Twitter Sentiment Analysis using Natural Language Processing

Yi Xiang Chee
Student ID: 1165917

September 27, 2022

1 Introduction

Twitter is a popular platform amongst Millennials and Generation Z's, where users post in 'Tweets', which mainly consist of texts. A massive number of debates and/or exchanges take place on the platform every day, which creates an abundance of data available for analysis. This may be of interest to those in the field of politics, sociology, economics, entertainment, et cetera.

In this project, we aim to perform **sentiment analysis** to classify the nature of the tweets into one of three sentiments ('positive', 'neutral', 'negative'). It is a challenging task in the field of Natural Language Processing because traditionally, research focused on long, well-structured pieces of texts, but tweets are short and casual, which adds difficulty to ascertaining their sentiment [1].

We have a dataset [2] of approximately 22,000 tweets for training (labelled) and 6,100 tweets for testing (unlabelled). We apply and investigate some pre-processing techniques, feature selection, and machine learning models on the dataset and find an optimal model that generalises well to new data.

Our study finds that after suitable pre-processing of text, feature selection, and fine-tuning of machine learning models, we are able to classify the sentiments of our tweets with decent performance. This is evident in the results and analysis presented below.

2 Method

Our strategy is to test the models' performance using a combination of different features, pre-processing techniques, and machine learning models.

2.1 Features

2.1.1 Pre-processing of Texts

Prior to feeding the data into the training models, some pre-processing steps are performed (subsequently):

- **Replacing all URLs and usernames** with tokens "URL" and "USERNAME". Intuitively, they typically do not provide useful insights. Converting them all into their own categories could potentially alleviate the problem of overfitting.
- **Conversion of emoji into text.** Emojis are converted into their underlying meanings in text using *demojize* from the *emoji* library.

- **Lemmatisation of words** in text. Word inflections were removed (for example, '*critics*' converts into '*critic*').
- **Removal of numbers.**. Most digits carry little to no sentiment, so they are removed to reduce noise in the dataset.

2.1.2 Feature Generation

We generated our features using the TF-IDF (term frequency–inverse document frequency) of each word/token. Each word/token would be a feature and its TD-IDF is its value.

2.1.3 Feature Selection

After vectorisation, our feature space is quite enormous, having about 26,000 features. This indicates a need for feature selection to reduce noise and speed up model training.

Feature selection of our data is done using Chi-square statistics as our criterion. Since Chi-square is nonparametric, it is robust to different distributions of data and does not require any underlying distribution assumptions on our data.

2.2 Classifiers

Two commonly used models used for sentiment analysis are Support Vector Machine (SVM) and Logistic Regression.

2.2.1 Baseline Model: Zero-R

Zero-R is chosen as our baseline model. For this model, we simply predict every instance as the majority class. We will be comparing the next two classifiers relative to the performance of this model.

2.2.2 Support Vector Machine (SVM)

The first model we are using is Support Vector Machine (SVM). SVM works by finding an optimal hyperplane that maximises its margin with the instances. It is suitable for our classification problem, and faster than other geometric models such as k-NN, given the large number of features we have. As we have more than two classes, we will be using the One-vs-All strategy to convert it into a two- class problem, building just 3 SVMs.

2.2.3 Logistic Regression

The second model we are using is Logistic Regression. Logistic Regression takes continuous features as input and outputs discrete labels, using the logistic function. It was chosen since we have continuous features (TF-IDF) and we would like to generate discrete labels.

2.3 Evaluation

2.3.1 Evaluation Method

We were given a labelled training set and an unlabelled testing set, so we must generate our own validation sets from training data for model evaluation and hyperparameter tuning of models. 80-20 holdout strategy was used to generate confusion matrix and metrics easily.

2.3.2 Evaluation Metrics

One main metric we use to observe changes in performance is **accuracy** since our main goal is to classify most instances accurately. Initial inspection of our training data uncovered an imbalance of classes. We have about 58% of tweets labelled 'neutral'. This highlights the importance of relying on metrics such as **precision** and **recall**. We use the macro-averages of these metrics since we want to place equal emphasis on classifying each class correctly.

Using these three scores simultaneously gives a full overview of our models' performances. This is also used in conjunction with confusion matrices, as well as the precision and recall for each class. This strategy allows us to investigate any potential presence of class imbalance and model bias.

3 Results

3.1 Feature Reduction

The pre-processing of our data has led to a significant 42% reduction in features (see Table 1), mainly through combining words with the same lemma, URLs, and usernames into single categories respectively.

Table 1: Training set before and after pre-processing

# Features Before	# Features After
44045	25755

After evaluating our models based on different number of features trained n, n=2800 gives the best accuracy for our models (Figure 1). Hence, we choose the top 2800 features with the most significant chi-square statistics to include in our training data.

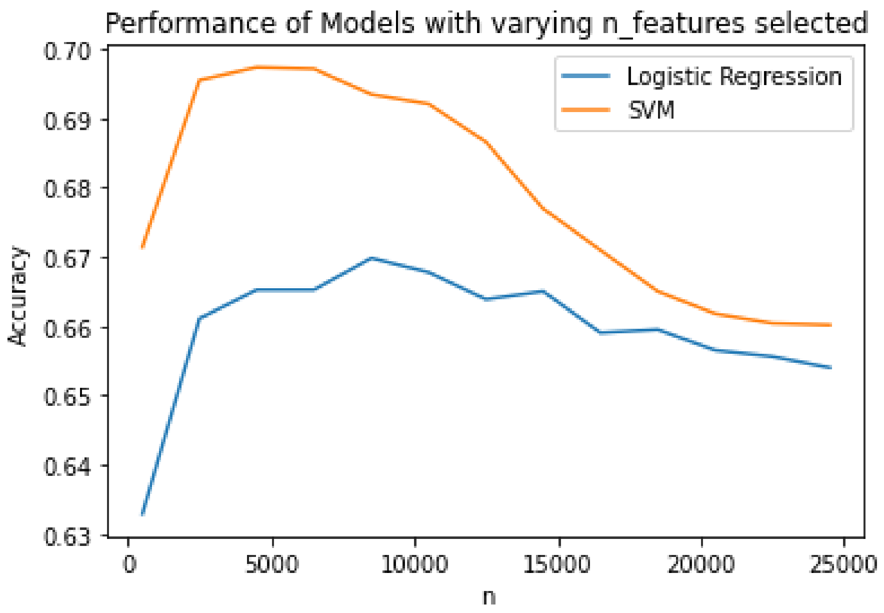


Figure 1: Accuracy of models with different number of features

3.2 Classifiers

Table 2: Initial performance of models

Classifier	Accuracy	Precision	Recall
Zero-R	57.33%	19%	33%
SVM	66.06%	63%	56%
Logistic Regression	66.41%	67%	52%

Table 3: Final performance of models

Classifier	Accuracy	Precision	Recall
Zero-R	57.33%	19%	33%
SVM	69.39%	66%	64%
Logistic Regression	66.29%	63%	68%

The initial and final results of these models are compiled in the tables above (see Tables 2 & 3). We will compare and contrast the results before and after pre-processing, feature selection, and hyperparameter tuning in this section.

- Increase across all metrics are recorded in SVM after pre-processing, feature selection, and hyperparameter tuning.
- A slight decline in accuracy (¡0.5%) and precision (¡5%) is recorded in Logistics Regression after pre-processing, feature selection, and hyperparameter tuning. However, **recall has recorded a significant increase of 16%**
- For the final performance, SVM has higher accuracy and precision, but lower recall than Logistic Regression.

To illustrate the difference in performance of the models, consider this tweet from the dataset:

"@heatherfard yo! hot uber driver last night turkeyday"

With its actual sentiment as positive, it was incorrectly classified by SVM as neutral but correctly classified by Logistic Regression as positive.

This is an example of a case that explains the higher recall for Logistic Regression than SVM. Our (macro-averaged) recall places equal importance for each class on the proportions of correctly classified instances. In other words, it is equally important to correctly classify positive/negative instances as neutral instances, despite having a highly imbalanced dataset, which we will explore in-depth in the next section.

4 Discussion / Critical analysis

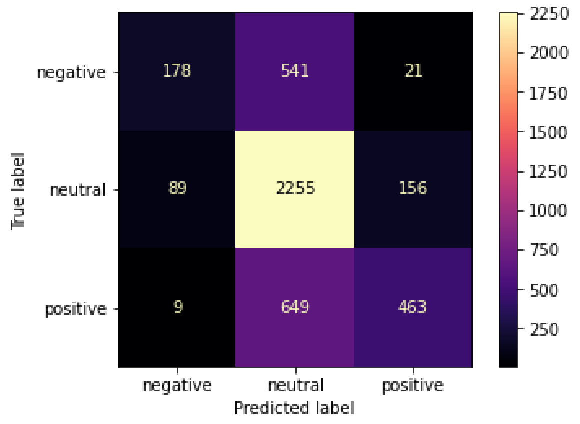
4.1 Pre-processing and feature selection

The following are some features we get before and after pre-processing and feature selection:

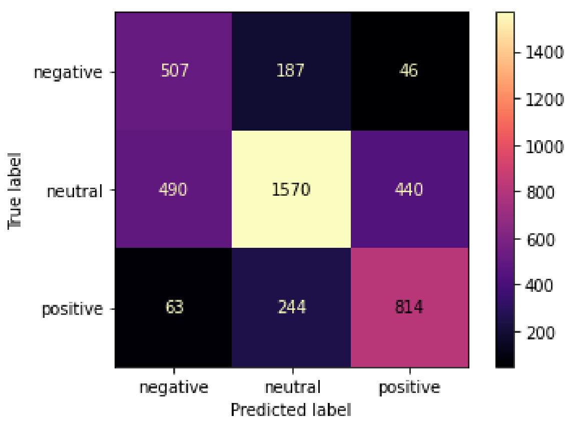
Before: *'0dgxlejij2', '0dv9dd8ss1', '0dydug2b3x'*

After: *'brings', 'brining', 'brink'*

As shown, our pre-processing method eliminated meaningless tokens such as usernames, whilst feature selection chose the most informative features for our models. Hence, our models are better at distinguishing positive/negative/neutral sentiments, as indicated in the increase in recall (see Tables 2 & 3). Looking at the confusion matrices for Logistic Regression for example (Figure 2a & 2b), we have more true positives for minority classes 'positive' and 'negative', and less bias in favour of 'neutral' predictions.



(a) Confusion Matrix of Logistic Regression (initial)



(b) Confusion Matrix of Logistic Regression (final)

Figure 2: Confusion Matrix of Logistic Regression

4.2 Model type and hyperparameter Tuning

For SVM, we explore the effects that different kernels and regularisation parameter C has on model performance.

Regularisation parameter C being too small produces large margins and being too large produces small margins for the hyperplane. From Figure 3, C=1 gives optimal performances (and optimal margin), hence it is a reasonable choice.

Amongst different kernels, polynomial kernel has the worst performance, while linear (LinearSVC) and rbf kernel (Radial Basis Function) have the best performances. Indeed, as most text classification tasks are linearly separable [3], using a polynomial kernel might have led to overfitting in this case, while it works well on LinearSVC (linear kernel) and rbf (does not assume distribution).

Rbf and LinearSVC have similar performances for $C \leq 1$, and it has been proved that their performances are asymptotically similar [4]. However, rbf takes much longer to run than linearSVC (see Table 4), since the complexity is $O(n^2)$ for libsvm (which rbf is based on), and O(n) for liblinear (which LinearSVC is based on), where n is the number of training instances. LinearSVC is preferred as it scales well with large datasets.

For Logistic Regression, our hyperparameters of interest are regularisation parameter C and solver.

Regularisation parameter C being too small tends to underfit data and being too large tends to overfit. From Figure 4, C=2 gives optimal performances, hence it is a reasonable choice.

Solvers are algorithms used to optimise loss function during training. Amongst different solvers, liblinear has the worst performance, while the rest are highly similar. The disparity in performance

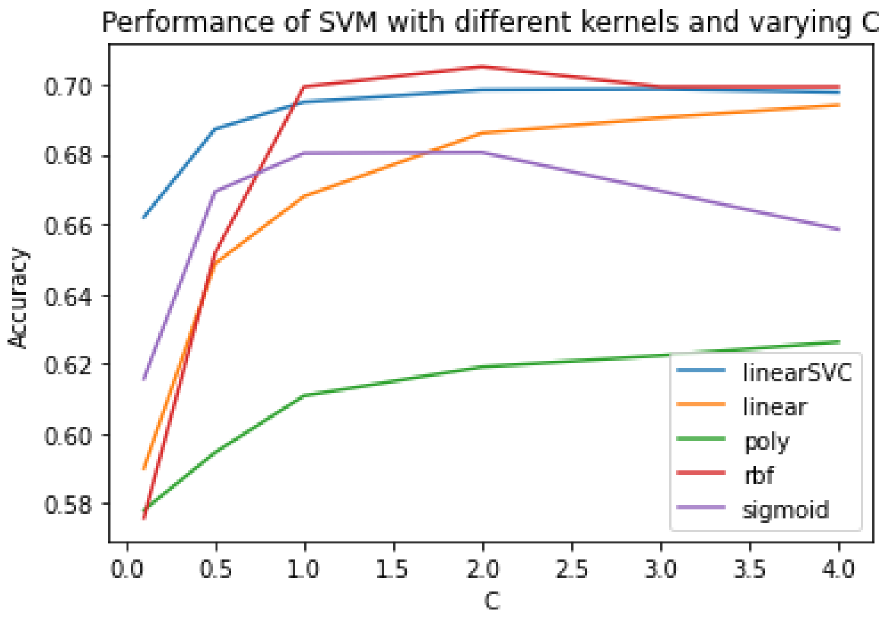


Figure 3: Accuracy of SVM with different Cs and kernels

Table 4: Training time for different SVM models

Model	Runtime (s)
SVM with rbf kernel	150.33
LinearSVC	1.86

between liblinear and other solvers probably originates from its inability to handle multinomial loss in multiclass problems, so it's not preferred for this task.

We use sag (Stochastic Average Gradient) as our solver for the model as suggested in the documentation for large datasets and feature space. It achieves convergence (in optimising loss function) faster by retaining previous gradients in memory. Again, comparing the number of iterations performed by default solver lbfgs against sag, sag takes much fewer iterations to converge (see Table 5).

Table 5: Number of iterations for each solver

Solver	C=0.5	C=1	C=2
lbfgs	111	131	216
sag	20	22	26

4.3 Performance (Recall)

The significant disparity in recalls between SVM and Logistic Regression is worth investigating. For tweet data in general, we usually obtain training sets with large feature space after pre-processing, which makes them sparse. As SVM is geometrically motivated, this adds difficulty to finding the best hyperplanes that separate classes. Conversely, this is less of an issue for Logistics Regression, a probabilistic model, which can classify positive and negative tweets correctly more than SVM, signified

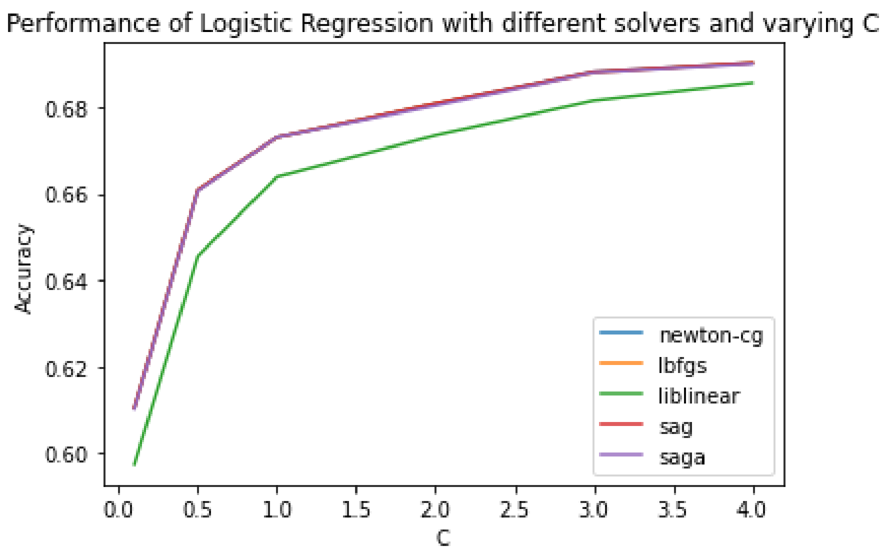


Figure 4: Accuracy of Logistic Regression with different Cs and solvers

by the high recall score.

Table 6: Recall for each class in final performance

Sentiment	Positive	Neutral	Negative
SVM	53%	78%	62%
Logistic Regression	69%	63%	73%

Comparing the recall for the two classifiers (see Table 6), even though the recall of neutral tweets is lower for Logistic Regression, the recalls for positive and negative tweets are much higher. The recalls are more consistent across classes and average higher. This again supports the argument that Logistic Regression is better at classifying different classes than SVM. To generalise our model better to different distributions, we would prefer Logistic Regression in this scenario to mitigate any sampling bias, hence it was chosen as our final model.

5 Conclusion

After suitable pre-processing, feature selection, and hyperparameter tuning, we have observed an improvement in performance on sentiment classification of tweets.

At the conclusion of this project, we determined that Logistic Regression is the most suitable model in this setting. Albeit having a slightly lower accuracy than SVM, it has better recalls across most classes. A final model is built using our tuned hyperparameters, trained using the full, pre-processed dataset, and predictions generated for submission.

We suggest some strategies for future attempts in this project, such as converting slangs/emoticons in tweets using customised dictionaries, generating features using n- grams, or using deep learning models for classification.

References

[1] Alec Go, Richa Bhayani, and Lei Huang. “Twitter sentiment classification using distant supervision”. In: *CS224N project report, Stanford* 1.12 (2009), p. 2009.

[2] Sara Rosenthal, Noura Farra, and Preslav Nakov. “SemEval-2017 task 4: Sentiment analysis in Twitter”. In: *arXiv preprint arXiv:1912.00741* (2019).

[3] István Pilászy. “Text categorization and support vector machines”. In: *The proceedings of the 6th international symposium of Hungarian researchers on computational intelligence*. Vol. 1. Citeseer. 2005, pp. 1–10.

[4] S Sathiya Keerthi and Chih-Jen Lin. “Asymptotic behaviors of support vector machines with Gaussian kernel”. In: *Neural computation* 15.7 (2003), pp. 1667–1689.