emb_train_ord) == words[j]),]
   }
 }
}
dtm_emb_train[1:5, 1:5]
max(dtm_emb_train)

int_test <- colnames(dtm_test) %in% rownames(emb_train_ord)
dtm_test <- dtm_test[,int_test]
words_test <- colnames(dtm_test)
dtm_emb_test <- matrix(0, nrow = nrow(dtm_test), ncol = ncol(emb_train_ord))
for (i in 1:nrow(dtm_test)){
 for (j in 1:ncol(dtm_test)){
   if (dtm_test[i,j] != 0){
     dtm_emb_test[i,] <- dtm_emb_test[i,] + dtm_test[i,j] *
       emb_train_ord[which(rownames(emb_train_ord) == words_test[j]),]
   }
 }
}
dtm_emb_test[1:5, 1:5]
max(dtm_emb_test)

dtm_train_y <- cbind(dtm_emb_train, y_train)
dtm_train_y <- as.data.frame(dtm_train_y)
dtm_test_y <- cbind(dtm_emb_test, y_test)
dtm_test_y <- as.data.frame(dtm_test_y)
```

```
#dtm_test_y <- dtm_test_y %>% rename("year_train"="year_test")

dtm_train_y$y_train <- as.factor(dtm_train_y$y_train)
dtm_test_y$y_test <- as.factor(dtm_test_y$y_test)
#dtm_train_y$year_train <- as.factor(dtm_train_y$year_train)
#dtm_test_y$year_test <- as.factor(dtm_test_y$year_test)

 ##### CV 300 DIM#####

#Ordinal Logistic regression
library(MASS)
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "polr",
               control = list(maxit = 100))
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Linear Discriminant Analysis directly on DTM
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "lda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
```

```
confusionMatrix(predictions,dtm_test_y$y_test)

#Quadratic Discriminant Analysis directly on DTM
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "qda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

# #Mixture Discriminant Analysis directly on DTM
library(mda)
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "mda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

# #Flexible Discriminant Analysis directly on DTM
library(earth)
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "fda")
end_time <- Sys.time()
```

```
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

# #Classification trees directly on DTM
library(rpart)
start_time <- Sys.time()
cpGrid <- expand.grid(.cp=seq(0.01, 0.5,0.01))
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "rpart",
              tuneGrid = cpGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

# ### Random Forest
library(randomForest)
start_time <- Sys.time()
mtry <- floor(sqrt(ncol(dtm_train_y)-1))
if (mtry > 10){
 a <- mtry - 10
} else {a <- mtry}
b <- mtry + 10
mtryGrid <- expand.grid(.mtry=seq(a, b, 1))
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "rf",
              tuneGrid = mtryGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)
```

Word2Vec Skip-Gram

```
library(tidyverse)
library(tidytext)
library(textdata)
library(text2vec)
library(keras)
library(uwot)
library(tensorflow)
library(nomclust)
library(ggplot2)
library(word2vec)
library(caret)
library(doParallel)
library(nnet)

set.seed(121565)

fnn_train <- read.csv(file = "/work/wtp/jhaus4/fnn_train.csv",
                      header = T,
                      stringsAsFactors = F)
fnn_test <- read.csv(file = "/work/wtp/jhaus4/fnn_test.csv",
                      header = T,
                     stringsAsFactors = F)
fnn_valid <- read.csv(file = "/work/wtp/jhaus4/fnn_dev.csv",
                      header = T,
                      stringsAsFactors = F)
fnn <- rbind(fnn_train, fnn_valid, fnn_test)

LIAR_train <- read.csv(file = "/work/wtp/jhaus4/liar_train.csv",
                        header = T,
                       stringsAsFactors = F)
LIAR_test <- read.csv(file = "/work/wtp/jhaus4/liar_test.csv",
                      header = T,
                      stringsAsFactors = F)
LIAR_valid <- read.csv(file = "/work/wtp/jhaus4/liar_dev.csv",
                        header = T,
                       stringsAsFactors = F)
LIAR <- rbind(LIAR_train, LIAR_valid, LIAR_test)

common <- intersect(fnn$id, LIAR$id)
rawdata <- LIAR[LIAR$id %in% common, ]

data <- rawdata %>%
  mutate(newsTextClean=gsub('[[:punct:]]+', '',
                       gsub('\\\\n|\\.|\\,|\\;',' ',
                       gsub("[^[:alnum:]]", " ",
                       tolower(substr(statement,1,nchar(statement)-1))))))

data <- data %>% separate(date,
    c("Year", "Month", "Day","Junk"), sep = "-")

news_tokens <- data %>% select(id, label.liar, Year, newsTextClean) %>%
  unnest_tokens(word, newsTextClean, to_lower = FALSE) %>%
  #Use just "[a-z']+" for lower case letters
  mutate(word = str_extract(word, "[a-z']+"))
news_tokens <- news_tokens %>% drop_na(word)
#news_tokens <- news_tokens %>% anti_join(stop_words)

news_new <- news_tokens %>% group_by(word) %>%
  mutate(token_freq = n()) %>%
  filter(token_freq >= 30) %>%
  group_by(id, label.liar, Year) %>%
  summarise(newsTextClean = str_c(word, collapse = " "))

#Use just "[a-z']+" for lower case letters
max.length <- max(str_count(news_new$newsTextClean, "[a-z']+"))

trainIndex <- createDataPartition(news_new$label.liar,
    p = .8, list = F, times = 1)
train.control <- trainControl(method = "cv", number = 10)
cl <- makePSOCKcluster(5)
registerDoParallel(cl)

news_new_train <- news_new[trainIndex,]
news_new_test <- news_new[-trainIndex,]

#Getting the labels
y_train <- news_new_train %>% select(label.liar) %>% pull() %>% as.array()
y_test <- news_new_test %>% select(label.liar) %>% pull() %>% as.array()
y_train <- as.factor(y_train)
y_test <- as.factor(y_test)
levels(y_train)
levels(y_test)
levels(y_train) <- c("true", "mostly-true", "half-true",
  "barely-true", "false", "pants-fire")
levels(y_test) <- c("true", "mostly-true", "half-true",
  "barely-true", "false", "pants-fire")

#year_train <- news_new_train %>% select(Year) %>% pull() %>% as.array()
#year_test <- news_new_test %>% select(Year) %>% pull() %>% as.array()
#year_train <- as.factor(year_train)
#year_test <- as.factor(year_test)
#levels(year_train)
#levels(year_test)

it_train <- itoken(news_new_train$newsTextClean,
                   tokenizer = word_tokenizer,
                   ids = news_new$id,
                   progressbar = T, tolower = FALSE)
it_test <- itoken(news_new_test$newsTextClean,
                  tokenizer = word_tokenizer,
                  ids = news_new$id,
                  progressbar = T, tolower = FALSE)
vocab_train <- create_vocabulary(it_train)
vocab_test <- create_vocabulary(it_test)
vocab_train <- prune_vocabulary(vocab_train)
vocab_test <- prune_vocabulary(vocab_test)
vectorizer_train <- vocab_vectorizer(vocab_train)
vectorizer_test <- vocab_vectorizer(vocab_test)
news_new_train <- news_new_train %>% select(newsTextClean) %>% pull()
news_new_test <- news_new_test %>% select(newsTextClean) %>% pull()

#Vectorize tokens - token receiving a unique integer
tokenizer_train <- text_tokenizer(lower = FALSE) %>%
    fit_text_tokenizer(news_new_train)
```

```
tokenizer_test <- text_tokenizer(lower = FALSE) %>%
    fit_text_tokenizer(news_new_test)

#put integers in a sequence
sequences_train <- texts_to_sequences(tokenizer_train, news_new_train)
sequences_test <- texts_to_sequences(tokenizer_test, news_new_test)

dtm_train <- create_dtm(it_train, vectorizer_train)
dtm_train <- as.matrix(dtm_train)
dtm_test <- create_dtm(it_test, vectorizer_test)
dtm_test <- as.matrix(dtm_test)

############WORD2VEC SG WITH 300 DIM##################
model_train <- word2vec(x = news_new_train, type = "skip-gram",
                        dim = 300, iter = 100, min_count = 0L)
model_train$success
emb_train <- as.matrix(model_train)
emb_train_ord <- emb_train[order(rownames(emb_train)),]

### Getting embeddings per document
intersection <- colnames(dtm_train) %in% rownames(emb_train_ord)
dtm_train <- dtm_train[,intersection]
words <- colnames(dtm_train)
dtm_emb_train <- matrix(0, nrow = nrow(dtm_train),
  ncol = ncol(emb_train_ord))
for (i in 1:nrow(dtm_train)){
  for (j in 1:ncol(dtm_train)){
    if (dtm_train[i,j] != 0){
      dtm_emb_train[i,] <- dtm_emb_train[i,] + dtm_train[i,j] *
        emb_train_ord[which(rownames(emb_train_ord) == words[j]),]
    }
  }
}

int_test <- colnames(dtm_test) %in% rownames(emb_train_ord)
dtm_test <- dtm_test[,int_test]
words_test <- colnames(dtm_test)
dtm_emb_test <- matrix(0, nrow = nrow(dtm_test),
  ncol = ncol(emb_train_ord))
for (i in 1:nrow(dtm_test)){
  for (j in 1:ncol(dtm_test)){
    if (dtm_test[i,j] != 0){
      dtm_emb_test[i,] <- dtm_emb_test[i,] + dtm_test[i,j] *
        emb_train_ord[which(rownames(emb_train_ord) == words_test[j]),]
    }
  }
}

dtm_train_y <- cbind(dtm_emb_train, y_train)
dtm_train_y <- as.data.frame(dtm_train_y)
dtm_test_y <- cbind(dtm_emb_test, y_test)
dtm_test_y <- as.data.frame(dtm_test_y)
#dtm_test_y <- dtm_test_y %>% rename("year_train"="year_test")

dtm_train_y$y_train <- as.factor(dtm_train_y$y_train)
dtm_test_y$y_test <- as.factor(dtm_test_y$y_test)
#dtm_train_y$year_train <- as.factor(dtm_train_y$year_train)
#dtm_test_y$year_test <- as.factor(dtm_test_y$year_test)

##### CV 300 DIM#####

#Ordinal Logistic regression
```

```
library(MASS)
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "polr",
               control = list(maxit = 100))
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Linear Discriminant Analysis directly on DTM
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "lda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Quadratic Discriminant Analysis directly on DTM
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "qda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
```

```
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Mixture Discriminant Analysis directly on DTM
library(mda)
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "mda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Flexible Discriminant Analysis directly on DTM
library(earth)
start_time <- Sys.time()
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "fda")
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

#Classification trees directly on DTM
library(rpart)
start_time <- Sys.time()
cpGrid <- expand.grid(.cp=seq(0.01, 0.5,0.01))
```

```
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "rpart",
               tuneGrid = cpGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)

### Random Forest
library(randomForest)
start_time <- Sys.time()
mtry <- floor(sqrt(ncol(dtm_train_y)-1))
if (mtry > 10){
  a <- mtry - 10
} else {a <- mtry}
b <- mtry + 10
mtryGrid <- expand.grid(.mtry=seq(a, b, 1))
model <- train(y_train ~ ., data = dtm_train_y,
               trControl = train.control, method = "rf",
               tuneGrid = mtryGrid)
end_time <- Sys.time()
glm.time <- end_time - start_time
glm.time
model
predictions <- model %>% predict(dtm_test_y)
confusionMatrix(predictions,dtm_test_y$y_test)
```

References

10 Ahmed, H., Traore, I., & Saad, S. (2017). Detecting opinion spams and fake news using text classification. *Security and Privacy*, *1*(1), e9. http://doi.org/10.1002/spy2.9

Alammar, J. (2018a, December). The illustrated bert, elmo, and co. (How NLP cracked transfer learning). GitHub. Retrieved from `https://jalammar.github.io/illustrated-bert/`

Alammar, J. (2018b, June). The illustrated transformer. GitHub. Retrieved from `https://jalammar.github.io/illustrated-transformer/`

Allaire, J., & Chollet, F. (2022). *Keras: R interface to 'keras'*. Retrieved from `https://CRAN.R-project.org/package=keras`

Allaire, J., & Tang, Y. (2022). *Tensorflow: R interface to 'TensorFlow'*. Retrieved from `https://CRAN.R-project.org/package=tensorflow`

Barushka, A., & Hajek, P. (2019). The effect of text preprocessing strategies on detecting fake consumer reviews. In *Proceedings of the 2019 3rd international conference on e-business and internet*. ACM. http://doi.org/10.1145/3383902.3383908

Bikienga, S. (2018). *Leadership and economic growth: A text analytics approach* (PhD thesis).

Boom, C. D., Canneyt, S. V., Demeester, T., & Dhoedt, B. (2016). Representation learning for very short texts using weighted word embedding aggregation. http://doi.org/10.1016/j.patrec.2016.06.012

Breiman, L., & Cutler, A. (2004, March). Random forests. Retrieved from `https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm`

Chaubard, F., Fang, M., Genthial, G., Mundra, R., & Socher, R. (2019). CS224n: Natural language processing with deep learning. Retrieved from `https://web.stanford.edu/class/cs224n/readings/cs224n-2019-notes01-wordvecs1.pdf`

Coronel, J. C., Poulsen, S., & Sweitzer, M. D. (2019). Investigating the Generation and Spread of Numerical Misinformation: A Combined Eye Movement Monitoring and Social Transmission Approach. *Human Communication Research*, *46*(1), 25–54. http://doi.org/10.1093/hcr/hqz012

Corporation, M., & Weston, S. (2022). *doParallel: Foreach parallel adaptor for the 'parallel' package*. Retrieved from `https://CRAN.R-project.org/package=doParallel`

DataTechNotes. (2019, February). Precision, recall, specificity, prevalence, kappa, F1-score check with r. *DataTechNotes: A blog about data science and machine learning*. Retrieved from `https://www.datatechnotes.com/2019/02/accuracy-metrics-in-classification.html`

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, & T. Solorio (Eds.), *Proceedings of the 2019 conference of the north american chapter of the association for computational linguistics: Human language technologies, NAACL-HLT 2019, minneapolis, MN, USA, june 2-7, 2019, volume 1 (long and short papers)* (pp. 4171–4186). Association for Computational Linguistics. http://doi.org/10.18653/v1/n19-1423

Dietrich, J. P., & Leoncio, W. (2023). *Citation: Software citation tools.*

Fox, J., & Weisberg, S. (2019). *An R companion to applied regression* (Third). Thousand Oaks CA: Sage. Retrieved from https://socialsciences.mcmaster.ca/jfox/Books/Companion/

Grave, E., Bojanowski, P., Gupta, P., Joulin, A., & Mikolov, T. (2018). Learning word vectors for 157 languages. In N. Calzolari, K. Choukri, C. Cieri, T. Declerck, S. Goggi, K. Hasida, … T. Tokunaga (Eds.), *Proceedings of the eleventh international conference on language resources and evaluation, LREC 2018, miyazaki, japan, may 7-12, 2018*. European Language Resources Association (ELRA). Retrieved from http://www.lrec-conf.org/proceedings/lrec2018/summaries/627.html

Harris, Z. S. (1954). Distributional structure, *10*(2-3), 146–162. http://doi.org/10.1080/00437956.1954.11659520

Hastie, T., & Tibshirani, R. (1996). Discriminant analysis by Gaussian mixtures. *Journal of the Royal Statistical Society. Series B. Methodological*, *58*(1), 155–176.

Hastie, T., Tibshirani, R., & Buja, A. (1994). Flexible discriminant analysis by optimal scoring. *Journal of the American Statistical Association, 89*(428), 1255–1270.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). Flexible discriminants. In *The elements of statistical learning, second edition: Data mining, inference, and prediction* (pp. 417–458). Springer.

Holan, A. D. (2020, October). PolitiFact - the principles of the TRUTH-o-METER: PolitiFact's methodology for independent fact-checking. Retrieved from https://www.politifact.com/article/2018/feb/12/principles-truth-o-meter-politifacts-methodology-i/

Horev, R. (2018, November). Bert explained: State of the art language model for NLP. *Medium.* Towards Data Science. Retrieved from https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

Horne, B. D., & Adali, S. (2017). This just in: Fake news packs a lot in title, uses simpler, repetitive content in text body, more similar to satire than real news. *CoRR*, *abs/1703.09398.* Retrieved from http://arxiv.org/abs/1703.09398

Hua, C., Choi, Y.-J., & Shi, Q. (2021, April). Companion to BER 642: Advanced regression methods. *Chapter 12 Ordinal Logistic Regression.* Bookdown. Retrieved from https://bookdown.org/chua/ber642_advanced_regression/ordinal-logistic-regression.html

Jeppson, H., Hofmann, H., & Cook, D. (2023). *Ggmosaic: Mosaic plots in the 'ggplot2' framework.* Retrieved from https://github.com/haley jeppson/ggmosaic

Johnson, R., & Wichern, D. (2008). In *Applied multivariate statistical analysis* (6th ed.). Pearson.

Karani, D. (2020, September). Introduction to word embedding and Word2Vec. *Medium*. Towards Data Science. Retrieved from https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa

Kuhn, M. (2022). *Caret: Classification and regression training*. Retrieved from https://CRAN.R-project.org/package=caret

Kwartler, T. (2017). *Text mining in practice with r*. Wiley.

Landauer, T. K., & Dumais, S. T. (1997). A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, *104*(2), 211–240. http://doi.org/10.1037/0033-295x.104.2.211

Leung, K. (2022, January). Micro, macro & weighted averages of F1 score, clearly explained. *Towards Data Science*. Towards Data Science. Retrieved from https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f

Li, J. (2017). Mixture discriminant analysis. *Class Notes*. Retrieved from http://www.personal.psu.edu/jol2/course/stat597e/notes2/mda.pdf

Liaw, A., & Wiener, M. (2002). Classification and regression by randomForest. *R News*, *2*(3), 18–22. Retrieved from https://CRAN.R-proje ct.org/doc/Rnews/

McCormick, C. (2016, April). Word2Vec tutorial - THE skip-gram model. Retrieved from http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Melville, J. (2022). *Uwot: The uniform manifold approximation and projection (UMAP) method for dimensionality reduction.* Retrieved from https://CRAN.R-project.org/package=uwot

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013, January). Efficient estimation of word representations in vector space. ICLR. Retrieved from https://arxiv.org/abs/1301.3781

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in Neural Network Information Processing Systems*, *26*, 3111–3119. Retrieved from https://arxiv.org/abs/1310.4546

Ott, M., Cardie, C., & Hancock, J. T. (2013). Negative deceptive opinion spam. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies* (pp. 497–501).

Ott, M., Choi, Y., Cardie, C., & Hancock, J. T. (2011). Finding deceptive opinion spam by any stretch of the imagination. *Proceedings of ACL 2011: HLT, Pp. 309-319.* Retrieved from https://arxiv.org/abs/1107.4557v1

Palachy, S. (2019, October). Document embedding techniques. *Medium*. Towards Data Science. Retrieved from https://towardsdatascience.com/document-embedding-techniques-fed3e7a6a25d#ecd3

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532–1543). http://doi.org/10.3115/v1/d14-1162

Potthast, M., & Kiesel, et al., Johannes. (2018). A stylometric inquiry into hyperpartisan and fake news. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers)*, 231–240.

R Core Team. (2022). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from https://www.R-project.org/

Reimer, N. (2022). Computing sentence embeddings. *Computing Sentence Embeddings - Sentence-Transformers documentation*. Retrieved from https://www.sbert.net/examples/applications/computing-embeddings/README.html

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing*. Association for Computational Linguistics. Retrieved from http://arxiv.org/abs/1908.10084

Sadeghi, F., Jalaly Bidgoly, A., & Amirkhani, H. (2020). FNID: Fake news inference dataset [Data set]. IEEE Dataport. http://doi.org/10.21227/fbzdsw81

Sahil. (2021, February). Word embedding: glove. Medium. Retrieved from https://sahiltinky94.medium.com/word-embedding-glove-dd27f630c663

Selivanov, D., Bickel, M., & Wang, Q. (2022). *text2vec: Modern text mining framework for r.* Retrieved from https://CRAN.R-project.org/package=text2vec

Serrano, M. Ángeles, Flammini, A., & Menczer, F. (2009). Modeling statistical properties of written text. *PLoS ONE*, *4*(4), e5372. http://doi.org/10.1371/journal.pone.0005372

Shojaee, S., Murad, M. A. A., Azman, A. B., Sharef, N. M., & Nadali, S. (2013). Detecting deceptive reviews using lexical and syntactic features. In *2013 13th international conference on intellient systems design and applications*. IEEE. http://doi.org/10.1109/isda.2013.6920707

Shu, K., Sliva, A., Wang, S., Tang, J., & Liu, H. (2017). Fake news detection on social media. *ACM SIGKDD Explorations Newsletter*, *19*(1), 22–36. http://doi.org/10.1145/3137597.3137600

Silge, J., & Robinson, D. (2016). Tidytext: Text mining and analysis using tidy data principles in r. *JOSS*, *1*(3). http://doi.org/10.21105/joss.00037

Silge, J., & Robinson, D. (2021, June). Text mining with r: A tidy approach. *Welcome to Text Mining with R | Text Mining with R*. Retrieved from https://www.tidytextmining.com/index.html

Sulc, Z., Cibulkova, J., Rezankova, H., & Hornicek, J. (2022). *Nomclust: Hierarchical cluster analysis of nominal data*. Retrieved from https://CRAN.R-project.org/package=nomclust

Sungur, E. A. (2011). Classification & regression trees. Retrieved from http://mnstats.morris.umn.edu/multivariatestatistics/cart.html

Therneau, T., & Atkinson, B. (2022). *Rpart: Recursive partitioning and regression trees*. Retrieved from https://CRAN.R-project.org/package=rpart

Torregrossa, F., Allesiardo, R., Claveau, V., Kooli, N., & Gravier, G. (2021). A survey on training and evaluation of word embeddings. *International Journal of Data Science and Analytics*. http://doi.org/10.1007/s41060-021-00242-8

Trevor Hastie & Robert Tibshirani. Original R port by Friedrich Leisch, S. original by, Hornik, K., & code., B. D. Ripley. B. N. has contributed to the upgrading of the. (2022). *Mda: Mixture and flexible discriminant analysis*. Retrieved from https://CRAN.R-project.org/package=mda

Trevor Hastie, S. Milborrow. D. from mda:mars by, & Thomas Lumley's leaps wrapper., R. Tibshirani. U. A. M. F. utilities with. (2021). *Earth: Multivariate adaptive regression splines*. Retrieved from https://CRAN.R-project.org/package=earth

Uysal, A. K., & Gunal, S. (2014). The impact of preprocessing on text classification. *Information Processing & Management*, *50*(1), 104–112. http://doi.org/https://doi.org/10.1016/j.ipm.2013.08.006

Venables, W. N., & Ripley, B. D. (2002). *Modern applied statistics with s* (Fourth). New York: Springer. Retrieved from https://www.stats.ox.ac.uk/pub/MASS4/

Wickham, H. (2016). *ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York. Retrieved from https://ggplot2.tidyverse.org

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., … Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, *4*(43), 1686. http://doi.org/10.21105/joss.01686

Wijffels, J. (2021). *word2vec: Distributed representations of words*. Retrieved from https://CRAN.R-project.org/package=word2vec

Xie, Y. (2014). Knitr: A comprehensive tool for reproducible research in R. In V. Stodden, F. Leisch, & R. D. Peng (Eds.), *Implementing reproducible computational research*. Chapman; Hall/CRC.

Xie, Y. (2015). *Dynamic documents with R and knitr* (2nd ed.). Boca Raton, Florida: Chapman; Hall/CRC. Retrieved from https://yihui.org/knitr/

Xie, Y. (2023). *Knitr: A general-purpose package for dynamic report generation in r*. Retrieved from https://yihui.org/knitr/

Zheng, R., Li, J., Chen, H., & Huang, Z. (2006). A framework for authorship identification of online messages: Writing-style features and classification techniques. *Journal of the American Society for Information Science and Technology*, *57*(3), 378–393. http://doi.org/10.1002/asi.20316

Zhu, H. (2021). *kableExtra: Construct complex table with 'kable' and pipe syntax*. Retrieved from https://CRAN.R-project.org/package=kableExtra