Final Project: Text Analytics

Author

Willem van der Schans

University

David Eccles School of Business

University of Utah

Date

30th of July, 2020

Table of Contents

1. Introduction

1.1 Project Topic

The purpose of this project is to take a deep dive into text classification with a focus on comparing classification algorithms, trough performance metrics, and outcomes. Text classification is a form of predictive and descriptive analytics that allows for a wide variety of applications. Examples of classification applications are News filtering, Document Organization, Opinion Mining, and spam filtering (Aggarwal & ChengXiang, 2012).

The interest in the topic of algorithm performance and comparison comes from a personal interest regarding predictive analytics. The predictive usage of algorithms is fascinating due to the field being inherently complex, originating from a large number of algorithms available (Kowsari, 2019). Comparing algorithms in the proposed method allows for a deep understanding of the application of the algorithms covered. The knowledge generated in this project regarding the algorithms will allow for more insight when deciding on what algorithm to use for future applications regarding text classification. From a previous project, which will be partly discussed in section 1.4, a great amount of understanding of the differences between several numeric classification models was gained. The goal of this project is to repeat part of the process followed in the aforenoted project but with a focus on text analytics (van der Schans, 2020). Being confident in both numeric and text analytics applications allows for a broad machine learning toolset with the knowledge to perform (predictive) analytics regardless of the provided data type.

This project focuses on review classification and will be employed to gain a deeper understanding of different algorithms. The potential focus originates from the datasets that have been found that fit the goal of the project. Within the project, the preference is to take the focus off of data cleaning and focus on algorithm comparison; section 1.2.1 will go over the proposed datasets and the current preference for what dataset to use.

1.2 Data

The data set that has been decided on to be used is an IMBD dataset extracted from Kaggle. (Lakshmipathi, 2019) The IMBD data set includes reviews and a sentiment each review brings with it. Initially, in the project proposal we considered two other datasets; the Reuters-21578 Text Categorization Collection (UC Irvine, n.d.) and a Clickbait data set (Chakraborty, Paranjape, Kakarla, & Ganguly, 2016). The Reuters data set consists of 21 .smg files. The clickbait data set consists of two text files of strings, of which one file consists of titles that are spam, and one file with articles titles that are not spam. A cleaned data set has been found of the Reuters data set (ishaan007, 2017) that allows for a greater focus on classification algorithms over data cleaning. The difference between the clickbait data set and the Reuters data sets is the target of the dataset.

Both datasets mentioned earlier, have very similar text to analyze, namely headlines of news articles. The target of the clickbait dataset is to determine if a headline is clickbait, and the target of the Reuters dataset is headline classification into subjects or topics. The reason we used neither of these data sets, however, is due to the text they contain being just headlines. Generally, headlines are shorter than reviews. When comparing the IMDB and clickbait data set, the average word count per document, respectively, is 231.15 and 105.28, a difference of 220%. With the goal of this paper being to compare algorithms, enlarging shortcomings is important. Finding shortcomings requires us to push algorithms to their limits, and one way of doing that is by using a larger amount of text per document. The difference in word count per document is the reason the IMDB dataset has been chosen to replace the previously considered datasets.

The IMDB data set comes in a .csv with the correct structure for classification with a dimension of 50,000 by 2. One of the columns in the IMDB data set contains the reviews or the documents, and the other column contains the target variable expressed in text: positive or negative. As previously mentioned, the average word length of the documents in the IMDB data set is 231.15. The frequency of target classes is 50%, with 25,000 documents existing within both classes.

1.3 Algorithm decision parameters

Aggarwall and ChengXiang divide text classification algorithms into six sections; Decision trees, Pattern (rule)-based classifiers, SVM classifiers, Neural Network Classifiers, Bayesian (Generative) Classifiers, and other classifiers. Other classifiers "include nearest neighbor classifiers, and genetic algorithm-based classifiers" (Aggarwal & ChengXiang, 2012). Six algorithms will be chosen to cover all six of the sections, as described by Aggarwall and ChengXiang. We choose algorithms on the premise to create a mix of algorithms covered in course material and algorithms not covered in the course material. As of now, SVM (Chatterjee, Support Vector Machine, 2020), naive Bayes (Chatterjee, Naive Bayes, 2020), and decision trees (Chatterjee, Decision Trees and Random Forests, 2020) are models covered within course material that will be explored and compared in this project. Covering three sections within the classifiers mentioned above, which leaves room for three more sections. The other chosen classifiers are chosen based on the criteria that they are well known, are different in the way they operate, and are included in the Caret package in R (Kuhn, Available models, 2019). The other included algorithms outside of those covered in the course material are KNN, NNet, and Rpart.

1.4 Research Objective and Structure

The goal of this paper is to try to build a framework of tools that can be used in the complex world of algorithms. Previously completed work includes a project with a similar goal for numeric classification with a binary target (van der Schans, 2020). In this project, five initial models were compared, as shown in appendix A and figure 1, after the initial comparison grid searching was applied to the three highest performing models, which further increased scores after which a final comparison was made, as shown in appendix B and figure 2.

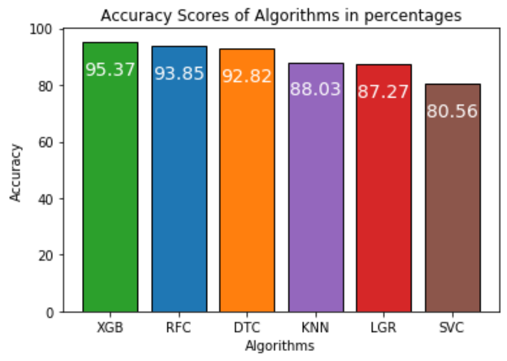

Figure 2: Initial Project score finan6500 project.

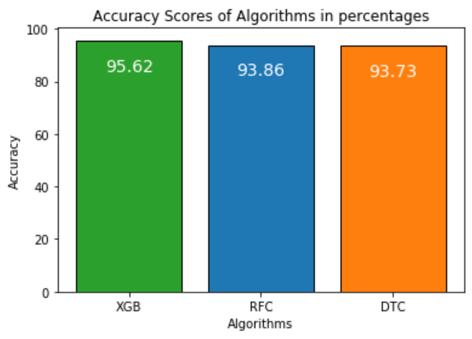

Figure 1: Final Algorithm scores after grid searching

This project sets out to follow a similar process with a larger focus on how the models work and how they differ in their application and performance. The focus will manifest in a theoretical survey on the methodic differences between algorithms and a post-results discussion based on the model performance. Proposed main questions that will be answered are:

1. How does the algorithm work?
2. How does the algorithm perform against other included algorithms?
3. Why does the algorithm perform the way it does, and how can model performance be improved?

Covering all questions allows the reader to place the algorithms in a mental framework and use them as tools in future text classification applications.

2. Theoretical Framework

2.1 Text analytics and classification

Text analytics is defined as "the automated process of translating large volumes of unstructured text into quantitative data to uncover insights, trends, and patterns" (Pascual, 2019). Analytics as a whole can be subdivided into three types, predictive analytics, prescriptive analytics, and descriptive analytics. All three types have different goals, as can be observed in the names of the analytics types, respectively; to predict, to prescribe, and to describe (Sharda, Asamoah, & Ponna, 2013). Text classification can fall under both predictive and descriptive analytics based on what the goal of the application is, however, in this paper, the focus lies on predictive text classification.

Predictive text classification falls under supervised learning. Supervised learning entails that the outcome is known while training the algorithm, allowing the algorithm to learn the mapping function from the input dataset (train data) to the output target variable. After this, the model can be applied to unknown data (test data) to predict the unknown target variable for the newly introduced data based on the mapping function the algorithm constructed from the training data set. (Brownlee, Supervised and unsupervised machine learning algorithms, 2019).

The process of predictive text classification follows the same basic principles but can vary widely based on the data set the process is applied to. The process that is described by Halibas et al. is a good example of the process that most predictive text classification will follow in one way or another—starting with manual data cleaning, after which pre-processing and feature extraction is conducted, this includes tokenization and n-gram generation. When feature extraction is completed, the dataset is split into a test and train data set to be able to generate in-sample and out-of-sample performance measures. After splitting the data set, the model is applied, and the accuracy measures are generated and evaluated. (Halibas, Shaffi, & Mohamed, 2018).

2.2 Text classification Algorithms

As can be made up of section 2.1, the algorithm used in predictive text analytics is one of the most important steps in the process. With the algorithm creating the mapping function, the outcome is solely dependent on this step. Once the algorithm is picked, the role of the data analyst in this process is to provide the most optimally cleaned data and set the most optimal hyperparameters, which dependent on the algorithm has a limited effect on the outcome. It is, therefore, important to pick the right algorithm for the job to get the most optimal results. In this section, the workings of the algorithms that will be included in this paper will be discussed. The section will follow the classification of different text classification algorithms of Aggarwall and ChengXiang (Aggarwal & ChengXiang, 2012).

2.2.1 Decision trees

In technical terms, a decision tree is a hierarchical decomposition of the data space, in which a predicate or a condition on the attribute value is used in order to divide the data space hierarchically (Aggarwal & ChengXiang, 2012). What this means is that data is broken up into subsets that get smaller on every split. A decision tree can split based on entropy and in its extension information gain. Entropy is a measure of uncertainty in a data set. If a binominal classification data set exists with a 50/50 split of the two possible outcomes, the entropy is the highest it can be; 1. The information gain measures the decrease in entropy before and after a split in the decision tree. The way a decision tree algorithm employs this is (Sharma, 2020):

1. Calculate entropy of the target in the data set
2. Split the dataset on different features and calculate the entropy of each new branch.
3. Calculate the information gain for each feature split and retain the feature split with the highest information gain.

There are more ways for a decision tree to split, for example, on variance reduction, Gini impurity, and Chi-square. In this project, the parRF model (a randomForest wrapper allowing parallel processing) is used, and the tree is split on the Gini Impurity since this is the packages default behavior (Liaw, Package 'randomForest', 2018). The Gini impurity is 1-Gini, in which "Gini is the probability of correctly labeling a randomly chosen element if it was randomly labeled according to the distribution of labels in the node." (Sharma, 2020). The lower the Gini impurity is, the higher the purity of a node. The way a decision tree uses this to split is by calculating the Gini impurity of each split, after which it retains the split with the lowest value of the Gini impurity. Gini is preferred to using information gain due to no logarithms being used in calculating the Gini impurity, which makes it computationally less demanding (Sharma, 2020). It should be noted that decision trees are prone to overfitting in practice due to the subsamples created in the tree becoming extremely small and specific at deep nodes. (Liberman, 2017)

Instead of using a regular decision tree algorithm, this project uses a random forest algorithm. We opted to use Random forest due to rule-based classifiers and decision trees having an overlap in their workings netting close to the same results when using Rpart and Part, for example. Random forest builds on decision trees to combat overfitting. In short random forest generates a large number of decision trees on randomly selected rows and features, after which the results are averaged. Each tree that is created chooses its final class, and the class that gets predicted the most is the winner and the final predicted class. Due to random forest essentially being an ensemble model, overfitting is reduced, but the computation power needed for random forest compared to decision trees is significantly larger. (Liaw & Wiener, Classification and Regression by randomforest, 2002).

2.2.2 Bayesian (Generative) Classifiers

A Bayesian classifier is a classifier that uses the Bayes theorem shown in figure 3. The Bayes theorem incorporates marginal probabilities; P(A) and P(B) and conditional probabilities; P(A | B) and P(B | A). The Bayes theorem gives an alternate way to find conditional probability with that if one conditional probability is hard to calculate the reverse conditional probability, if available or easy to calculate, can be used to calculate the target conditional probability (Brownlee, A Gentle Introduction to Bayes Theorem for Machine Learning, 2019).

$$P(A|B) = P(B|A) * P(A)/P(B)$$

Figure 3: Bayes Theorem

The application of a Bayesian classifier in this paper is by using a Naïve Bayes classifier. Naïve Bayes works on the premise that the predictors are independent of each other. In other words, the algorithms assume that all features independently contribute to the target outcome (Sunil, 6 Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R, 2017). The Naïve Bayes algorithm works by calculating the probability that each of the features of a data point exists in all possible target classes. It then selects the class for which the probability is the highest (Missinglink.ai, n.d.)

A naïve Bayes classifier is fast and also predicts very fast being computationally non-demanding. The assumption of feature independence is important for naïve Bayes, and if it holds, it will often outperform other models; however, in real-life data sets, features are hardly ever completely independent of each other, which violates the assumption. A pro for the implementation of Naïve Bayes in this specific project is that naïve Bayes performance well for categorical variables given the train categories are exhaustive. For numerical variables, however, naïve Bayes makes the strong assumption of a normal distribution, which can lower performance if not satisfied by the data. (Sunil, 6 Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R, 2017)

2.2.3 SVM classifiers

SVM stands for support vector machine and is generally used in classification. An SVM generates several hyperplanes in a multidimensional space, after which the hyperplane that optimally separates classes and minimizes classification errors is retained. In this process, support vectors are data points that are the closes to the hyperplane. Support vectors are effectively the cut-off point for a specific class. These support vector points are used to calculate a margin between both classes to be able to better define and calculate the resulting hyperplane. While some classes can be separated in a two-dimensional setting, other classes cannot, and therefore SVM utilized the kernel trick. Kernel trick means that if a problem cannot be solved in x-dimensions, then the algorithm will try again in x+1 dimensions until the problem can be solved (Aggarwal & ChengXiang, 2012). There are several kernels available for SVM, of which the most popular are the linear kernel, the polynomial kernel, and the radial basis function kernel. By using "svmlinear" in caret in this paper, we will be utilizing the linear kernel with the function shown in figure 4 (Navlina, 2019).

$$K(x, xi) = sum(x * xi)$$

Figure 4: Linear Kernel

SVM is memory efficient compared to other algorithms because the only decision points are a subset of the data as in the support vectors. SVM is effective when a clear margin of separation is found, and in high-dimensionality in data sets exists, if there is no clear margin of separation, SVM will have issues providing good performance due to no hyperplane being able to separate classes accordingly. Using SVM with larger data sets influences the training time to be extremely large, making SVM a less suitable algorithm for applications that involve data sets that are larger. (Sunil, Understanding Support Vector Machine(SVM) algorithm from examples (along with code), 2017)

2.2.4 Pattern (rule)-based classifiers

Pattern rule-based classifiers work by creating a set of rules to classify data. Decision trees are generally related to rule-based classifiers; however, since Aggarwall and ChengXiang separated them in their classification of algorithms, they will be reviewed individually. The way rules generated by a rule-based classifier are setup is that on the left-hand side of the rule, a condition is present, and on the right-hand side, the rule contains the class label. Generally, generated rules are based on the occurrence of words in documents as opposed to the absence of words in documents since occurrence is more informative then absence in most cases (Aggarwal &

ChengXiang, 2012). A rule can look like this: “happy → positive.” Where the occurrence of the word happy in a document would make the algorithm classify the document in the positive class. Rules are mainly generated based on two criteria: support and confidence. Support is the number of documents that are relevant to the rule in the training data set. If a rule of which both the left- and right-hand side are relevant to 20,000 documents, as opposed to 20 documents for another rule, the first rule will be chosen over the second one due to the first rule having higher support. The confidence of the rule is the second criterium, which shows the strength of the rule and quantifies how confident the algorithm is that the right-hand side of the rule is satisfied if the left-hand side of the rule is satisfied.

In this project, we will be using “PART,” which is a rule-based classifier from the Rweka package (Hornik, 2020). The way PART works is based on the method described by Frank and Witten in which no global optimization is needed to come to accurate results making the algorithms simpler. The way the algorithm works is that it creates a decision tree for every rule to be made; the leaf with the largest support in this tree is then made into a rule, and the tree is discarded (Eibe & Witten, 1998).

The advantages of using rule-based classifiers are that they are easy to interpret and visually generate. Rule-based models can often be plotted or shown in a tree-like structure, and the logic of such a model aligns with the way humans think. Cons for using rule-based classifiers start with them completely neglecting regression, which causes loss of information in transforming a linear problem to a classification problem. Rule-based classifiers also can be memory and computationally intensive based on the input data, increasing training time, and the hardware requirements needed to run the model (Molnar, 2020).

2.2.5 Neural Network Classifiers

Neural networks are a set of algorithms that are loosely based on a brain as they consist of neurons that fire given a certain amount of simulation. Neural networks are designed to recognize patterns in data of a numerical nature contained in vectors. The reliance on numerical data necessitates that all data fed to a neural network has to be converted to numerical data before a neural network can process the data. (Nicholson, 2019)

What would be a neuron in a brain is called a node in a neural network algorithm. A node consists of the data input and weights. The weights determine if a certain input should be amplified or dampened to get to the correct results. After an input and its weight has been computationally combined, an activation function within the node will evaluate it. The activation function determines if this input is allowed to continue further into the algorithm affecting the outcome. If the activation function does not fire, the input will be dropped due to it not being a good enough predictor, and the input will then not affect the outcome. Nodes are contained within layers, in which a layer is a row of nodes. A neural network has at least an input layer, a hidden layer, and an output layer. The hidden layer is where the previously described process finds place. In deep learning, a neural network has more than one hidden layer allowing for more complex analysis to be realized. (Nicholson, 2019)

Neural networks work well with non-linear data, which is the reason they are being used, for example, to do image recognition. As long as data can be converted to numeric data, a neural network can be used for both classification and regression, making it a widely applicable algorithm. The control that the data analysts have over neural networks is also great; any amount of inputs and hidden layers can be set for the algorithm to work with. The main limitation for neural networks, however, is that they are computationally very expensive to run, meaning it is often a quicker and cheaper choice to run another algorithm dependent on the dataset and the size of the data set. Training data is very important for neural network algorithms, and over-fitting and generalization are known cons of neural networks. (Ciaburro & Venkateswaran, 2017)

2.2.6 Other classifiers (KNN)

KNN stands for k-nearest neighbors, which is an algorithm that assumes that similar data points being neighbors, in other words being close together, implies similarity. KNN tries to find groups of data points that are close together to try to predict the target variable based on these groupings (Harrison, 2019). KNN is a lazy algorithm in that the generated function is initially only approximated on the train data, and a large chunk of the actual computing happens when testing data is provided to the algorithm. When testing data is provided, KNN calculates the distance between each data point fed to the algorithm and its nearest K-nearest neighbors. These testing data points are also known as the query points. In this instance, K is a hyperparameter that can be defined by the data analyst. Whatever class most of the K-nearest neighbors of a query point belong to the query point will also be assigned to. The effect K has is large with a small K resulting in overfitting and a large K resulting in underfitting (Schott, 2019).

A large benefit of KNN is that it makes no assumptions about the data. Making no assumptions means that there is nothing to be violated compared to, for example, naïve Bayes with assumes independent probabilities. KNN is relatively fast initially but gets progressively slower based on the size of the dataset. KNN is also versatile and able to do both regression and classification. The importance of the K value, however, can drastically influence the outcome of the algorithm, meaning often grid searching is needed to come close to optimal results.

Additionally, KNN does not work well with high dimensional data since distances become less representative the higher the number of dimensions is. (Schott, 2019)

3. Method

Since the focus of this paper lies in the exploration of different classification algorithms, the coding methodology has been kept simple and straight forward. The following section described the process and decisions that we made while coding the project.

3.1 Dataset and Data preparation

As discussed in section 1.2 of this paper, the data set used in this project is an IMDB review dataset from Kaggle. The dataset consists of two columns: one column with movie reviews and one column with the sentiment of the review being positive or negative. Upon loading the data with "fread," we converted the binominal target variable from a character variable to a numeric factor variable with values; negative=0 and positive=1.

With 50,000 documents of data and an average word count of 229.37 words per document, the dataset held around 11.5 million words. Initially, we aimed to use the full dataset in this project, but due to hardware limitations, this would not be possible. Initial dimension reduction meant removing 75% of the dataset resulting in 12,500 rows being retained. Row selection was performed trough createdatapartition to retain the class division of the target variable in the resulting dataset at 50/50.

After the initial dimension reduction, we split the dataset up with an 80/20 split in a train and test data set respectively to be able to conduct a hold-out evaluation and produce in-sample and out-of-sample performance metrics for all six algorithms. We tokenized both the test and train data sets by words. Tokenization involved removing numbers, punctuation, symbols, and hyphens and also converting the tokens to lowercase. We then removed stopwords from the tokenized data sets and conducted stemming afterward to reduce dimensionality further. After stemming, we converted the tokenized datasets to document term matrices weighted by term frequency-inverse document frequency or tf-idf, when then trimmed the tf-idfs to further reduce dimensionality with a minimum term frequency of 10 and minimum term-document frequency of two.

To further reduce dimensionality and being able to run the algorithms with the hardware limitations imposed by the workstation used, we employed latent semantic analysis. We constructed train and test singular value decompositions (SVD) from the three matrices of which the LSA output existed for both the original test and train tf-idfs. Both the resulting SVD's had 302 columns and 10,000 and 2,500 rows respectively for train and test as expected.

3.2 Machine Learning

For machine learning, we used the caret package. Caret is a wrapper for over 200 different machine learning models, and by using this package, we streamlined the whole process mainly on the two following points.

Firstly, the syntax of caret is always identical regardless of what model we ran. The generalized syntax means that all models can run with the same settings allowing for better comparison between the algorithms. Using caret also means that the resulting R object of each model follows the same structure; this allows for generalization of one code chunk in which just the references have to be changed accordingly for each algorithm, which greatly reduces coding time.

Secondly, caret incorporates grid searching by default (search = grid), giving all algorithms an equal chance to estimate their optimal hyperparameters as not to penalize algorithms that rely on hyperparameter tuning. Control over the hyperparameters and function of an algorithm is seen as an asset as it allows versatility and a broader application. The maximum tune length was set to 7 to save on runtime. Tune length determines the level of detail present in the grid used for grid searching, the higher the tune length integer is, the more detailed the grid search will. Having a larger tune length parameter will exponentially increase runtime; therefore, we decided on a value that provided a runtime that was significant but manageable (1,336 seconds) for all algorithms except random forest. For random forest, we conducted a manual grid search due to the algorithm taking an excessive amount of time to run and the step size in the automatic grid searching being nonsensical. A rule of thumb for a random forest algorithm is that a good starting point for determining the number of trees to run in the model should be close to equal to the square root of the amount of column in the data set. With 302 columns in our SVD, however, that meant a large number of trees and, therefore, a very long run time. The final grid search consisted of c(27,35,79) for mtry, all three of which are optimal values in our numerous iterations of the grid. While the time is shorter with an effective tune length of 3 over 7, we chose not just to include the optimal mtry value to simulate the actual runtime of the algorithm and make it comparable to other algorithms.

The main settings used in caret other then the ones mentioned above had to do with cross-validation of 5 folds, three times for most algorithms. For random forest, however, we used three folds two times. Again, to reduce random forest run time. To make run times comparable, we multiplied the runtime of random forest with 5.83 to make up for the difference between the 18 models ran and the 105 models that were supposed to run. The model used metric= “accuracy,” maximize = TRUE to make sure the caret selected the optimal hyperparameters in the grid search based on the highest accuracy score. In this case, accuracy is a good measure since we have a balanced class of samples so that mistakes between classes have an even weight. Class-specific measures are not suitable as a guiding metric here since both classifications are important.

We will not just evaluate accuracy in this paper, but we created a metrics list to be used with the mmetric function of Rminer; metric_list <- c("ACC", "TPR", "PRECISION", "F1"). While accuracy is very powerful within the dataset used in this project due to the 50/50 class divide, other measures give class-specific info, which will give us more insight into how the final accuracy score came to be. A good explanation of the meaning of two of the other included metrics namely; the true positive rate and precision, is given by Rawat and states; "While recall (TPR) expresses the ability to find all relevant instances in a dataset, precision expresses the proportion of the data points our model says was relevant actually were relevant." (Rawat, 2019). In the end, accuracy does not tell the whole story of model performance, and foregoing other metrics is forgoing possible insight.

We collected the final mmetric outputs in a data frame, which we used to collect the final in-sample and out-of-sample performance scores. For further visualization, we plotted the collected data in a grouped bar chart (figure 5), showing both the in-sample and out-of-sample accuracy net to each other. This visualization allows for easy comparison between models, but also an internal comparison between a models' in-sample and out-of-sample accuracy is made easier this way.

4. Discussion

This section will discuss the algorithm comparison outcome based on figure 5, and Appendix C. Figure 5, as previously discussed, consists of a visualization of the in-sample and out-of-sample accuracy for each algorithm included in this project. To reiterate: the six exact caret models that we ran in order of occurrence in figure 5 are; "knn", "naive_bayes", "nnet", "parRF", "rpart", and "svmLinear".

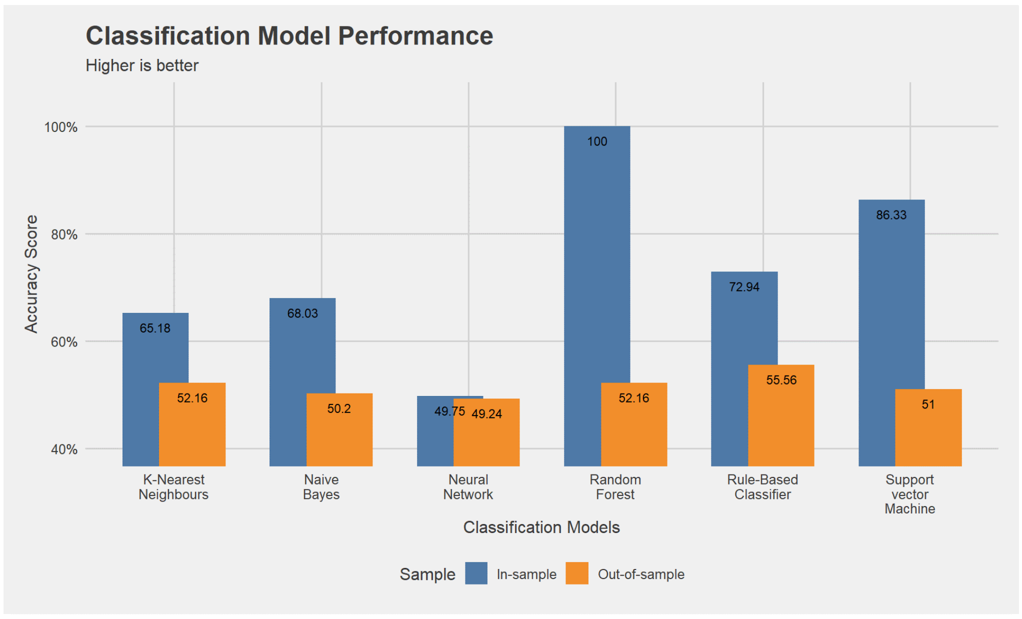

Figure 5: Algorithm Accuracy Scores

Figure 5 shows that for almost all models, there is a clear degree of overfitting present and the final accuracy scores are less than stellar. Using a majority predictor would have given us a 50% accuracy score on both of our classes, this 50% accuracy is the benchmark for the models to beat. While most of our algorithms beat the 50% score and, therefore, the benchmark, the difference is minimal, and our neural network underperformed compared to the benchmark. The main reason that comes to mind for poor performance is the amount of training data used to train our models. Having to remove 75% of the dataset due to hardware limitations means throwing away a significant amount of information. For text analytics, which is inherently more complex in its meaning, then numeric data, ten thousand rows simply are not enough to properly train a text classification algorithm. As we can see, the models are overfitting, meaning that the training sample likely contains much noise and lacks a degree of possible generalization. Both suspicions mentioned earlier further reinforce the notion that the training dataset is not a large enough sample to represent the population. If our training set does not represent the population, we cannot possibly expect our algorithms to perform well on a new data set since mapping functions cannot be generalized properly.

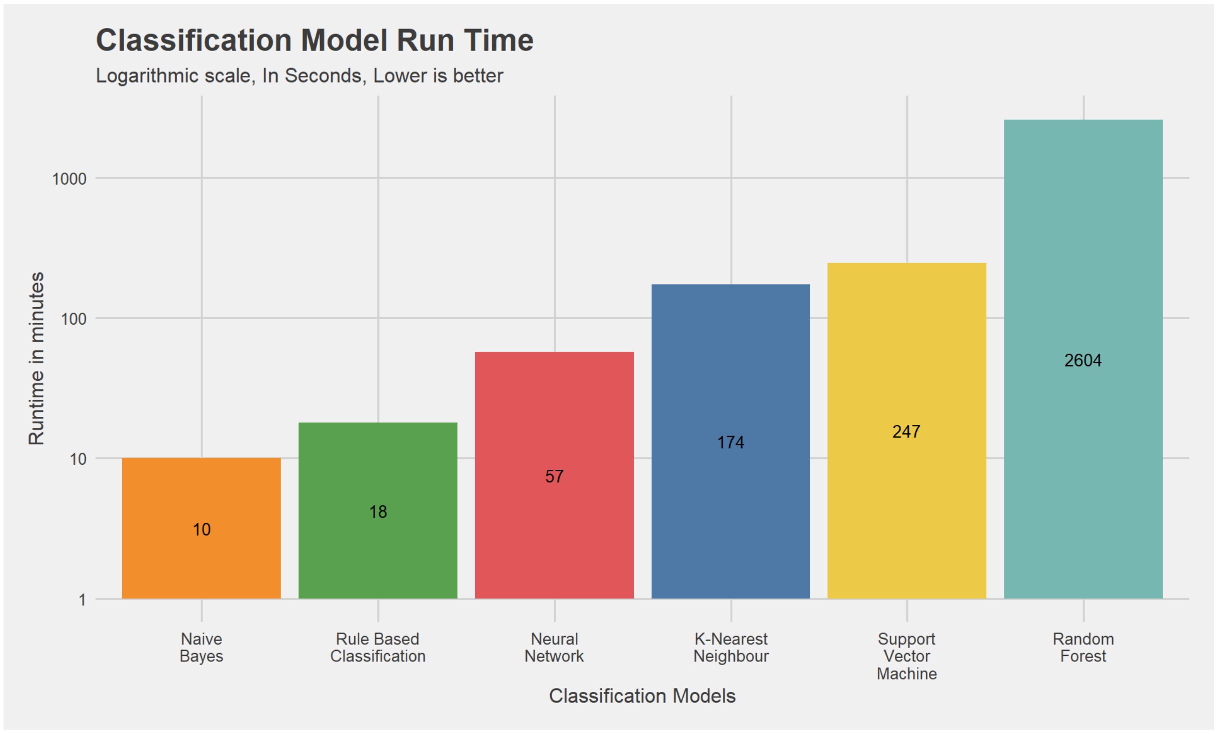

Figure 6: Algorithm Run Time in seconds

Even though all models are performing poorly, we would like to discuss each different model output and highlight possible reasons for poor performance and points of improvement to be applied when building onto this paper. Note that All performance metrics can be found in appendix C, and all model runtimes can be found in Appendix D.

4.1 K-Nearest Neighbours

KNN is slightly overfitted with an in-sample accuracy score of 65.18% and an out-of-sample accuracy score of 52.16%. The out of sample accuracy score only being 2.16% higher than the majority class benchmark is a poor result. While this can be partly explained by the size of the training data set, the relative high dimensionality within the data most likely is also a reason. When looking at appendix C, the true positive rates for both classes hover around 50 within a range of 10 both ways; it can, therefore, be derived that KNN never found a good point of separation between the two classes since it makes a large number of mistakes in both classes equally. As stated in section 2.2.6, dimensionality reduces the meaning of the distances for the class divide. The negative effect of increased dimensionality is also referred to as the “curse of dimensionality,” likely affecting our use case. The run time of KNN taking 162 seconds is quite high, and it further demonstrates that the dimensionality of the data set probably has been too for KNN.

4.2 Naïve Bayes

Naïve Bayes is performing about as well as KNN with an in-sample accuracy score of 68.03% and an out-of-sample accuracy score of 50.20%. What is interesting about the performance metrics of Naïve Bayes is that the negative recall rate is low at 19.12%, while the positive recall rate is relatively high at 81.28%. When looking at the precision, however, both classes score 50% within the out-of-sample score. What can be derived is that naïve Bayes is heavily predicting the positive class which is the reason its accuracy is around 50%, the same as the class division of the dataset. The true positive rate for the negative class label is 19.12% meaning the algorithm did not correctly identify 81% of the instances in the negative class label and predicted positive when the algorithm should have predicted negative. Since the algorithm hardly predicted negative, however, its precision can be high since when it did choose negative, it was accurate 50.53% of the time. What is interesting is that this behavior only occurred in the test data set; in the training dataset, naïve Bayes predicted both classes more frequently with its accuracy being 18% higher in-sample then out-of-sample.

The suspected reason Naïve Bayes runs this poorly is that its assumption is being violated. Features having independent probabilities seems to directly combat language where words are used together continuously to convey meaning based on context and sentence structure. It should be noted that in the exploratory stage of this project we ran naïve Bayes on a regular document term matrix with the textmodel_nb {quanteda.textmodels} algorithm and we found an accuracy score of around 80%. The main difference between the dfm in the exploratory phase and this phase was that all 50,000 rows were contained in the original dfm providing naïve Bayes more training data after an 80/20 split.

To summarize, the findings suggest that naïve Bayes is performing poorly due to both its assumption being violated and a lack of training data being fed to the algorithm. In the future, both ideas should be further analyzed. Additional research should also be conducted to find the difference between the caret naïve_bayes and the textmodel_nb algorithms as it can also be the case that the naïve_bayes model used is not optimized for text which would contradict the 80% dfm accuracy, however, so this thought seems unlikely to be true. Naïve Bayes did run very fast, which cements it as a potential option when using live data streams that have to be predicted or when quick benchmarks have to be set.

4.3 Neural Network

The neural network performed under the benchmark set by the majority class predictor with an accuracy below 50%. The output shows that the Neural network, even more so then naïve Bayes, heavily predicted the positive class. With the neural network having a recall rate in the negative class of only 2%, it means that it correctly identified only 25 out of the 1,250 negative class labels. When applying the same math to the positive labels, it can be derived that the neural network predicted the positive class 97.24% of the time. The neural network did this both in the in-sample and out-of-sample data sets, and it can, therefore, be derived that it found no pattern to predict on and reverted to a majority classifying like state. What is interesting is that, if the algorithm had reverted to a majority classifying state completely, it would have provided better results, than it currently did.

The expected reason the neural network performed so poorly is due to a lack of training data. Neural networks can be very powerful tools, but to be trained, they need a vast amount of training data. Ten thousand training rows does not seem enough for the algorithm to find patterns and create a mapping function that is generalizable to a dataset. For future use cases, the neural network should only be considered for datasets with a higher amount of training entries. Further study should be conducted as to where the threshold lies to get a better understanding as to what an appropriate training data set size for neural networks is. Surprisingly the runtime of the neural network was average, most likely due to it not being able to find patterns and quickly moving trough iterations because of this.

4.4 Random Forest

We had the most issues running random forest in this project due to its extremely long runtime. We had to change the grid searching parameters to lower the number of models from 105 to 18 models, after which it still took seven and a half minutes to run the random forest algorithm with parallel processing over five cores. While grid searching, it became apparent that the model was overfitting with all in-sample measures being at 100%, meaning that everything is predicted correctly in every case. The out-of-sample accuracy score, however, is quite significantly lower, with 52.16%. The lower score is interesting since random forest should be combatting overfitting compared to our rule-based classifier or decision tree model, and in this case, it does not. In the case of the test data set, random forest predicted a negative class label more often than a positive class label as opposed to the three models previously discussed with 1,596 negative label predictions over 904 positive label predictions.

Like with KNN, it seems that the data fed to the random forest algorithm is too high-dimensional for it to perform properly and within reasonable runtime. Three hundred two columns seem too much to handle for random forest in this application, and it seems to have issues finding decision points that are meaningful in the provided data set. Caret contains other algorithms that work on random forest premises but work better with high dimensional data. Due to the data being high dimensional, the expectation we had was that many trees would be used to create a large number of decision points to classify the data appropriately, but the grid searching outcome was mtry 27 which is relatively low for the number of features fed to the random forest.

The computationally intensive nature of this model made it extremely hard to work with given the current application of this project. Given the runtime, the results are lackluster. Smaller datasets that hold more concentrated information seem more appropriate for random forest, clearly defining its use case.

4.5 Rule-Based Classifier

The rule-based classifier we ran in this project got the best out-of-sample score out of all included algorithms. The model does seem to overfit quite a bit with an accuracy score difference between the in-sample and out-of-sample score of around 17%. The model predicts a negative class label more than a positive class label with a respective 1,741 over 759 prediction rate. The model was especially good at identifying negative cases when they occur with a recall rate of 75.2%. The higher recall rate can most likely be attributed to stronger negative word usage in reviews with a negative sentiment, which was easier to identify for the algorithm then the positive word or the absence of these negative words in positive reviews explaining the 40% difference in recall rate. The rule-based classifier ran quickly with a runtime of only 18 seconds.

4.6 Support Vector Machine

While KNN and support vector machines are different in their workings, one principle both of the algorithms rely on is a clear separation of classes within the data set. As we discussed for the KNN model, this clear separation of data most likely not present in this data set, meaning that the support vector machine model is not likely to perform well. The final out-of-sample accuracy score of 51% more or less confirms the previously stated expectation, even with the relatively high dimensionality of the data set. The support vector machine predicted both classes close to equally with 1,419 over 1,081 in favour of the negative class label. With both recall rates around 50%, this algorithm is balanced in its approach but wrong approximately half of the time.

The runtime of the model was quite long, with 247 seconds. As stated before, this data set most likely did not suit SVM well in terms of class separation, resulting in the poor results even with the high-dimensionality present.

4.7 Future research

Due to the poor performance of all included algorithms, it can be concluded that the data set used is not optimal for classification. The poor performance is likely based on the training dataset being too small for the algorithms to train on. In addition to the dataset being small, the data also seemed non-informational regardless of its size. Lastly, the data set seemed to lack any form of a significant class divide creating issues for at least 4 out of 6 classifiers.

For future studies, it would be interesting to see the same method applied to several different text classification data sets to study if results improve, or if the algorithms that are included in this study are just not optimal for text classification. Building upon this project in the future would involve following the same process with multiple larger data sets, more runtime, a higher grid searching tune length, and better hardware.

This project focused on the workings of the algorithms itself; however, it would be interesting to explore what dataset attributes suit the included algorithms the best. Information on which datasets attributes provide good performance for each included model would further enhance the framework created in this paper and allows for a clearer direction on the use cases of the covered algorithms.

5. References

Aggarwal, C. C., & ChengXiang, Z. (2012). A survey of text classification algorithms. *Mining text data*, 163-222.

Brownlee, J. (2019, 12 4). *A Gentle Introduction to Bayes Theorem for Machine Learning.* From Machine Learning Mastry: https://machinelearningmastery.com/bayes-theorem-for-machine-learning/

Brownlee, J. (2019, 8 12). *Supervised and unsupervised machine learning algorithms.* From machinelearningmastery: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

Chakraborty, A., Paranjape, B., Kakarla, S., & Ganguly, N. (2016). Stop clickbait: Detecting and preventing clickbaits in online news media. *ieee/acm international conference on advances in social networks analysis and mining (asonam)* (pp. 9-16). IEEE.

Chatterjee, P. (2020). Decision Trees and Random Forests. In P. Chatterjee, *Text Classification.*

Chatterjee, P. (2020). Naive Bayes. In *Text Classification* .

Chatterjee, P. (2020). Support Vector Machine. In P. Chatterjee, *Text Classification.*

Ciaburro, G., & Venkateswaran, B. (2017). *Neural Networks with R.* From https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781788397872/1/ch01lvl1sec27/pros-and-cons-of-neural-networks

Eibe, F., & Witten, I. H. (1998). Generating accurate rule sets without global optimization.

Halibas, A. S., Shaffi, A. S., & Mohamed, M. A. (2018). Application of Text Classification and Clustering of Twitter Data for Business Analytics. *Majan International Conference* (pp. 1-7). Oman: Faculty of Computing Sciences.

Harrison, O. (2019, 9 10). *Machine Learning Basics with the K-Nearest Neighbors Algorithm.* From Towards Data Science: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761#:~:text=KNN%20works%20by%20finding%20the,in%20the%20case%20of%20regression).

Hornik, K. (2020, 2 2). *Package 'RWeka'.* From CRAN: https://cran.r-project.org/web/packages/RWeka/RWeka.pdf

ishaan007. (2017, March 19). *vector space modelling.* From Github: https://github.com/ishaan007/vector_space_modelling/tree/master/Code/Data

Kowsari, K. (2019, May 21). *Text Classification Algorithms: A Survey.* From medium: https://medium.com/text-classification-algorithms/text-classification-algorithms-a-survey-a215b7ab7e2d

Kuhn, M. (2019, 03 07). *Available models*. From The caret Package: https://topepo.github.io/caret/available-models.html

Kuhn, M. (2020, 3 20). *Package 'caret'*. From Cran: https://cran.r-project.org/web/packages/caret/caret.pdf

Lakshmipathi, N. (2019, 03 08). *Imdb dataset of 50k movie reviews*. From Kaggle: https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

Liaw, A. (2018, 03 22). *Package 'randomForest'*. From rpart: https://cran.r-project.org/web/packages/randomForest/randomForest.pdf

Liaw, A., & Wiener, M. (2002). Classification and Regression by randomforest. *R news*, 18-22. From Citizen Net: https://blog.citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics

Liberman, N. (2017, 1 26). *Decision Trees and Random Forests*. From Towards Data Science: https://towardsdatascience.com/decision-trees-and-random-forests-df0c3123f991#:~:text=Decision%20trees%20are%20prone%20to,could%20lead%20to%20unsound%20conclusions.

Missinglink.ai. (n.d.). *Classification with Neural Networks: Is it the Right Choice?* From Missinglink.ai: https://missinglink.ai/guides/neural-network-concepts/classification-neural-networks-neural-network-right-choice/

Molnar, C. (2020, 07 27). *Interpretable ml book*. From Interpretable Machine Learning: https://christophm.github.io/interpretable-ml-book/rules.html

Navlina, A. (2019, 12 27). *svm classification scikit learn python*. From DataCamp: https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python#svm

Nicholson, C. (2019). *A Beginner's Guide to Neural Networks and Deep Learning*. From pathmind: https://pathmind.com/wiki/neural-network

Pascual, F. (2019, 11 20). *What is text analytics*. From Monkeylearn: https://monkeylearn.com/blog/what-is-text-analytics/

Rawat, S. (2019, 10 2). *Is accuracy EVERYTHING?* From Towards Data Science: https://towardsdatascience.com/is-accuracy-everything-96da9afd540d

Schott, M. (2019, 4 22). *K-Nearest Neighbors (KNN) Algorithm for Machine Learning*. From Medium: https://medium.com/capital-one-tech/k-nearest-neighbors-knn-algorithm-for-machine-learning-e883219c8f26

Sharda, R., Asamoah, D., & Ponna, N. (2013). Business analytics: Research and teaching perspectives. *Proceedings of the International Conference on Information Technology Interfaces, ITI*, (pp. 19-27). doi:10.2498/iti.2013.0589

Sharma, A. (2020, 06 30). *4 Simple Ways to Split a Decision Tree in Machine Learning*. From Analytics Vidhya: https://www.analyticsvidhya.com/blog/2020/06/4-ways-split-decision-tree/

Sunil, R. (2017, 9 11). *6 Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R*. From Analytics Vidhya: https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/#:~:text=It%20is%20a%20classification%20technique,presence%20of%20any%20other%20feature.

Sunil, R. (2017, 9 13). *Understanding Support Vector Machine(SVM) algorithm from examples (along with code)*. From Analytics Vidhya: https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

UC Irvine. (n.d.). *Reuters-21578 Text Categorization Collection Data Set*. From UC Irvine Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/Reuters-21578+Text+Categorization+Collection

van der Schans, W. P. (2020). *Finance 6500: Predicting Loan Default*. University of Utah, Salt Lake City. From https://github.com/Kydoimos97/FINAN6500DATA/blob/master/Documentation/Documentation_FINAN6500_WillemvanderSchans.pdf

6. Appendix

6.1 Appendix A

Model	**CV_Score**	**HO_Out-Of-Sample_Score**	**HO_In-Sample_Score**
XGBoost Classification	0.9537	0.9519	0.9880
Random forest Classification	0.9385	0.9405	1.0000
Decision Trees Classifier	0.9282	0.9452	0.9442
K-Neighbors Classification	0.8803	0.9018	1.0000
Logistic Regression	0.8727	0.8811	0.8841
Support Vector Classification	0.8056	0.8090	0.8050

Initial five models finan6500 project

6.2 Appendix B

Model	**CV_Score**	**HO_Out-Of-Sample_Score**	**HO_In-Sample_Score**
XGBoost Classification	0.9562	0.9506	0.9893
Random forest Classification	0.9386	0.9419	1.0000
Decision Trees Classifier	0.9373	0.9452	0.9467

Model performance after grid searching

6.3 Appendix C

Model	Sample	ACC	TPR_Neg	TPR_Pos	Precision _Neg	Precision _Pos	F1
Naive Bayes	In-sample	68.03	74.48	61.58	65.97	70.70	69.97
Naive Bayes	Out-of-sample	50.20	19.12	81.28	50.53	50.12	27.74
Random Forest	In-sample	100.00	100.00	100.00	100.00	100.00	100.00
Random Forest	Out-of-sample	52.16	66.00	38.32	51.69	52.99	57.98
Neural Network	In-sample	49.75	2.64	96.86	45.67	49.87	4.99
Neural Network	Out-of-sample	49.24	2.00	96.48	36.23	49.61	3.79
Rule-Based Classifier	In-sample	72.94	72.38	73.50	73.20	72.69	72.79
Rule-Based Classifier	Out-of-sample	55.56	75.20	35.92	53.99	59.16	62.86
Support vector Machine	In-sample	86.33	85.30	87.36	87.09	85.60	86.19
Support vector Machine	Out-of-sample	51.00	57.76	44.24	50.88	51.16	54.10
K-Nearest Neighbors	In-sample	65.18	69.80	60.56	63.90	66.73	66.72
K-Nearest Neighbors	Out-of-sample	52.16	56.72	47.6	51.98	52.38	54.25

Final Algorithm Scores

6.4 Appendix D

Algorithm list	**Run-time (Seconds)**
Naive Bayes	10
Random Forest	2604
Neural Network	57
Rule Based Classification	18
Support Vector Machine	247
K-Nearest Neighbor	174

Algorithm Run Time