Team 5
Ng Jing Da 1003496
See Yong Chun 1003485
Ignasia Hanny 1003542

Introduction

This competition aims to develop a model capable of predicting the sentiment of weather tweets with maximum accuracy using any available package in R.

1. Approach

1.1. Understanding the Problem and the Dataset

There are a total of 22,500 tweets in the “train.csv” dataset and 7,500 tweets in the “test.csv” dataset. Only the tweets in the “train.csv” dataset are labelled with their corresponding sentiment values. The sentiment values are 1,2 and 3, which correspond to “negative”, “neutral” and “positive”. Our task is to train a model by considering these sentiments and output predictions of the sentiments of tweets in the “test.csv” dataset.

There can be a lot of ambiguity in determining the overall sentiment of a tweet because a tweet may contain multiple sentences with different sentiments. Furthermore, the language used in a tweet is often not standardised, and some words may not contribute to the overall sentiment of the tweet. This makes preprocessing of the words crucial before building our model. Our aim is thus to find a small subset of words that can be used as robust predictors to predict the sentiment of new data with decent accuracy.

1.2. Our Approach

Given the nature of this problem, i.e classification with 3 categories, some models are ruled out from the beginning, namely linear and logistic regression. This is because the sentiments to be predicted are entirely categorical and not continuous.

We thus considered models such as CART, Random Forests (RF), and Support Vector Machines (SVM), which are machine learning methods known to be effective to solve classification problems. We ruled out CART because RF, which is built upon combining multiple CART models, is more effective in handling classification problems of this caliber.

R has a very efficient Random Forests package and this method has also been widely used in multiple text classification scenarios. Instead of leaning on a single model, our team decided to concurrently run SVM models. We found that while typically RF models with more than 300 predictor variables (words) took longer than 20 minutes, SVM methods were relatively much faster. We determined an internal metric of effort-speed ratio and focused our effort on finding the trade-off between learning and fine-tuning the hyperparameters of our RF and SVM models and weighing it against the time required to return the result of the prediction. A longer run time will entail less time for us to learn and further improve the model. Below are our key findings:

1.2.1. CARTs

To develop a benchmark, we ran a CART prediction model and obtained an accuracy of 79%. After which, our group decided to perform 10-fold cross validation testing using the CART model, which increased our accuracy to approximately 81%. We set it as the baseline to set our expectations in regards to the performance of our Random Forest and Support Vector Machine models.

Team 5
Ng Jing Da 1003496
See Yong Chun 1003485
Ignasia Hanny 1003542

1.2.2. Random Forest

The baseline RF model net us an accuracy of approximately 83%, which performed better than CART as expected. We then dove deeper into tweaking the Random Forest parameters for better performance.

To enhance our pre-processing method, we attempted a combination of:

- adjusting the DTM column size (changing the amount of less frequent words removed)
- removing stopwords, emojis, emoticons, and numbers
- Utilising an n-gram approach (bigram and trigrams)
- Lemmatizing instead of stemming
- Manually removing words that seem irrelevant (e.g. rt, mention, etc)

We found that an increase in the number of predictors led to a huge increase in running time. Hence, the goal was to find a balance in minimising the number of predictor variables without compromising on the model accuracy. This was achieved through an iterative approach of adjusting our DTM sparse terms removal values between 0.992 to 0.998.

We also explored removing extra stopwords on top of the predefined stopwords from the tm package and combining it with an n-gram technique to look at sequences of 2 to 3 words to capture different types of potentially relevant predictor variables. In total, we tried building our models using a number of predictors between 100 to 560.

Our most accurate RF model has an accuracy of 85.6% with 460 predictors from the improved pre-processed training dataset. In addition to a simple Bag of Words model (counting the frequency of words in each tweet and labelling “1” and “0” if they appear), we tried to vectorize our data using *tf-idf* (Term Frequency and Inverse Document Frequency). This method highlights words of potential importance within the document by multiplying the term frequency of each word with how rarely it is found. It resulted in a marginal improvement to our model accuracy by ~0.2%.

1.2.3. Support Vector Machine

The highest SVM result obtained was a linear SVM with an accuracy of 83.7%. We found that adjusting the cost hyperparameter did not lead to any significant improvement. The SVM model ran much faster than RF, requiring 30 seconds as opposed to 17 minutes. However, on average, RF provided a much higher prediction accuracy than SVM.

By tuning the hyperparameters of each classifier, we were able to bump up our accuracy by a tiny bit each time; however that also led to longer run times, and due to the hardware limitation of our laptops, R would often crash. Furthermore, we realised that performing K-fold cross-validation on Random Forests is not feasible as it simply took too long. Moving forward, as the accuracy of our model began to plateau, our group decided to try exploring the Neural Network method.

Team 5
Ng Jing Da 1003496
See Yong Chun 1003485
Ignasia Hanny 1003542

1.2.4. Recurrent Neural Network

To implement a neural network, we first examined available datasets online to supplement our training dataset. Given that the majority of the model's complexity resides in the embedding of layers, our team decided to make use of two popular word vector models, GloVe and fastText (listed in our README.html). We considered the results of our previous models when defining the parameters for the neural network model (max words considered, max predictor variables) and underwent the same pre-processing steps, adjusted for neural networks.

To prevent overfitting, we kept track of the difference between validation loss and our training loss for each epoch. We decided that while a higher number of epochs provided greater validation accuracy, we believe that it would not give us better results on the test dataset, due to the disparity between the training loss and validation loss functions.

Running a neural network requires a lot of processing power and time. This is especially true as we considered more embeddings and higher epoch numbers. To maximize the time we have in training the model, our team added a callback function which monitors the validation loss. Once the validation loss crosses a threshold and triggers the subsequent X runs (where X = the patience we set), the model will terminate. By using this feature in conjunction with a real-time learning curve, we are able to determine if this model is worth pursuing, this differentiates it from machine learning classifiers, which provide no indication as to when it will end.

We used GloVe and fasText embeddings to enhance our neural network. GloVe leverages both the global matrix factorization and the local corpus to compute a loss function. The former aims to represent the distribution of words in a document. GloVe trained vectors can hence derive semantic relationships between words from their co-occurrence matrix. If two words occur in the same context frequently within a small word window, GloVe will force the model to encode this distribution into a global context.

Meanwhile, fastText is a method which represents each word as an n-gram of characters, rather than learning the vectors for words directly.

E.g., for n = 4, Thunderstorm is represented as "thun", "hund", "unde", "nder" … "torm"

This helps capture the meaning of rarer and shorter words and allows the embedding to connect words with different suffixes and prefixes. Once the n-gram model is completed, a skip-gram model is trained to learn the embeddings. Even if certain words are not present in the training data, when predicting the test data, the words can still be split into n-grams to obtain the embeddings.

In short, GloVe focuses on word-occurrences while fastText focuses on representing words that were not previously present in the model dictionary. We embed the preprocessed word vectors (.RDS files) from these sources for use in our model.

Team 5
Ng Jing Da 1003496
See Yong Chun 1003485
Ignasia Hanny 1003542

2. Final Results

```
Trained on 18,000 samples (batch_size=2,048, epochs=100)
Final epoch (plot to see history):
                    loss: 0.2196
    categorical_accuracy: 0.9117
                val_loss: 0.3661
val_categorical_accuracy: 0.8696
```

As seen above, our team trained our model on a batch size of 2048 with 100 epochs. In theory and practice, the number of epochs is usually set at sufficiently large or infinity to allow the model to run until it reaches the minimum value of error (denoted by loss). However, in our case, we noticed that a value of 100 or more epochs resulted in the accuracy level of our validation set (denoted by val_categorical accuracy) to plateau. Hence we capped our number of epochs at 100. Our final kaggle result has a 87.4% accuracy which was a good sign of minimal overfitting. Overall, the neural network method has proven to give a higher accuracy than our RF method. It also took less time to train. However, it presents a steep learning curve for us.

3. Possible Improvements and Limits

In total, our neural network model made use of 18,000 samples (rows of data). However, since we set our batch size to 2048, the final batch only has 1616 samples to train. The batch size could have been refined better to ensure equal number of samples per batch.

Our group uses the Long Short-Term Memory (LSTM) model which is a type of recurrent neural network capable of learning sequences of observations. In order to reduce overfitting and improve model performance, we added dropouts to the embedding layer. Dropout is a regularization method where input and recurrent connections to the LSTM layer are probabilistically excluded from activation and weight updates while training a network.The effect is to simulate a large number of networks with varying network structures to increase the robusity of the model. We set the same fixed dropout and recurrent dropout level of 0.5 for all 4 embeddings. One way to improve on this is to use a GridSearch method to test different rates systematically and return the most optimal dropout level for each embedding.

In conclusion, while the results obtained using all of our machine learning methods are not 100% replicable, the accuracy of our results are approximately consistent. Due to the randomness in the data and differences between the training and test dataset, it is difficult to obtain a very high accuracy of sentiment predictions. Considering the ambiguity of the sentiments and the lack of experience working with neural networks, our team believes that this data competition has exposed us to some important lessons when it comes to text analysis and data processing using R.