The Analytics Edge 2020 Week 13 Data Competition Report

Team 35

Rainer Kwan Mun Hin (1002704), Chi Ziheng(1003030), Wu Qi(1003624)

High-level Description of our Approach

Data pre-processing

First, we processed our dataset by using the usual text processing techniques: converting all characters to lowercase, removing punctuation and stemming words. During the process, we noticed that there were features present in many tweets that were not retained after processing the data. Including these features would allow us to build a more robust document term matrix.

We noticed that the dataset contains a large amount of “@mention” and “{link}” terms. By plotting a word cloud, we observed that the words “mention” and “link” were among the most frequent words present in the dataset. This could severely affect the accuracy of the classification model. Therefore, we attempted to remove elements present in the raw data that do not help us differentiate between the three levels of sentiment. Some of the elements we removed are non-ASCII characters such as “@mention”, and “{link}”. After testing, we confirmed that the removal of these elements would improve our model accuracy.

By leveraging the “Textclean” package in R, a tool that helps clean and process texts to improve the representability of raw data. This package checks for substrings that are not optimal for analysis and replaces/removes them with substrings that are easier to analyse. After experimenting with most of the functions in this package, we narrowed it down to a few functions that are suitable for our use case. Some of the functions we implemented include: the replacement of ordinal words, adding of comma spaces, and the separation of contracted words. Logically speaking, many functions that exist in this package should help clean and improve our data. However, after testing, we realised that these functions may actually decrease the accuracy of our model, the opposite of their intended effect. Therefore, our choice of text processing functions is based on its effects on the results of our testing.

Another approach we took to boost our model accuracy was to replace the emoticons present in the tweets with meaningful words. Since our model could not process emoticons which are commonly used to express feelings, our goal was to convert those emoticons into words that could be read by the model. We defined separate functions that targeted different emoticons and replaced them with sentimental words. Some

replacement examples include converting ":)" ":D" ":-)" to smiley , ":(" ":-(" ")':" to sad and "....." to frown. We repeated this process for as many emoticons we can identify in the dataset.

Lastly, we also realised that removing stop words lowers our prediction accuracy. Therefore, we chose to retain English stop words when preparing our document term matrix.

After making the document term matrix (DTM), we decided to remove sparse terms present in the DTM at a 0.99995 level. We adjusted this parameter by using a greedy approach. Since we eventually decided on an Artificial Neural Network model (ANN) as our target model, a DTM with more variables should theoretically lead to a higher accuracy.

Model Selection: Artificial Neural Networks (ANN)

The model that produced the best results is the Artificial Neural Network model (ANN). Due to the high dimensionality of the processed dataset, complex nonlinear relationships between variables and large amounts of suitable data available, ANNs are suitable for modelling this problem.

Choice of Library

We decided to use the Keras library in R to implement our ANN model. Keras allows Tensorflow, a popular machine learning library, to be used in R. Keras provides a lot of functionality, tutorials and documentation which is extremely useful. Other R packages like neuralnet were too computationally slow for our use case whereas Keras is extremely fast in comparison.

Approach

We split the processed dataset into training and test datasets (70-30 split). Using the keras_model_sequential() function in the Keras library, we generated multiple ANN models. Through empirical testing, we found that a single hidden layer with 3 nodes consistently gave us the best results. Also, a low number of epochs (7-10) and a medium batch size (36-40) gave us consistently high accuracy numbers (~89%) when compared to the test dataset.

Other models

Before experimenting with ANNs, we tested four different approaches: Multinomial Regression; Naive Bayes; CARTs and Random Forest. We compared the performances of the four models (using our own test dataset) in terms of accuracy in the table below.

	Random Forest	Naive Bayes Classifier	CARTs	Multinomial Logistic Regression
Normal pre-processing	82.7%	76.6%	74.4%	81.6%
Fine-tuned pre-processing	82.9%	76.6%	74.5%	83.5%
Fine-tuned Hyperparameters	85.0%	76.6%	78.7%	84.7%

Normal pre-processing includes the following steps: converting text to lowercase, removing stop words and punctuation; stemming; removing sparse terms that have at least 99% empty entries. On top of it, we included additional text cleaning steps as mentioned earlier in the data pre-processing section.

Random Forest

Random forest performed the best among all the algorithms we learned in class (0.84) because it leverages on multiple classification trees and takes the best average score. However, this algorithm is difficult to interpret by nature and the space for model accuracy improvement is limited to the tuning of only a few hyperparameters. After exhausting the fine-tuning of hyperparameters, we hit an accuracy ceiling of 85%. Also, we could not derive any meaningful conclusions from the results besides the identification of the most frequent words used in most trees.

Naive Bayes Classifier

We implemented and tested the Naive Bayes Classifier model during our model exploration stage. Naive Bayes Classifier model produces fast results based on conditional probabilities. However, the accuracy was not satisfactory as it barely outperformed CART. The model assumes independence between the variables' contribution to the predicted result, which is not very applicable to the sentiment analysis of tweets since the sentiment of a tweet is highly dependent on the relationships between words.

CARTs

CART was our first attempt during our model exploration stage but it performed poorly as a single tree is unable to learn and grasp complex relationships between informal language and sentiment. We forgoed this algorithm in favour of other approaches.

Multinomial Logistic Regression

Multinomial logistic regression gives reasonably good performance especially after we used LASSO to tune its parameters. However, it did not surpass the Random Forest models as we believe it is constrained by its assumption of linearity between dependent variables and the independent variable. Words that make up tweets contain complex relationships between them and some words entail stronger emotions than others, therefore assuming that each word is linearly related with sentiment is not an ideal assumption.

Interpretability and limitations of ANNs vis-a-vis other models

CARTs vs ANNs

The CARTs model has greater interpretability but it is not able to capture the complexity of context that is embedded in the tweets. The highest accuracy achieved did not surpass 80% even after pruning. On the other hand, ANNs have the advantage of being able to analyse a given input in sequence, allowing it to interpret the context of a sentence better.

Random Forest vs ANNs

The Random Forest model lacks interpretability and is computationally intensive—hyperparameter tuning is extremely time-consuming. ANNs are much more computationally efficient in comparison.

Further improvements

1. In our initial testing with different models, we removed all English stop words as defined in the tm package. However, it is not always applicable as some models give us higher accuracies when stop words are not removed. Thus, a better approach would be to experiment with the removal of stop words one at a time to find out each word's impact on accuracy.

2. Our single layer neural network gives us the highest accuracy among all the different methods we have tried. However, the model's accuracy depends heavily on the training dataset; it is easy to overfit the model with this approach. We tweaked only the epoch and batch size values in the training process. To further improve the model performance, modifying the cost function by assigning penalty values for each parameter then finding the model with the lowest penalty score may be a good approach.