2022

Insurance Meets Sentiment: An Empirical Study of Attitudes Toward Life, Health, and P&C Insurances

Akshi Agarwal
Hollins University

INSURANCE MEETS SENTIMENT: AN EMPIRICAL STUDY OF ATTITUDES TOWARD LIFE, HEALTH, AND P&C INSURANCES

Akshita Agarwal
Under the direction of Dr. Julie Clark

An honors thesis submitted in partial fulfillment
of the requirements for the degree of
Bachelor of Science
in Mathematics
at
Hollins University
Roanoke, Virginia
May 2022

Director of the Thesis: Julie M Clark
Professor Julie M. Clark

ABSTRACT

Sentiment Analysis, an up-and-coming subfield of Natural Language Processing (NLP), contains previously untapped potential that can be utilized to drive better business decision making. In this paper, we employ state-of-the-art sentiment analysis tools to compare the performances of traditional classification algorithms – namely Support Vector Machines (SVMs), bagging, boosting, random forest, and decision tree classifiers – on insurance-related textual data. We successfully demonstrate that algorithms such as bagging and boosting, which were constructed to enhance the performance of simpler algorithms such as decision tree classifiers, offer only marginal improvements in terms of classification accuracy and certain performance metrics for our data. However, the improved accuracy comes as the cost of slightly higher runtimes. Insurance companies could apply these findings to choose suitable algorithms and gain a more nuanced understanding of the needs of their insureds.

***Index Terms*—** Sentiment Analysis, Textual Analysis, Machine Learning, Natural Language Processing (NLP), Opinion Mining (OM)

1. INTRODUCTION

In 2007, a 17-year-old LA-based girl passed away when her health insurer, Cigna, one of the largest carriers in the world, reversed its decision to cover the liver transplant the girl needed [1]. In many cases, the insureds depend on their carriers to offset their financial burdens when they are in their most vulnerable state. The insurance companies, on the other hand, must balance the need to profitably provide their services while simultaneously manage the collective risk of meeting the financial needs of their insured. Often, the specifics of the insurance policies determine the extent to which a potentially life-saving treatment is accessible to an insured. Cigna's failure to properly assess this patient's case resulted in the loss of a young life and was arguably instrumental in shaping the public's attitudes towards health insurance carriers.

Economists coined the term *asymmetric information* to explain the uneven concentration of information between the buyer and the seller about the product being exchanged. Insurance markets are prime examples of markets whose players operate with asymmetric information: compared to their insurers, the insureds generally do not possess a thorough understanding of the potentially life-saving insurance products they purchase. Consequently, due to the unbalanced nature of the product and the exchange, the topic of insurance is fiercely debated in many countries, including the United States. In this paper, we aim to study attitudes toward life, health, and P&C insurances using tools from the field of sentiment analysis. As its name suggests, the problems in this field deal with measuring and predicting sentiments expressed through textual, auditory, or visual means. Sentiment-bearing data, such as tweets pertaining to the topic of insurance, when exploited correctly, can allow insurers to better understand the implications of the insurance products they design, which in turn might enable them to better meet the needs of their insureds. In this paper, we apply several machine learning classification algorithms to life, health, and P&C insurance-related tweets from around the world. Doing so will allow us to identify the algorithms that might be superior to others in predicting and explaining the sentiments expressed in insurance-related tweets.

We begin this thesis with a description of the two main methods of classifying sentiments (lexicon-based and machine learning approaches) in Section 2. In Sections 2.2.1-2.2.6, we provide a thorough survey of the machine learning algorithms whose comparative performance is of interest to us. We then outline the specifics of the design of our empirical study in Section 3. We conclude our paper with a discussion of the results in Section 4.

2. AN OVERVIEW OF LEXICON-BASED AND SELECT MACHINE LEARNING APPROACHES

Textual sentiment quantification uses tools from the broad field of Natural Language Processing (NLP) to quantify the sentiment of a given text. We discuss the lexicon-based and machine learning approaches as the two main methods of quantitatively estimating sentiments. These approaches are applied to the chosen data (textual) unit, which is usually a document, a paragraph, or a sentence. These approaches treat the random variable capturing sentiments as the response variable that needs to be predicted based on sentiment polarity of the data unit under study. This random variable is almost invariably a categorical variable with two or more classes. Since the problem of sentiment analysis deals with a categorical (rather than a quantitative) response variable, strictly speaking, it is a classification (rather than a regression) problem. All classification approaches, including that of sentiment analysis and prediction, aim to minimize the test error rate:

$$\text{Avg}(I(y_0 \neq \hat{y}_0)), \tag{1}$$

where (x_0, y_0) is a observation in the test set, I is the indicator function, and $\hat{y}_0$ is response value predicted using the input x_0. Analogously, regression problems aim to minimize the expected test mean squared error (MSE):

$$\begin{aligned} E\left(y_0 - \hat{f}(x_0)\right)^2 &= \text{Var}\left(\hat{f}(x_0)\right) + \text{Bias}\left(\hat{f}(x_0)\right)^2 \\ &\quad + \text{Var}\left(\varepsilon\right), \end{aligned} \tag{2}$$

where ε is the irreducible error whose presence indicates that any given statistical model $\hat{f}(\cdot)$ cannot explain the variations and trends in the test set perfectly. Although the test error rate for classification problems, as defined in (1), does not have the decomposition specified in (2), the ideas expressed in (2) nevertheless apply. To be explicit, the researcher needs to find a learning method $\hat{f}(\cdot)$ that simultaneously minimizes the variance and the bias associated with $\hat{f}(\cdot)$ in order to minimize the test error rate.

2.1. Lexicon-Based Approaches[1]

In this section, we discuss lexicon-based methods which lie at the core of the classification we will perform to assign an initial sentiment score to our textual data (for the purpose of assembling a training set). Turney (2002) defines lexicon-based sentiment analysis as "calculating sentiment for a document from the sentiment of words or phrases in the document." Naturally, researchers employing lexicon-based methods need a collection of tuples containing important unigrams (e.g., insurance) or n-grams (e.g., unfair health insurance, a 3-gram) and their corresponding sentiment scores for their analysis. Such a collection of tuples is known as a *sentiment lexicon*, which is matched with the text under consideration to decipher its sentiment.

Unsurprisingly, the task of assembling a suitable lexicon is complex. For instance, the researcher needs to decide upon an appropriate size for the lexicon. If one likens a lexicon to a training set, and the text under study to a test set, then a lexicon that is too small will likely underperform on the test data, and similarly, a lexicon that is too large will likely overfit the test set. Additionally, lexicons can be either general or domain-specific. General lexicons draw words from everyday language while domain-specific lexicons further contain words specific to the domain they are targeting. Examples of finance-specific lexicons include the Henry lexicon (Henry, 2008), which contains words such as "risk" and "penalty", and the Loughran and McDonald lexicon (Loughran and McDonald, 2011), which contains words including "cyberattack" and "refinancing". Political words such as "conservative", "coalition", and "greens" are contained in The Lexicoder Sentiment Dictionary (Young and Soroka, 2012), a politics-specific lexicon.

Liu (2005) sees three pure (non-hybrid) ways of generating lexicons: manually, dictionary-based, and corpus-based. To assemble a lexicon manually, the researcher defines a sentiment score scheme and then employs human annotators to assign a sentiment score to select words according to the scheme. Some well-known examples of manually-assembled lexicons include the Stone et al. General Inquirer (1963) and the Bradley and Lang ANEW words list (1999). The latter is a collection of about 1034 words [2]. Unlike the dictionary-based and corpus-based approaches, the researcher is not required to automate any aspects of the manual approach. Moreover, researchers have traditionally utilized manual approaches to obtain general (rather than domain-specific) lexicons. However, due to its cost-intensive and time-consuming nature, the manual approach can be inaccessible to some researchers.

Dictionary-based approaches can be considered a cost-effective extension of manual approaches. The researcher using this approach starts with a list of seed sentiment words whose polarity (the degree to which it expresses, for instance, positive, negative, and/or neutral sentiments) has been determined using manual approaches. The researcher then expands this list by using synonyms and antonyms coming from a large base dictionary, such as the WordNet database (Miller, 1995).

Corpus-based methods allow the researcher to obtain domain-specific lexicons by adapting an existing general lexicon to the targeted domain. To achieve this, the researcher exploits a priori syntactic patterns to include new words in the lexicon. Typically, the researcher relies on linguistic notions such as sentiment consistency (for example, adjectives with a similar sentiment orientation are believed to occur in groups) and sentiment coherency (for instance, the same sentiment orientation tends to be expressed in consecutive sentences and adversarial expressions such as "but" and "however" are used to signal changes in sentiment orientations) to compile a suitable domain-specific lexicon.

2.2. Machine Learning Approaches [2]

While using machine learning algorithms to perform sentiment analysis, the researcher first obtains (or, in some cases, constructs) an annotated dataset of text with corresponding sentiment values, such as a sentiment lexicon. Machine learning approaches differ from lexicon-based approaches in that they use sophisticated statistical techniques, such as regression methods, to assign sentiment scores to the text under consideration using the lexicon. In most cases, the random variable containing the sentiment scores are ordinal and can be either dichotomous or non-dichotomous, depending on whether the variable makes a distinction between positive and negative sentiments or positive, negative, and other chosen sentiments. Most non-dichotomous variables take on integer values between -1 and 1 (inclusive) to denote positive, negative, and neutral sentiments. Machine learning algorithms can then be deployed on this preprocessed dataset to identify the most important textual characteristics (i.e., words, n-grams, phrases, counts, etc.) in measuring sentiment.

Machine learning can be branched into supervised and unsupervised learning. Let

$$\mathbf{X}^T = [\mathbf{X}_1, \ldots, \mathbf{X}_p]^T \tag{3}$$

denote the set of independent variables under consideration, where $\mathbf{X}_j = x_{j,1}, x_{j,2}, \ldots, x_{j,n}, 1 \leq j \leq p$ for p predic-

[1]SentiStrength, the software we used to assign an initial, benchmark sentiment score to our textual data utilizes lexicon-based approaches. For details, refer to Section 3.2.

[2]We compare the performance of machine learning approaches in the empirical analysis section of this paper.

tors and n observations. Similarly, let $\mathbf{Y}$ denote the response variable, where $\mathbf{Y} = y_1, \ldots, y_n$. Supervised machine learning algorithms aim to delineate the relationship between $\mathbf{X}^T$ and the supervising response variable $\mathbf{Y}$, with the response variable guiding the algorithm throughout the analysis. Supervised machine learning models can be used to fulfill two, often competing, purposes: to accurately predict $\mathbf{Y}$ using a model based on $\mathbf{X}^T$, or to infer the relationship between $\mathbf{Y}$ and $\mathbf{X}^T$. Examples of supervised machine learning algorithms include multiple linear regression, maximum entropy classifier, bootstrap aggregation (often referred to as "bagging"), and boosting. Any of these algorithms can be used if prediction is the goal, although more flexible methods such as thin plate regression splines, which model the dataset more closely without being restricted by a specific model $f(\mathbf{X})$, are preferred. On the other hand, less flexible approaches such as linear regression, which allow for easy interpretation, are favored when inference is the overall goal of the analysis.

Researchers studying sentiment analysis have used creative ways to initially assign sentiment scores to the observations in the training set. Those studying sentiments in reviews posted on websites that offer users the option to provide a star-based rating with their text-based reviews, such as Amazon, utilize the star ratings as a basis for the initial sentiment scores. Researchers also utilize emotion icons (emoticons) to decipher the sentiments of the reviews left on feedback-collecting websites that collect reviews without offering reviewers the option to provide a star-based rating along with their textual review [3]. Often, researchers enlist human volunteers, along with the automated techniques described above, to assign an initial sentiment score to the reviews, ensuring the accuracy of the initial sentiment scores and accounting for the nuances of human language.

Contrastingly, unsupervised learning algorithms do not aim to predict $\mathbf{Y}$ based on $\mathbf{X}^T$ or infer conclusions about the relationship between $\mathbf{Y}$ and $\mathbf{X}^T$. Unsupervised machine learning refers to the peculiar situation where analysis is performed solely on $\mathbf{X}^T$, and $\mathbf{Y}$ is either disregarded or absent. These algorithms can be used to study the relationship between the observations $1, \ldots, n$, or that between the predictors $\mathbf{X}_1, \ldots, \mathbf{X}_n$. For example, an analyst at a health insurance company with access to data on demographic characteristics such as age, income level, etc. on a group of prospective insureds can employ clustering analysis (a type of unsupervised analysis) to classify them into various non-overlapping and exhaustive groups based on their characteristics. In doing so, the analyst must ensure that the members of each group possess characteristics that are similar to that of the other members of their group but dissimilar from the characteristics of those in other groups. In this case, the needs of the members of these subgroups might be better managed by offering the subgroups specific policies tailored to their needs, instead of using a blanket policy for all. Principal Component Analysis (PCA) is another type of unsupervised machine learning technique which aims to find low-dimensional representations of the predictor space, giving a clearer picture of the predictors when some are correlated. Since sentiment analysis is inherently a supervised problem, we did not utilize unsupervised methods in our study.

2.2.1. Maximum Entropy Classifier

Statisticians hold that, on average, (1) is minimized by the Naive Bayes algorithm, which assigns each test observation to its most likely class, given the predictor values associated with it. However, the predictive accuracy of the Naive Bayes algorithm dwindles when the predictors are not mutually independent. Although this approach can take more time to train, the Maximum Entropy (ME) Classifier can be used to reliably classify dependent predictors. This classifier is based on the idea that we should "model all that is known and assume nothing about that which is unknown" [4].

ME is a type of probabilistic classifier. Classification problems aim to identify certain features of the (test) observations to classify them into one of the predefined classes. For instance, an analyst at a life insurance company might classify a potential client as high-risk or low-risk, depending on whether the client smokes regularly. If the analyst were to use a probabilistic classifier in their modeling, the classifier would not only classify the client as high- or low-risk, but it would also give the analyst the probability of the client being in the high- and low-risk classes. To be precise, ME extracts certain features of the test observation, obtains a linear combination of the features and their corresponding weights, and uses the sum as an exponent in the modeling process.

Suppose we are trying to classify a test observation with input vector $\mathbf{x}$ into one of the C classes $c_1, \ldots, c_C$. For each of the N observations in the training set, $i = 1, \ldots, N$, let $f_i(c, \mathbf{x})$ denote an indicator function dependent on $\mathbf{x}$ and the response class c:

$$f_i(c, \mathbf{x}) = \begin{cases} 1 & \text{some criterion based on } c \text{ and } \mathbf{x} \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Further assume that w_{ci} denotes the corresponding weights. (The weights are determined by the researcher using cross validation.) The ME classifier uses (5) to compute the probability of $\mathbf{y}$ belonging to class c, given the predictor vector $\mathbf{x}$:

$$Pr(c \mid \mathbf{x}) = \frac{\exp\left(\sum_{i=1}^{N} w_{ci} f_i(c, \mathbf{x})\right)}{\sum_{c' \in C} \exp\left(\sum_{i=1}^{N} w_{c'i} f_i(c', \mathbf{x})\right)}. \tag{5}$$

The classifier classifies the test observation in the most probable response class. Note that the denominator in (5) is known as the normalization factor because its primary purpose is to transform the exponential into a true probability with a value that lies non-strictly between 0 and 1.

2.2.2. *Support Vector Machine*

Support Vector Machine (SVM) algorithms construct a linear separator in a hyperplane that separates the input data into different classes such that the smallest distance between the different classes is maximized. Consider the diagram in Fig. 1 which shows the SVM algorithm in $\mathbb{R}^2$. To separate the data points (in the training set) into two distinct classes, which are demarcated by the $\bigcirc$ and $\times$ symbols, the algorithm first draws a decision boundary (dotted line). It then draws parallel margins around the decision boundary (solid lines) such that the width of the margins is maximized, and no points from either class lie within the area enclosed by the margins. SVMs thus attempts to solve an optimization problem by simultaneously minimizing the training error and maximizing the margins between the classes.

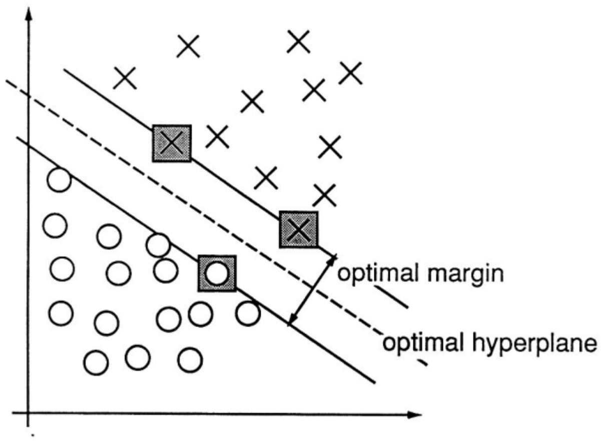

Fig. 1. A 2D representation of the SVM algorithm [5]

Specifically, let $\mathbf{X}$, $\mathbf{W}$, $\mathbf{b}$ denote the predictor vector, the weight vector, and bias respectively. Then, SVM maximizes $\frac{1}{||\mathbf{W}||}$, subject to the constraint that

$$y_i(\mathbf{x_i w_i} + \mathbf{b}) \geq 1 \quad \text{for all } 1 \leq i \leq n. \tag{6}$$

SVM algorithms perform particularly well when applied to a sparse data set with a large number of predictors. SVM algorithms are therefore highly suitable for sentiment-bearing data sets which are comprised of a large number of features (n-grams), and whose document vectors contain a small number of non-zero entries. However, SVM algorithms learn slowly and are computationally expensive.

2.2.3. *Decision Tree Classifiers*

Decision trees (refer to Fig. 2) offer another way to classify observations. Decision trees are known as regression and classification trees when operating on a quantitative and qualitative $\mathbf{Y}$, respectively. Because sentiment analysis has a qualitative $\mathbf{Y}$, we will discuss classification trees in this section.

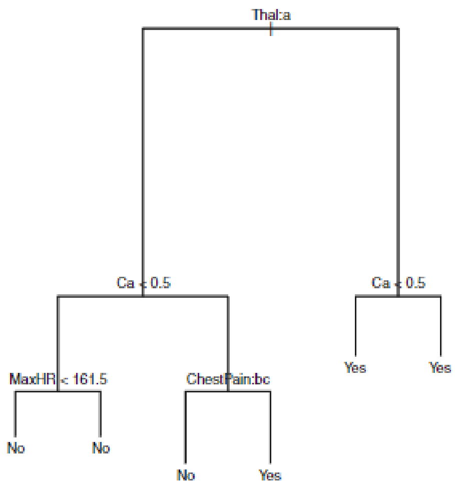

Fig. 2. A decision tree based on the qualitative predictors *Thal*, *ChestPain*, and quantitative predictors *Ca*, and *MaxHR*. This tree has six terminal nodes, three of which corresponding to *yes* prediction.

Many algorithms exist to grow classification trees, with the recursive binary splitting algorithm being the most prominent. Given a predictor $\mathbf{X}_j \in \mathbf{X}^T$ and a cutoff hyperplane $\mathbf{s} \in \mathbb{R}^p$, the recursive binary splitting algorithm defines p-dimensional regions

$$R_1(j, \mathbf{s}) = \{\mathbf{X} \,|\, \mathbf{X}_j < \mathbf{s}\} \text{ and } R_2(j, \mathbf{s}) = \{\mathbf{X} \,|\, \mathbf{X}_j \geq \mathbf{s}\}, \tag{7}$$

and seeks to find j and $\mathbf{s}$ that minimize the overall classification error rate. The classification error rate for region R_i is defined as $E_{R_i} = 1 - \max_k(\hat{p}_{R_i k})$, where $\hat{p}_{R_i k}$ is the proportion of training observations in region R_i that are from the kth class. The overall classification error rate, E, for a classification tree with p subregions, $R_i, \ldots, R_p$, is defined as the weighted average of the proportion of misclassified observations in each R_i:

$$E = \frac{1}{n} \sum_{i=1}^{p} n_{R_i} \cdot E_{R_i}. \tag{8}$$

The classification error rate, E, can be considered an analog of the residual sum of squares (RSS) that the ordinary least squares (OLS) methodology minimizes. Once $R_1(j, \mathbf{s})$ and $R_2(j, \mathbf{s})$ are obtained, the algorithm works to individually split $R_1(j, \mathbf{s})$ and $R_2(j, \mathbf{s})$, resulting in four distinct regions at the conclusion of the second iteration of the algorithm. The algorithm continues splitting the predictor space until a prespecified threshold (e.g., each R_i containing a certain minimum number of observations) is met. It then analyzes the response classes associated with the observations in each resultant R_i. It stipulates that the final response class predicted for each R_i be the same as that associated with the majority of the observations in that R_i. Said differently, if class k is the most commonly occurring response class in R_0 and $\mathbf{X}_0 \in R_0$ for some predictor vector $\mathbf{X}_0$ in the test set, then the algorithm will conclude that $\hat{f}(\mathbf{X}_0) = k$.

Although the R_i defined by the algorithm can take on any form, as long as all the resulting half-planes are mutually exclusive and exhaust the entire predictor space, equations (7) and (8) will always lead to rectangular half-planes, and doing so is standard. This short-sighted and greedy algorithm is not guaranteed to minimize the residuals in the entire predictor space, since subsequent splits in the predictor space depend on the previous splits, but it is considered sufficiently accurate for the purpose of growing decision trees.

2.2.4. Bagging

Decision trees are considered easy-to-understand machine learning tools that often suffer from low prediction accuracy (for instance, refer to [6]). Decision trees are seldom used for sentiment classification problems as an independent ensemble technique because they often give inconsistent results when applied to high-dimensional data [7]. The poor predictive performance of decision trees can be attributed to the trees having a high variance (refer to (2)). A decision tree ensemble, obtained by repeatedly applying the tree-building algorithm on various subsets of the training set, is said to have high variance if the prediction for a given $\mathbf{X}_{j,0}$ differs significantly from one tree to another. Certain techniques can be used to reduce the variance of decision trees, which, in turn, leads to more accurate tree-based predictions. Bootstrap aggregation, or bagging, is one such simple yet powerful statistical technique. As its name suggests, bagging is based on the fundamental statistical idea that the variance of the average of a set of quantities is less than the variance of the individual quantities. The bagging maneuver first obtains several bootstrapped subsets of the training set. It then constructs $D \in \mathbb{Z}^+$ deep decision trees, $f_1, \ldots, f_D$, using each subset, resulting in an ensemble of trees with low bias. It then takes a majority vote of all the predictions produced by the bootstrapped trees for $\mathbf{X}_{j,0}$ to obtain the final bagged prediction for $\mathbf{X}_{j,0}$. More precisely, for a given input $\mathbf{X}_{j,0}$, let

$$S = \{f_1(\mathbf{X}_{j,0}), \ldots, f_D(\mathbf{X}_{j,0})\}\,. \tag{9}$$

Then, for the input $\mathbf{X}_{j,0}$, the bagging procedure predicts the response $\mathbf{y}_i$ satisfying

$$\mathbf{f}_{\text{bagging}}(\mathbf{X}_{j,0}) = \underset{\mathbf{y}_i}{\operatorname{argmax}} \left\{ \sum_{s \in Y} \mathbf{1}_{\{s = \mathbf{y}_i\}} \right\}, \tag{10}$$

where $\mathbf{f}_{\text{bagging}}(\cdot)$ denotes the final prediction obtained using the bagging procedure.

2.2.5. Random Forest

If a set of predictors contains a particularly effective predictor, then a large number of decision trees constructed during the bagging procedure will be similar, because they will split the predictor space based on this predictor early on in the algorithm. This can be problematic since averaging highly correlated values leads to a smaller reduction in variance than does averaging uncorrelated values. The random forest procedure offers a remedy to this problem. During the construction of each bootstrapped tree in the random forest procedure, at each split, a fresh set of $m \in \mathbb{Z}^+$ predictors is chosen out of the p possible predictors where $m = \sqrt{p}$. The decision trees are only allowed to split on one of these m predictors, which results in an ensemble of decorrelated and dissimilar trees. Therefore, the random forest technique offers an improvement over the bagging procedure.

2.2.6. Boosting[3]

Boosting is a slow-learning ensemble method that can be applied to many statistical procedures, including decision trees. Similar to bagging, the researcher begins by drawing multiple bootstrapped training sets from the training set. However, unlike bagging, the trees in boosting are grown sequentially, meaning that the tree being grown at any given point in the algorithm takes into account the information already explained by the trees that have been grown before it. Several types of boosting procedures can be used in the classification setting, but gradient boosting is arguably the most dominant. In this section, we will discuss how gradient boosting can be applied to classification trees with a binary response variable.

Let p denote the true probability of success and $\hat{p}$ denote the estimated value of p. Note that

$$\hat{p} = \frac{\exp(\log(\widehat{\text{odds}}))}{1 + \exp(\log(\widehat{\text{odds}}))}, \tag{11}$$

where $\widehat{\text{odds}}$ are determined using the training set. The algorithm begins by assigning an initial value, $\hat{p}$, to all the observations in the training set. If $\hat{p}$ is at least as big as the prespecified threshold value (e.g., 0.5), then the model predicts $y = 1$ for all i, and $y = 0$ otherwise. Next, the algorithm computes the residuals for each observation using the formula $y_i - \hat{p}$ where $y_i \in \{0, 1\}$. Notice that each component of the vector containing the residuals will have two unique values. The residuals represent the information about the response variable that the initial assignment of $y = \hat{p}$ failed to capture. So, the next step in the process targets these residuals. The algorithm builds a classification tree to predict the residuals (instead of y) using $\mathbf{X}_j$. Usually, the recursive binary splitting algorithm is run either once or twice to construct the tree

[3]Unlike the previous sections, this section assumes that the response Y is dichotomous.

at this stage, resulting in a short tree. If the number of observations is larger than the number of terminal nodes of the tree, then at least one terminal node will contain more than one residual value. However, each terminal node of a classification tree should lead to a unique prediction value, so the algorithm computes the final prediction for each terminal node using

$$\frac{\sum_{i=1}^{n} e_i^{(1)}}{\sum_{i=1}^{n} \hat{p}^{(1)}\left(1-\hat{p}^{(1)}\right)}, \tag{12}$$

where $e_i^{(1)}$ and $\hat{p}^{(1)}$ denote the residual and predicted probability for the ith observation calculated at the first (initial) iteration of the boosting algorithm respectively. The algorithm then uses the modified tree thus created to obtain a fresh set of predicted residuals for each observation in the training set. The algorithm then multiplies these predicted residuals with the learning rate of the algorithm, a preselected parameter that governs how slowly the boosting algorithm learns, and it adds the product to $\hat{p}^{(1)}$. Since the resulting term will be of the form $\log(\text{odds})$, the algorithm uses these terms and (11) to obtain $\hat{p}^{(2)}$. The $\hat{p}^{(2)}$ values are used to obtain an updated set of residuals, which is then used to construct an updated decision tree, using the method described above. The algorithm continues until the prespecified stopping criterion, such as the residuals converging to zero, is met. Suppose the algorithm stops after $k \in \mathbb{Z}^+$ iterations. The algorithm will compute a final value for $\log(\text{odds})$ for each observation i using

$$\hat{p}^{(1)} + \lambda \sum_{j=2}^{k} e_i^{(j)}, \tag{13}$$

where λ denotes the learning rate (a tuning parameter) of the algorithm specified by the researcher. Typical values of λ include 0.01 and 0.001. The algorithm uses (11) to obtain the final predicted probability of success for each observation, which is then used to determine the final predicted value of the response variable using the same criterion we used to obtain the predicted response values using $\hat{p}$.

For some data sets, boosting has been shown to outperform random forest models [6]. However, since boosting learns slowly, the algorithm is time-intensive. It is also prone to overfitting, especially if λ or k are too large. Researchers often use cross-validation methods to select a suitable value for λ.

3. PROCEDURE

3.1. Preprocessing

3.1.1. Data Collection Procedure

We use recent tweets about health, life, and property and casualty (P&C) insurances catalysts to evaluate the performances of the SMV, decision tree classifier, bagging, random forest, and boosting algorithms. The Python code we used to scrape recent insurance-related tweets is presented in the Appendix (Section 8.1). To compile the dataset for health insurance-related tweets, our call to the Python function specified in Section 8.1 was programmed to harvest 2500 tweets generated between January 1, 2018 and March 1, 2022 containing the hashtag $\#healthinsurance$. Note that the hashtags we used in our program are case-insensitive: a function call using the parameter $\#healthinsurance$ gathers tweets containing, for instance, $\#Healthinsurance$, $\#healthInsurance$, $\#HealthInsurance$, among others. The program resulted in a dataset of 1393 tweets, suggesting that only 1393 tweets containing $\#healthinsurance$ (and its variants) were tweeted during our specified timeframe. This finding was rather surprising, as sentiments regarding health insurance tend to run high in countries lacking a publicly funded healthcare system, such as the US, although our sample of tweets was not limited to those produced by US-based users. Similar calls to the Python program in Section 8.1 with the appropriate hashtags resulted in a dataset of 1370 life insurance-related tweets. Since P&C insurance is an umbrella term used for refer to various insurance products, most popular among which are auto and homeowners insurances, we used various hashtags ($\#autoinsurance$, $\#condoinsurance$, $\#homeownersinsurance$, $\#liabilityinsurance$, $\#petinsurance$, $\#rentersinsurance$, and $\#travelersinsurance$) along with their variants to compile our dataset of P&C insurance-related tweets with 855 tweets.

3.1.2. Text Wrangling Procedure

In order to run our classification algorithm, we wrangled our scraped tweets into a usable format. Most classification software and programs require a document-term matrix as input. A document-term matrix is a matrix whose columns are spanned by the tokens or n-grams in the tweets it is analyzing, and whose rows are spanned by the individual pieces of textual data (here: tweets) under study. The matrix also contains the term frequency-inverse document frequency for each token of each tweet which signifies the importance of the token in the corpus of documents. Thus, it is easy to see that the number of observations (tweets) can quickly exceed the number of columns (tokens), making textual analysis a very high-dimensional classification problem. This can be problematic since, in the absence of appropriate textual wrangling procedures aimed at reducing the "noise" in the dataset, applying classification algorithms to high-dimensional data can force them to model the "noise" in the dataset, which would undesirably inflate the variance of each algorithm (refer to (2)). Along with dimension reduction, preprocessing also aims to obtain tokens that can be easily mapped to the n-grams in sentiment lexicons.

We illustrate the working of our preprocessing Python program (see Section 8.2) using the following hypothetical tweet:

> Thank you, #Anthem, for quickly processing the claim I filed for my son's antidepressants :)!
>
> #healthinsurance #BCBS #mentalhealth

The algorithm specified in Section 8.2 first turns any uppercase characters into lowercase, and then removes punctuation and emoticons. Next, it removes the *stop words*, very frequently occurring n-grams (usually, 1-grams), it believes bear little sentiment-related information. Some examples of stop words in our hypothetical tweet include "the", "for", and "my". Finally, the algorithm transforms the words into their root form; for instance, the words "quickly" and "processing" will be transformed into their respective root forms "quick" and "process". This important process allows software used to assign an initial sentiment score to textual data (discussed below) to more easily map the n-grams in the textual data under study to those in the sentiment lexicon. There are two competing procedures in the world of natural language processing that can be used to transform words into their root form: lemmatization and stemming. Stemming separates what it believes is the *stem* of a certain word from some any commonly used prefixes and suffixes the word may contain and retains the stem as the output. Occasionally, stemming can lead to nonsensical resultants; for instance, stemming will transform "studying" to "study" by removing the suffix "ing", but it will transform "studies" to "studi" by removing the suffix "es". Lemmatization, on the other hand, outputs the *lemma* of a word. It takes the context surrounding the word into consideration during the transformation process. For example, most lemmatization algorithms transform "studies" to "study" because are able to recognize that the former is the third person, singular, present tense of the latter. Since meaningless stems are unlikely to be recognized by the sentiment lexicons we used to assign an initial sentiment score to the tweets (discussed in Section 3.2), the algorithm in Section 8.2 uses lemmatization. The algorithm will convert our hypothetical tweet to the following text, possibly with minor variations, to be passed on to the next step in the process:

> thank you quick process claim antidepressant healthinsurance mentalhealth

These preprocessing steps winnow sentiment-bearing information from the noise and convert the raw tweets into a format that will lead the machine learning algorithms to perform optimally.

3.2. Processing

We used the software SentiStrength [8], which is considered by [9] to be "the most popular stand-alone sentiment analysis tool," to provide an initial sentiment score to our preprocessed tweets. SentiStrength has been tested, critiqued, and utilized by many researchers in the field of sentiment analysis; for a non-exhaustive list, see [8]. Research from psychology indicates that humans process positive and negative emotions in parallel [10]. Hence, the sentiments in sentiment-bearing human writing can rarely be classified as solely positive or negative. SentiStrength utilizes this insight to distinguish itself from other sentiment classifying tools. To be explicit, SentiStrength extracts the preprocessed words in the tweets, utilizes sentiment lexicons to assign each word a positive and negative sentiment score, and then assigns a final positive and negative sentiment score to each tweet based on how positive, neutral, or negative it deemed the individual words of the tweet to be. The positive (negative) ratings range from 0 to 5 (-5), where 0 signifies a neutral tweet and 5 (-5) signifies an extremely positive (negative) tweet.

Afterward, we used the R package RTextTools to apply the machine learning algorithms specified in Section 2.2 on our preprocessed and newly labeled tweets. We chose RTextTools over other R packages with similar capabilities because it is a "one-stop shop for conducting supervised learning with textual data" [11]. RTextTools converted textual data into a document-term matrix with the user-specified level of sparsity using a single command, foregoing the need to use additional software to tokenize the data. It then randomly bifurcated the datasets into training and test sets, trained each ML algorithm on the document-term matrix associated with the training set, and finally applied the trained algorithm on the document-term matrix associated with the test set to allow the user to evaluate and compare the performances of the ML algorithms.

4. RESULTS

We contrast the ML algorithms specified in Section 2.2 on three grounds: accuracy, performance, and time efficiency. We used mean 5-fold cross-validation scores to quantify how accurately the algorithms classify the observations in the test set. Additionally, we utilized precision, recall, and F-scores (also known as F1 scores) to quantify the performance of the algorithms. Runtime analysis helped us in measuring the time efficiency of the algorithms. Running each of our ML algorothms on three different datasets (one for each of life, health, and P&C insurance) allowed us to reach more comprehensive conclusions about the algorithms on all three grounds. We omit the Maximum Entropy algorithm from our analysis as it is no longer supported by RTextTools. The R program we utilized for the evaluation process is presented in Section 8.3 in the Appendix.

4.1. Accuracy Analysis

To perform 5-fold cross validation on a given dataset, the algorithm randomly divides the dataset into five non-overlapping and exhaustive subsets (also known as "folds") of comparable sizes. For the first iteration, the algorithm trains on four subsets and uses the remaining subset as the test set. It uses this remaining subset to compute the number of misclassified observations. The process repeats four more times, allowing each subset to act as the test set exactly once. The total number of misclassified observations obtained from all the five iterations of the algorithm are then added and divided by the total number of observations to obtain the final mean estimate of the 5-fold cross validation error rate. The R program presented in Section 8.3 finds the average number of correctly classified observations across the 5 folds for each dataset and algorithm under study.

The mean 5-fold cross validation measurements for each of the five algorithm under consideration are presented in Tables 1 and 2, and visualized in Figure 3. Note that the mean 5-fold cross validation scores range from 0.490 to 0.743 when considering positive sentiment classifications. This suggests that approximately 49% to 74% of the observations were correctly classified by the algorithms across the three datasets. For all algorithms, except boosting, the worst performance was on the dataset with life insurance tweets. On average, boosting is able to classify the observations with an initial positive sentiment classification most accurately, followed by the decision tree classifier, bagging, random forest, and finally SVM.

Algorithm	Mean Accuracy		
	Health	Life	P&C
SVM	0.654	0.490	0.661
Bagging	0.657	0.491	0.662
Boosting	0.666	0.708	0.743
Random Forest	0.659	0.484	0.665
Decision Tree Classifier	0.657	0.489	0.669

Table 1. Mean Accuracy Measurements for Positive Sentiment Classification with 5-fold CV

Algorithm	Mean Accuracy		
	Health	Life	P&C
SVM	0.668	0.583	0.658
Bagging	0.668	0.582	0.657
Boosting	0.701	0.801	0.698
Random Forest	0.669	0.584	0.660
Decision Tree Classifier	0.664	0.584	0.660

Table 2. Mean Accuracy Measurements for Negative Sentiment Classification with 5-fold CV

The mean 5-fold cross validation scores range from 0.582 to 0.801 when considering negative sentiment classifications, suggesting that approximately 58% to 80% of the observations were correctly classified by the algorithms across the three datasets in this setting. Notice that, overall, the algorithms were able to classify the observations with negative sentiment classification more accurately than those with positive sentiment classifications. Once again, all algorithms, except boosting, perform worse on the dataset with life insurance tweets. Moreover, on average, boosting is again able to classify the observations most accurately. The remaining methods are ranked as follows in terms of accuracy: random forest, SVM, decision tree classifier, and bagging. Therefore, boosting appears as a clear winner in this category.

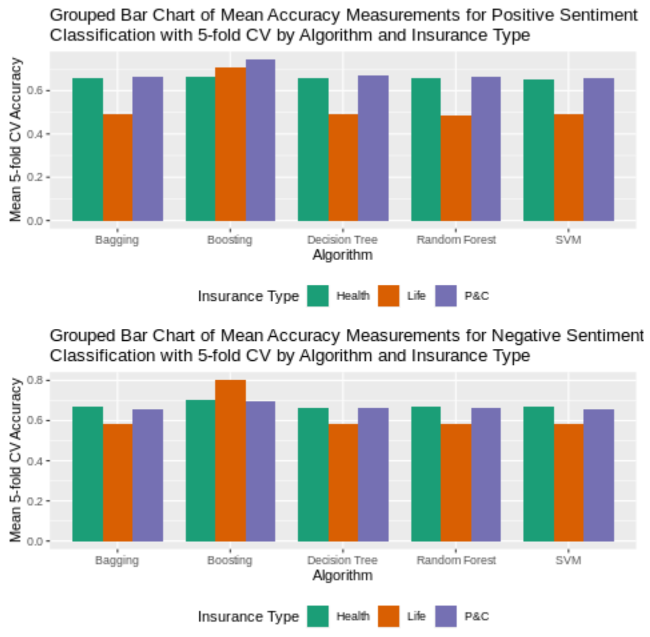

Fig. 3. Grouped Bar Chart of Mean Accuracy Measurements with 5-fold CV by Algorithm and Insurance Type

4.2. Performance Analysis

	True $Y = 1$	True $Y \neq 1$
Algorithm $Y = 1$	5	11
Algorithm $Y \neq 1$	7	670

Table 3. An Illustration of a Confusion Matrix for $Y = 1$ in a Multi-Class Classification Setting

	True $Y = 2$	True $Y \neq 2$
Algorithm $Y = 2$	60	44
Algorithm $Y \neq 2$	55	212

Table 4. An Illustration of a Confusion Matrix for $Y = 2$ in a Multi-Class Classification Setting

	True $Y = 3$	True $Y \neq 3$
Algorithm $Y = 3$	20	33
Algorithm $Y \neq 3$	51	83

Table 5. An Illustration of a Confusion Matrix for $Y = 3$ in a Multi-Class Classification Setting

	True $Y = 4$	True $Y \neq 4$
Algorithm $Y = 4$	60	55
Algorithm $Y \neq 4$	77	111

Table 6. An Illustration of a Confusion Matrix for $Y = 4$ in a Multi-Class Classification Setting

The R package RTextTools allows the researcher to evaluate the performance of the ML algorithms by contrasting their precision, recall, and F-scores, all of which are standard accuracy metrics in the field of text (opinion) mining. Precision and recall metrics originated as methods used to evaluate the accuracy of algorithms using confusion matrices. For a particular class and a classification algorithm, the algorithm decides whether an observation gets classified in the class. Since we are dealing with supervised data, we can, in turn, benchmark this classification against the gold standard of human classification. (SentiStrength has been proven to possess human level accuracy for sentiment classifications [8].)

Precision measurements answer the question of how much a researcher can trust an algorithm when it indicates that an observation belongs to a certain class [12]. Naturally, the precision measurement for an algorithm is defined as the ratio of the number of observations the algorithm correctly assigns to a given class to the total number of observations the algorithm assigned to that class, irrespective of whether the classification was correct. Note that the number of observations an algorithm assigns to a particular class is not necessarily the same as the true size of that class. RTextTools adopts a one-versus-rest approach when constructing *confusion matrices* (behind the scene) to compute precision (and recall) for a multi-class classification problem, like the one that is the subject of our study [13]. To illustrate, assume momentarily that the response variable Y denotes the positive rating of the tweets. Recall that $Y \in \{1, 2, 3, 4\}$, depending on the extent to which the algorithms deem the tweets to be positive. Assume the confusion matrices for each of the four classes in Tables 3-6. The elements of Tables 3-6 are added component-wise to obtain a final pooled confusion matrix. We present the pooled confusion matrix for this example in Table 7. The precision measurement for this example is given by $145/(145 + 143) \approx 0.503$. Notice that a high precision value suggests that the number of observations being *incorrectly* classified into a given class is low [12].

The concept of the recall metric is deceivingly similar to that of the precision metric. Recall measurements answer the

	True Yes	True No
Algorithm Yes	145	143
Algorithm No	190	1076

Table 7. An Illustration of a Pooled Confusion Matrix in a Multi-Class Classification Setting

question of how much a researcher can trust a given classifier to find all the members of a given class [12]. Mathematically, recall measurements are defined as the ratio of the number of observations the algorithm correctly assigns to a given class to the true size of that class. (True class size is determined by SentiStrength.) To illustrate, note that the recall measurement corresponding to Table 7 is $145/(145 + 190) \approx 0.433$. A high recall value suggests that the number of observations belonging to a given class, but being classified into a different class, is low [12].

Precision and recall measurements are governed by a trade-off: higher precision values correspond to lower recall values, and vice versa. Whether a researcher prefers higher precision values (at the cost of lower recall values) or higher recall values (at the cost of lower precision values) depends on the goal of the research. For instance, a researcher attempting to predict the likelihood of a certain medical condition would prefer the algorithms that result in higher recall and lower precision measurements. In this case, the researcher's algorithms would raise a significant number of false alarms, but would not miss anyone who is likely to (truly) have the condition, which is the more desirable case in this scenario. Contrastingly, social media algorithms predicting the content a user is likely to engage with based on the user's past activity tend to favor higher precision values at the expense of lower recall values. Here, the social media algorithm is likely to present to the user a large number of content pieces similar to those the user engaged with in the past along with a small number of content pieces the user has not yet engaged with. The assumption is that by presenting the content the user is known to have an interest in, the social media website will likely increase the amount of time the user spends surfing the website.

Statisticians and data scientists broadly define F-scores as a metric balancing precision and recall measurements. Although the descriptions of F-scores can vary from one scholar to the other, in this paper, we define F-scores as the weighted average of precision and recall measurements:

$$\text{F-score} = \frac{\left(\beta^2 + 1\right) \times \text{precision} \times \text{recall}}{\beta^2 \times (\text{precision} + \text{recall})}, \qquad (14)$$

where $\beta \in \mathbb{R}$ is the tuning parameter determined by the researcher [14]. The F-score is evenly balanced when $\beta = 1$ [14]. It favours precision when $\beta > 1$, and recall otherwise [14]. The specification in (14) is consistent with how this metric is computed by the package RTextTools [11].

We present the precision, recall, and F-scores we obtained for each of our three datasets in Table 8 in this section and Table 11 in Section 8.4 in the Appendix. These results are visualized in Figures 6 and 7 in Section 8.4 in the Appendix and Figure 4 in this section. Notice that the precision measurements range from 16% to 19.4% for positive sentiment classifications, whereas they range from 15.3% to 33.6% for negative sentiment classifications. Additionally, the recall measurements range from 20% to 25% for both positive and negative sentiment classifications. Although these precision and recall measurements do not fall in the desirable range of 60% to 85%, each of the recall measurements is comparatively higher than the corresponding precision measurement for their respective sentiment classifications. This implies that the algorithms have set a low threshold for classifying the observations into the target response classes. Said differently, all five algorithms are likely to be able to locate the cases of a given class, however a sizeable number of observations classified into a given response class are incorrectly classified. The F-scores for positive sentiment classifications range from 12.8% to 20%, whereas those for negative sentiment classifications range from 16.2% to 19.5%. In general, F-scores range from 0% to 100% and higher F-Scores indicate that precision and recall measurements are highly dissimilar from one another. Researchers generally prefer higher F-scores because this indicates that the algorithm has produced a large number of false positives (negatives) and a small number of false negatives (positives). This might be favorable, depending on the goal of the research, as described previously. Since our recall and corresponding precision measurements are highly similar to each other, our F-scores are low. This implies that the number of false positives and false negatives produced by all our algorithms are similar.

4.3. Runtime Analysis

Certain complex ML algorithms entail high processing times, making them challenging to implement on larger datasets. The runtimes for each of our ML algorithms are presented in Tables 9-10, and are visualized in Figure 5.

For all three datasets, the SVM algorithm has the highest runtime, followed by bagging, boosting, random forest, and decision tree classifiers, irrespective of the initial response classification. All algorithms take under one-and-a-half minutes to run on our datasets (of relatively small size). Bagging, which is theoretically assumed to offer an improvement over decision tree classifiers, did not lead to significantly better CV, precision, recall, and F-scores when applied to the datasets at hand. Moreover, the runtime of bagging (the algorithm with the highest runtime) was 1.25 to 1.42 times that of the decision tree algorithm (the algorithm with the lowest runtime) across all datasets and sentiment classifications. Therefore, since relatively more complex methods such as bagging and SVM were found to have slightly higher runtimes than the simpler methods, the researcher might not prefer to utilize them, depending on the objective of the study.

Algorithm	Precision Measurements Health	Life	P&C
SVM	0.160	0.194	0.168
Bagging	0.160	0.162	0.168
Boosting	0.160	0.194	0.168
Random Forest	0.160	0.094	0.168
Decision Tree Classifier	0.160	0.094	0.168
	Recall Measurements		
SVM	0.250	0.204	0.250
Bagging	0.250	0.202	0.250
Boosting	0.250	0.204	0.250
Random Forest	0.250	0.200	0.250
Decision Tree Classifier	0.250	0.200	0.250
	F-Scores		
SVM	0.195	0.148	0.200
Bagging	0.195	0.180	0.200
Boosting	0.195	0.148	0.200
Random Forest	0.195	0.128	0.200
Decision Tree Classifier	0.195	0.128	0.200

Table 8. Precision, Recall, and F-scores for Positive Sentiment Classification

Algorithm	Runtime (in seconds) Health	Life	P&C
SVM	0.817	0.854	0.739
Bagging	0.724	0.749	0.654
Boosting	0.671	0.666	0.637
Random Forest	0.660	0.653	0.592
Decision Tree Classifier	0.613	0.600	0.578

Table 9. Runtimes for Positive Sentiment Classification

5. CONCLUSION

The goal of this study was to compare the predictive powers of the SVM, bagging, boosting, random forests, and decision tree algorithms using datasets containing tweets related to health, life, and P&C insurances as catalysts. We utilized SentiStrength, a software reportedly capable of classifying textual data with human-level accuracy, to determine each tweet's level of positivity and negativity. We then used the machine learning algorithms to repeat the classification process and benchmarked their performances against that of SentiStrength. Our results revealed that boosting was able to classify the tweets in their (true) sentiment categories most accu-

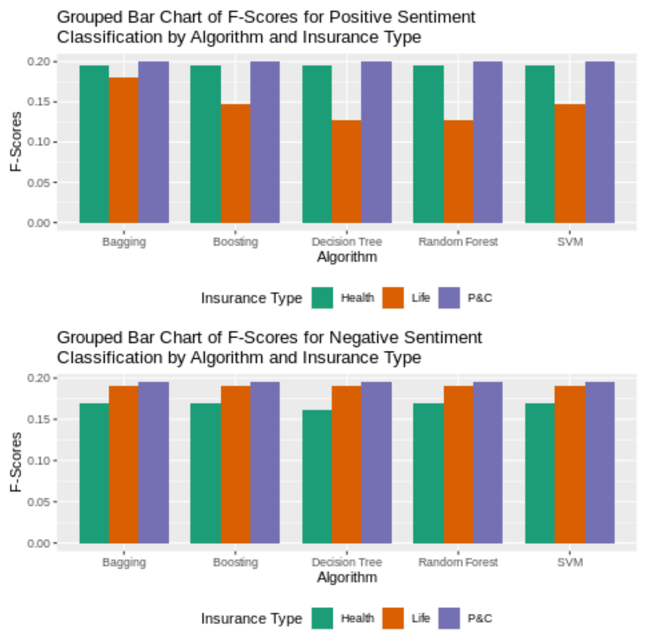

Fig. 4. Grouped Bar Chart of F-Scores for Negative Sentiment Classification by Algorithm and Insurance Type

Algorithm	Runtime (in seconds)		
	Health	Life	P&C
SVM	0.799	0.859	0.705
Bagging	0.733	0.738	0.661
Boosting	0.692	0.695	0.627
Random Forest	0.666	0.653	0.591
Decision Tree Classifier	0.614	0.601	0.562

Table 10. Runtimes for Negative Sentiment Classification

rately. Boosting, however, entailed a slightly higher runtime than random forest and decision tree algorithms. The SVM and bagging algorithms possessed the highest runtimes of all the algorithms we considered; however, they did not yield any significant improvements in performance or accuracy for this cost. Therefore, the algorithm a researcher should favor in analyzing sentiment-bearing, insurance-related textual data depends on the goal of their research. Algorithms such as SVM, bagging, and boosting are shown to possess slightly higher accuracy and predictive powers at the expense of higher runtimes. Conversely, algorithms such as random forests and decision tree classifiers are comparatively more time efficient, but display slightly lower accuracy and predictive powers.

One way of furthering this research is to repeat the study after gathering more textual data. Non-textual data, such as auditory and visual sentiment-bearing data, can also be gathered and analyzed if the researcher is willing to employ additional software capable of preprocessing non-textual data. Finally, several state-of-the-art, hybrid approaches (for instance, refer to [3]) have emerged in the budding field of sentiment analysis that claim to be able to more accurately predict sentiments. Interested researchers can compare the performances of these techniques on insurance-related data with the performances of the algorithms discussed in this paper. This might allow analysts and researchers, such as those affiliated with companies capable of making life-saving decisions, to discover algorithms that better address the needs of their insureds.

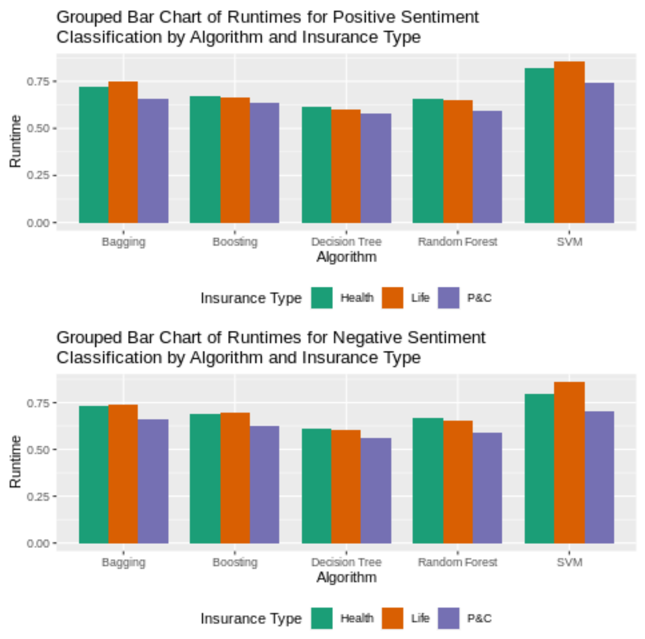

Fig. 5. Grouped Bar Chart of Runtimes by Algorithm and Insurance Type

6. ACKNOWLEDGEMENTS

I am indebted to the unwavering support and invaluable feedback that Dr. Julie M. Clark provided throughout this project. I would also like to express my sincere gratitude to Dr. Giarcarlo Schrementi for providing expert guidance with the Python-intensive aspects of the thesis.

7. REFERENCES

[1] Editorial Team, "Girl dies due to cigna's decision not to pay for liver transplant," Dec 2007.

[2] Margaret M. Bradley and Peter J. Lang, "Instruction manual and affective ratings - university of vermont," 1999.

[3] Matt Taddy, "Measuring political sentiment on twitter: Factor optimal design for multinomial inverse regression," *Technometrics*, vol. 55, no. 4, pp. 415–425, Nov 2013.

[4] Edward Loper, "The maximum entropy classifier," May 2002.

[5] Corinna Cortes and Vladimir Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[6] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani, *An Introduction to Statistical Learning: with Applications in R*, Springer, 2013.

[7] Sida Wang and Christopher D. Manning, "Baselines and bigrams: Simple, good sentiment and topic classification," 2018.

[8] Zhun Hung, "Sentistrength," Dec 2009.

[9] Mike Thelwall, Kevan Buckley, Georgios Paltoglou, Di Cai, and Arvid Kappas, "Sentiment strength detection in short informal text," *Journal of the American Society for Information Science and Technology*, vol. 61, no. 12, pp. 2544–2558, 2010.

[10] Raul Berrios, Peter Totterdell, and Stephen Kellett, "Eliciting mixed emotions: A meta-analysis comparing models, types, and measures," *Frontiers in Psychology*, vol. 6, 2015.

[11] P. Jurka, Timothy, Loren Collingwood, E. Boydstun, Amber, Emiliano Grossman, and van Atteveldt, Wouter, "Rtexttools: A supervised learning package for text classification," *The R Journal*, vol. 5, no. 1, pp. 6, 2013.

[12] Juan Orozco Villalobos, "Precision vs recall," Jan 2020.

[13] Mehul Gupta, "Calculating precision & recall for multi-class classification," Apr 2020.

[14] Marina Sokolova, Nathalie Japkowicz, and Stan Szpakowicz, "Beyond accuracy, f-score and roc: A family of discriminant measures for performance evaluation," *Lecture Notes in Computer Science*, p. 1015–1021, 2006.

[15] Monika Kabir, Mir Md. Kabir, Shuxiang Xu, and Bodrunnessa Badhon, "An empirical research on sentiment analysis using machine learning approaches," *International Journal of Computers and Applications*, p. 1–9, 2019.

[16] Dan Jurafsky and James H. Martin, *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*, Pearson Prentice Hall, 2009.

[17] Udacity, "Support vector machine - georgia tech - youtube," Feb 2013.

[18] Zixuan Zhang, "Support vector machine explained," Aug 2019.

[19] Aratrika Pal, "Gradient boosting trees for classification: A beginner's guide," 2022.

[20] Kamil Slowikowski, "Remove all traces of emoji from a text file.," 2018.

[21] Alessia D'Andrea, Fernando Ferri, Patrizia Grifoni, and Tiziana Guzzo, "Approaches, tools and applications for sentiment analysis implementation," *International Journal of Computer Applications*, vol. 125, no. 3, pp. 26–33, 2015.

[22] Griffin Leow, "Scraping tweets with tweepy python," Oct 2020.

[23] Muriel Kosaka, "Cleaning & preprocessing text data for sentiment analysis," Nov 2020.

8. APPENDIX

8.1. Wed Scraping Code

Presented below is the Python program that was used to scrape the tweets that were used in this study. The code was adapted from [22].

```
def scraptweets(search_words, date_since,
numTweets, numRuns):

    # Define a for-loop to generate tweets
    # at regular intervals

    # Define a pandas dataframe to store the date:
    db_tweets = pd.DataFrame(columns = [
    'username', 'acctdesc', 'location',
    'following','followers', 'totaltweets',
    'usercreatedts', 'tweetcreatedts',
    'retweetcount', 'text', 'hashtags']
                                    )
    program_start = time.time()
    for i in range(0, numRuns):

        # We will time how long it takes to
        # scrape tweets for each run:
        start_run = time.time()

        # Collect tweets using the Cursor object
        # Each item in the iterator has various
        # attributes that you can access to get
        # information about each tweet
        tweets = tweepy.Cursor(api.search,
        q=search_words, lang="en",
        since=date_since,
        tweet_mode='extended').items(numTweets)

        # Store these tweets into a python list
        tweet_list = [tweet for tweet in tweets]
        print(len(tweet_list))

        # Begin scraping the tweets individually:
        noTweets = 0
        for tweet in tweet_list:
        # Pull the values
            username = tweet.user.screen_name
            acctdesc = tweet.user.description
            location = tweet.user.location
            following = tweet.user.friends_count
            followers = tweet.user.followers_count
            totaltweets =
            tweet.user.statuses_count
            usercreatedts = tweet.user.created_at
            tweetcreatedts = tweet.created_at
            retweetcount = tweet.retweet_count
            hashtags = tweet.entities['hashtags']
            try:
                text =
                tweet.retweeted_status.full_text
            except AttributeError:
            # Not a Retweet
                text = tweet.full_text

            # Add the 11 variables to the
            # empty list - ith_tweet:
            ith_tweet = [username, acctdesc,
            location, following, followers,
            totaltweets, usercreatedts,
            tweetcreatedts, retweetcount,
            text, hashtags]

            # Append to dataframe - db_tweets
            db_tweets.loc[len(db_tweets)] =
            ith_tweet

            # increase counter - noTweets
            noTweets += 1

        # Run ended:
        end_run = time.time()
        duration_run =
        round((end_run-start_run)/60, 2)

        print('no. of tweets scraped for run {}
        is {}'.format(i + 1, noTweets))
        print('time take for {} run to complete
        is {} mins'.format(i+1, duration_run))

        time.sleep(920) #15 minute sleep time

    # Once all runs have completed, save them to
    a single csv file:
    from datetime import datetime

    # Obtain timestamp in a readable format
    to_csv_timestamp =
    datetime.today().strftime('%Y%m%d_%H%M%S')

    # Define working path and filename
    path = os.getcwd()
    filename = path + to_csv_timestamp
    + 'tweets.csv'

    # Store dataframe in csv with creation
    # date timestamp
    db_tweets.to_csv(filename, index = False)

    program_end = time.time()
    print('Scraping has completed!')
    print('Total time taken to scrap is
    {} minutes.'.format(round(program_end -
    program_start)/60, 2))

    # Calling the function
    scraptweets('#insertyourhashtag
    -filter:retweets', 'insert_date_since',
    insert_numTweets, insert_numRuns)
```

8.2. Preprocessing Code

Presented below is the Python program that was used to preprocess the raw P&C insurance-related tweets. A similar procedure was used to preprocess raw life insurance-related and health insurance-related tweets. The code was adapted from [23]

```
    nlp = spacy.load('en',
                disable=['parser', 'ner'])

    # lowercase all reviews
    PandC_df['new_reviews'] =
    PandC_df['text'].apply(
        lambda x: " ".join(x.lower() for
        x in x.split())
    )

# remove punctuation
PandC_df['new_reviews']
= PandC_df['new_reviews'].str.replace(
'[^\w\s]','')

# remove emojis
def remove_emoji(text):
    emoji_pattern = re.compile("["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"
        # symbols and pictographs
        u"\U0001F680-\U0001F6FF"
        # transport and map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags
        u"\U00002702-\U000027B0"
        u"\U000024C2-\U0001F251"
                    "]+", flags=re.UNICODE)
return emoji_pattern.sub(r'', text)

PandC_df['new_reviews'] =
PandC_df['new_reviews'].apply(
    lambda x: remove_emoji(x)
    )

# removing stop words
stop = stopwords.words('english')
PandC_df['new_reviews'] =
PandC_df['new_reviews'].apply(
    lambda x: " ".join(x for
    x in x.split() if x not in stop)
    )

# Lemmatization removes the grammar tense and
# transforms each word into its original form.

def space(comment):
    doc = nlp(comment)
    return " ".join([token.lemma_
        for token in doc])
PandC_df['new_reviews'] =
PandC_df['new_reviews'].apply(space)

# display the processed dataset
PandC_df.head()
```

8.3. Classification Code

Presented below is the R program that was used to classify and evaluate health insurance-related tweets based on their positive sentiment rating. A similar procedure was used to classify and evaluate health insurance-related tweets based on their negative sentiment rating as well as life insurance-related and PC insurance-related tweets based on their respective positive and negative sentiment ratings. The code was adapted from [11].

```
library(RTextTools)
# Load the dataset
data_health <- read.csv(file_path,
                header=TRUE)

# Create a document-term matrix
doc_matrix <- create_matrix(
                data_health$new_reviews,
                language="english",
                removeNumbers=FALSE,
                stemWords=FALSE,
                removeSparseTerms=.5)

# Create a container
container <- create_container(doc_matrix,
                data_health$Positive,
                trainSize=1:1000,
                testSize=1001:nrow(
                data_health),virgin=FALSE)

# Train the models
SVM <- train_model(container,"SVM")
BOOSTING <- train_model(container,"BOOSTING")
BAGGING <- train_model(container,"BAGGING")
RF <- train_model(container,"RF")
TREE <- train_model(container,"TREE")

# Classify the models
SVM_CLASSIFY <- classify_model(container, SVM)
BOOSTING_CLASSIFY <- classify_model(container,
                        BOOSTING)
RF_CLASSIFY <- classify_model(container,
                            RF)
BAGGING_CLASSIFY <- classify_model(container,
                            BAGGING)
TREE_CLASSIFY <- classify_model(container,
                            TREE)

# Generate precision, recall, and F-scores
analytics <- create_analytics(container,
            cbind(SVM_CLASSIFY,
            BOOSTING_CLASSIFY,
            BAGGING_CLASSIFY,
            RF_CLASSIFY,
            TREE_CLASSIFY))
summary(analytics)

# Cross Validation
SVM <- cross_validate(container, 5, "SVM")
# A similar approach was used for
# the other algorithms

# Runtime Analysis for SVM
start.time <- Sys.time()
SVM <- train_model(container,"SVM")
SVM_CLASSIFY <- classify_model(container, SVM)
analytics <- create_analytics(container,
                cbind(SVM_CLASSIFY))
end.time <- Sys.time()
end.time - start.time
```

8.4. Additional Accuracy Analysis Results

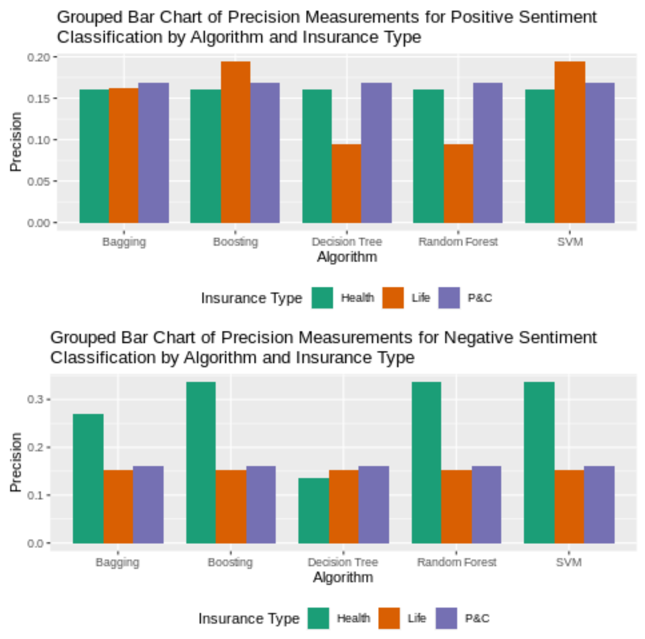

Fig. 6. Grouped Bar Chart of Precision Measurements by Algorithm and Insurance Type

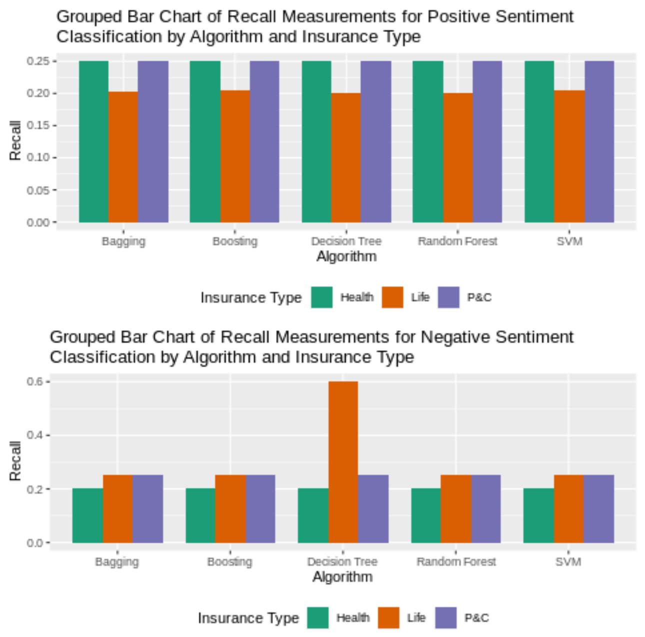

Fig. 7. Grouped Bar Chart of Recall Measurements by Algorithm and Insurance Type

	Precision Measurements		
Algorithm	Health	Life	P&C
SVM	0.336	0.153	0.160
Bagging	0.270	0.153	0.160
Boosting	0.336	0.153	0.160
Random Forest	0.336	0.153	0.160
Decision Tree Classifier	0.136	0.153	0.160
	Recall Measurements		
SVM	0.204	0.250	0.250
Bagging	0.204	0.250	0.250
Boosting	0.204	0.250	0.250
Random Forest	0.204	0.250	0.250
Decision Tree Classifier	0.200	0.250	0.250
	F-Scores		
SVM	0.170	0.190	0.195
Bagging	0.170	0.190	0.195
Boosting	0.170	0.190	0.195
Random Forest	0.170	0.190	0.195
Decision Tree Classifier	0.162	0.190	0.195

Table 11. Precision, Recall, and F-scores for Negative Sentiment Classification