COMP30027 Twitter Sentiment Report

1. Introduction

The aim of this project is to predict sentiment of each tweet. By building model on tweets in text form, we expect to identify if a tweet is “negative”, “neutral” or “positive”. The dataset used for training has a length of 21802, and the dataset used for testing has a length of 6099. Instances in training dataset contain attributes “id” and “text”, and label “sentiment”. Instances in testing dataset do not contain label. Since text is difficult to build model directly on it, we need to preprocess the data.

2. Method

Before we start to build model, we need to clean and vectorize the data. Feature selection is needed to reduce the dimensionality.

2.1 Data cleaning

Libraries used includes unidecode, constractions, string, BeautifulSoup and nltk.

Since our focus is the texts in English, we expect to remove all unnecessary characters, for instance html symbols, accented characters, contractions, numbers, punctuations, and extra white spaces. To further improve our result, we need to remove stop words, lemmatize the remaining words in texts, and finally convert all characters to lower case. At the end of this stage, only pure texts are left, and they are all converted into their root form. This will efficiently reduce features in the processing of building model because words that sharing the same meaning are reduced to one single form.

2.2 Split Data

Although we are given training and testing datasets, testing dataset does not contain label, therefore we are not able to evaluate our model. Hence, we decide to split the clean training dataset into train_sup and test_sup by random hold-out. Test_size is set to 0.28 because this will give us 6105 test instances, which is similar to the size of our original testing dataset(6099). We use train_sup and test_sup for the rest of data preprocessing process and data modelling.

2.3 Vectorization

TFIDF is used for vectorization. TFIDF gives each word a weight to emphasize more important features and ignore less important ones. In the process of vectorization, we set ngram_range = (1,2). This allows us to analyse single words and two words together. This is because in English, two words combining sometimes completely change the meaning or sentiment. For example, “like” and “not like”, the former one might lead to a positive sentiment, however, the later one might be negative. By doing so, our modal will be more generalized, but we face increase in dimensionality. If we do not group two words together, we will have 30693 features, however, after we combine them, number of features increases to 157260(due to choosing 2 objects in a set), which is more than 5 times than before. Therefore, we must select the most relevant features out of these.

2.3 Feature Selection

We use SelectKBest to reduce features. The advantage of SelectKBest is that it can choose k best features for us to build model. If k is too large, it will lead to overfitting and long runtime. By experiments, let k = 8000 gives the best result. The method we choose here is Chi-Square.

2.4 Modelling

2.4.1 Baseline

By DummyClassifier with strategy = “most_fre quent”, we create a model with 0-R. We sim ply predict labels by the prior probabilities of the dataset.

2.4.2 Naïve Bayes

Using MultinomialNB, we create a naïve bayese model with alpha = 0.5 and fit_prior = False. Naïve Bayes modal calculates the likelihoods of each feature and predict labels based on that. It is simple and fast to build.

2.4.3 SVM

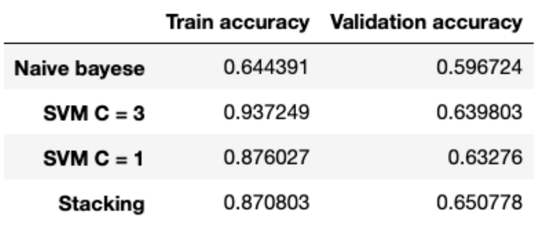
SMV performs better when data is linear separable, but we are not sure if our data meets this requirement. However, SVM is still an appropriate model because it supports large number of features in dataset. By grid search algorithm, setting C = 3 gives the best score but it may lead to overfitting, therefore we use C = 1 in this project.

2.4.4 Stacking

To build a stacking model, we use StackingClassifier. Multiple estimators are needed here. Since we have already built two models, naïve bayes and SVM, and their validation accuracies are better than the baseline. Therefore, we can choose them as the estimators for our stacking model. The final estimator is a logistic regression model, using LogisticRegression with C = 1. Since SVM gives us a good performance, logistic regression might also be suitable here because it prefers linear separable data, like SVM does. In addition, stacking model combines multiple models together, we need a final estimator that is fast and efficient to prevent long runtime. Therefore, logistic regression is suitable here (Setting C = 1 gives a rather large margin and more generalized model).

3. Result

The train accuracies and validation accuracies of models in 2.42-2.44 are presented in (Figure 1).



	Train accuracy	Validation accuracy
Naive bayese	0.644391	0.596724
SVM C = 3	0.937249	0.639803
SVM C = 1	0.876027	0.63276
Stacking	0.870803	0.650778

Figure 1-train, validation accuracy

Except for naive bayes, all models may face an overfitting issue, because the difference between the train accuracy and validation accuracy is large (Figure1).

To evaluate their performance, we need to first investigate the class distribution in the dataset (Figure 2).

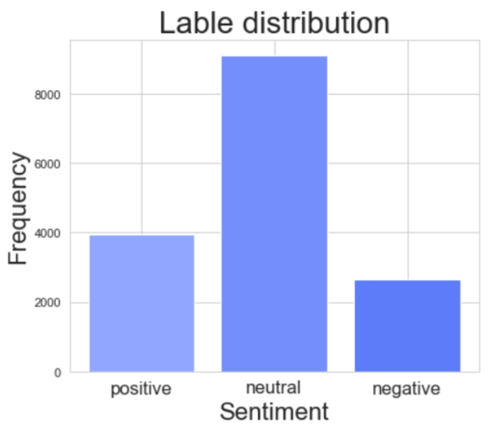
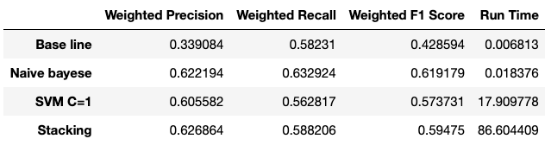


Figure 2- Class distribution

Our dataset is unevenly distributed; therefore, weighted averaging is needed here.



	Weighted Precision	Weighted Recall	Weighted F1 Score	Run Time
Base line	0.339084	0.58231	0.428594	0.006813
Naive bayese	0.622194	0.632924	0.619179	0.018376
SVM C=1	0.605582	0.562817	0.573731	17.909778
Stacking	0.626864	0.588206	0.59475	86.604409

Figure 3- Evaluation matrix

Evaluation matrix (Figure 3) shows that stacking model has the most outstanding weighted precision. However, it has a high time complexity, with an 86.6s runtime. Stacking utilizes the above models as estimators and uses a final estimator to further improve the performance, therefore, it has the best scores and longest runtime.

By considering all four evaluation methods in (Figure 3), Naïve Bayes is indeed the best model with excellent weighted precision and the highest weighted recall and weighted F1 score. Furthermore, it is extremely fast with only 0.0184s runtime.

4. Discussion/Critical Analysis

The first step of modelling is to have a baseline. Baseline is important and necessary because it allows us to decide whether the following methods are useful. Comparing accuracies or other evaluation methods of each model with baseline, we will get the idea of how much better this model is than the result of classify all instances according to the most common class in train data(0-R). In addition, preparing a baseline model helps us to get a sense for the difficulty of this project. (Figure 3) shows the weighted scores of the baseline model. With a weighted precision of 0.339, weighted recall of 0.582 and weighted F1 of 0.429, we know that there is much more space to improve. Also, low weighted precision with high weighted recall means the model doesn't need much evidence to say "positive". This is another evidence showing our dataset has an uneven distribution.

During feature selection section, we first set k = 20000, which is very big, and we receive an excellent accuracy. However, it causes overfitting and long runtime. This might because we use too many features from the training data but they may not appear in the test dataset. Therefore, our models are not generalized. (Figure 1) shows the accuracies after we reduce k to 8000. Differences between train accuracies and validation accuracies are still big but we cannot continue to reduce k as the accuracies will drop below 0.60. We also reduce C in svm model from 3 to 1, to allow more mistakes and larger margin. This makes svm model more generalized but it is still in danger of overfitting(Figure 1).

In the last section, we mentioned that Naïve Bayes is the best model. This is not expected when we build the modal. Naïve Bayes assumes conditional independence among features. In our dataset, features are English word or words, which means the assumption might be violated as the appearance of a word might depend on what the previous word is. However, the Multinomial Naïve Bayes classifier is suitable when our features are discrete no matter it is in integer or fraction. In section 2.3, we use TFIDF to vectorize each words in the dataset, and convert each feature into a fractional count. This may result in the good performance of the Naïve Bayes model. In Naïve Bayes model, we set alpha = 0.5. This is to implement a Laplace smoothing by giving unseen events a count of 0.5 and increase all counts to ensure monotonicity.

Smoothing my over-estimate, the likelihood of unseen events but it indeed helps to increase accuracy.

By plotting a confusion matrix (Figure 4) of our Naïve Bayes model, we can compare the performance on the validation set.

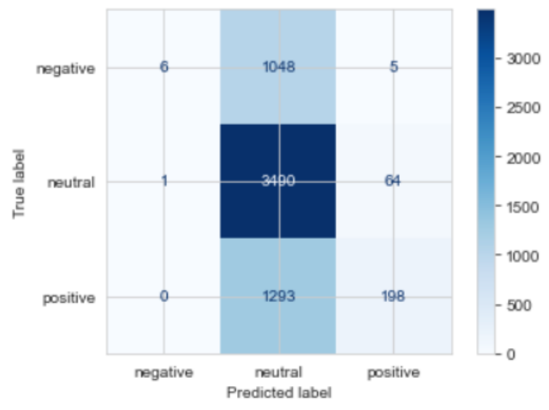


Figure 4- Confusion matrix

A good model should have a confusion matrix with high value diagonal element. In another word, we expect the plot has darker colours on the diagonal. However, (Figure 4) has darker colours on the second column, which means this modal tends to predict most instances as "neutral". We hypothesis that the reason behind is uneven distributed dataset. (Figure 2) indicates more instances are labelled as "neutral" than other labels, this causes our modal receives more instances labelled "neutral" during training and therefore tends to predict more "neutral" other than other classes. To test out hypothesis, we delete some instances labelled "neutral" and "positive", to make the number of instances in each class constant (3715 instance in each class). We use the new dataset to train the model again and receive a new confusion matrix (Figure 5).

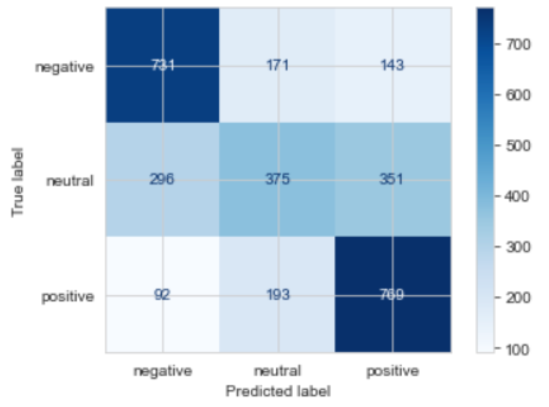


Figure 5- New confusion matrix

Now, more "negative" and "positive" instances are predicted correctly, which proves our hypothesis. However, the modal's ability of predicting "neutral" decreases. This can be shown by the dark colours in the second row, which indicates that higher False Negative regarding to "neutral".

5. Conclusions

In conclusion, data cleaning is needed to improve our accuracy for all models. Vectorization with TFIDF gives each word a weight and converts them into fractional counts of words. We then use SelectKBest with Chi Square method to select the most 8000 relevant features to reduce features. It is found that Multinomial Naïve Bayes performs better than the other classifiers. It also prevents overfitting problem and has a short runtime. We finally use the new Naïve Bayes model created in section 4 to predict our final test dataset, which is the one without labels, and receive a public score of 0.602 on Kaggle.

6. References

Rosenthal, Sara, Noura Farra, and Preslav Nakov (2017). SemEval-2017 Task 4: sentiment analysis in Twitter. In Proceedings of the 11th International Workshop on semantic evaluation (SemEval '17). Vancouver, Canada.

Solc, T. (n.d.). Unidecode: ASCII transliterations of Unicode text. [online] PyPI. Available at: https://pypi.org/project/Unidecode/ [Accessed 6 May 2022].

GeeksforGeeks. (2020). NLP - Expand contractions in Text Processing. [online] Available at: https://www.geeksforgeeks.org/nlp-expand-contractions-in-text-processing/ [Accessed 6 May 2022].

docs.python.org. (n.d.). string — Common string operations — Python 3.9.1 documentation. [online] Available at: https://docs.python.org/3/library/string.html.

PyPI. (2019). beautifulsoup4. [online] Available at: https://pypi.org/project/beautifulsoup4/.

Nltk.org. (2009). Natural Language Toolkit — NLTK 3.4.5 documentation. [online] Available at: https://www.nltk.org.

Scikit-learn.org.(2018). sklearn.feature_extraction.text.TfidfVectorizer — scikit-learn 0.20.3 documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html.

scikit-learn.org.(n.d.). sklearn.feature_selectio n.SelectKBest — scikit-learn 0.23.0 documentation. [online] Available at: https://scikit-learn.org/stable/modules/generat ed/sklearn.feature_selection.SelectKBest.html.

scikit-learn.org.(n.d.). sklearn.dummy.Dummy Classifier — scikit-learn 0.23.2 documentation. [online] Available at: https://scikit-learn.org/stable/modules/generat ed/sklearn.dummy.DummyClassifier.html.

Scikit-learn.org.(2019). sklearn.naive_bayes. MultinomialNB — scikit-learn 0.22 documentation. [online] Available at: https://scikit-learn.org/stable/modules/generat ed/sklearn.naive_bayes.MultinomialNB.html.

scikit-learn.org. (n.d.). sklearn.svm.SVC — scikit-learn 0.24.1 documentation. [online] Available at: https://scikit-learn.org/stable/modules/generat ed/sklearn.svm.SVC.html?highlight=svc#sklea rn.svm.SVC.

scikit-learn.org.(n.d.). sklearn.ensemble.Stacki ngClassifier — scikit-learn 0.23.2 documentation. [online] Available at: https://scikit-learn.org/stable/modules/generat ed/sklearn.ensemble.StackingClassifier.html.

scikit-learn.(n.d.). sklearn.linear_model.Logist icRegression. [online] Available at: https://scikit-learn.org/stable/modules/generat ed/sklearn.linear_model.LogisticRegression.ht ml?highlight=logisticregression#sklearn.linea r_model.LogisticRegression [Accessed 6 May 2022].