Paper for COMP30027 Report

Anonymous

1. Introduction

Various social media came into being in recent years. With the wide popularity and application of social media, people like to update and post their status on social media. Twitter, as a social media platform with a huge base of users, discerning the emotions in the messages people post has also become an important project. The classification of the sentiment of tweets may help the platform understand user preferences and help platforms display customized tweets based on user preferences. This project aims to build a supervised machine learning model to predict the sentiment of Tweets. A training dataset contains 21802 instances and a test dataset with 6099 instances are given. In the training dataset, each instance contains a Tweets ID, a Tweet text, and a sentiment label (positive, negative, or neutral) assigned. The testing dataset does not include a sentiment label and everything else is the same. We go through the whole process to build a machine learning model in order to predict the sentiment of the tweet texts in the testing dataset including preprocessing, feature selection, training, validating, error analysis, etc.

2. Method

2.1 Preprocessing: Data cleaning

2.1.1

We remove the column *"ID"* and *"Unnamed"* since they are irrelevant to emotion and only focused on the *"text"* column. By browsing the text column, we see a lot of distracting information that may interfere with the sentiment classification result like mixed Capital and lowercase, contraction, numbers, punctuations, URL links, hashtag to a topic, @someone, e.t.c.

2.1.2

Firstly, we convert all tweet text into lowercase to make them uniform. Then we remove duplicate characters, for example, convert *"okkkkk"* to *"ok"*. Following that we expand all the contractions and remove accented characters to ensure that words have the same meaning such as *"don't"* and *"do not"*, *"cafe`"* and *"cafe''* will not be learned as two different words. We also remove the URL, @username and HTML since they don't indicate any sentiment. So far, each tweet text consists of only standard English words and punctuations separated by spaces.

2.1.3

Secondly, we decide to keep the emojis instead of removing them as normal punctuations because they are strongly correlated with the sentiment. However, There are various faces indicating positive sentiment *(":D", ":)", ":)", ":-)")* we map them all to the same faces *":)"* using regular expressions. And we do the same for the sad faces.

2.1.4

Thirdly, we experiment with removing English stop words once we remove all numbers and extra white space. We find it may alter the nature of sentiment (e.g. *"not good"* becomes *"good"*), hence we remove stop words other than not and no since they may imply negative sentiment.

2.1.5

Fourthly, we remove all punctuation except *"?"* and *"!"* which may potentially relate to the strong sentiment. Multiple *"?"* and *"!"* are all mapped to a single one as they express the same emotion.

2.1.6

Fifthly, after the text is tokenized and lemmatized, we remove words with less or equal to 3 occurrences and more than 7000 occurrences. Since words with too high and too low frequency are not of high value, they might just be a typo or commonly used words.

2.1.7

Finally, we remove number words, months and country names as they do not indicate sentiment.

2.2 Observation of the data

Prior to training the model, we plot the distribution of the training test, we see the training test class label is very imbalanced, with 12659 Neutral labels, 5428 positive labels, and 3715 negative labels. Hence, there exist sample bias towards neutral class.

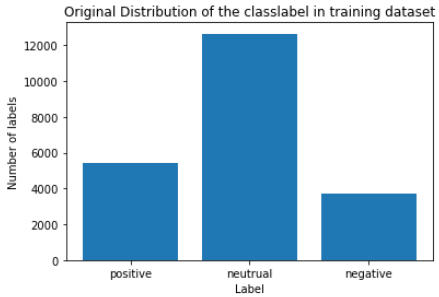

Figure 1- distribution of class labels in the whole training dataset

2.3 Random holdout, Kfold, stratifiedkfold and oversampling

Since the labels of the testing dataset are not given, the approach of splitting the training dataset into a train validation set is adopted to allow us to have a general understanding of how the model is performing. The design of the size of the validation set was based on the learning curve.

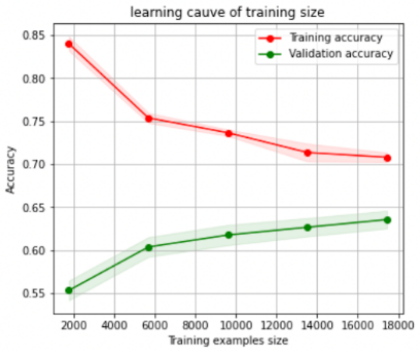

Figure 2- learning curve of training set size and accuracy, used to determine the split

From the plot, we can see that when the training set is larger than 16000, the validation accuracy grows very slowly, because of that, we set the validation set size as (21802-16000)/21802 ≈ 0.266 ≈ 0.27

We first consider using random holdout (test_set = 0.27):

- Advantage: It has a very short runtime as it only splits the training dataset once, trains the model once and predicts the validation set, and evaluates the model once.

- Disadvantage: It is less reliable compared to k folds because there is a potential issue of selecting some unrepresentative instances as the validation set and resulting in an unreal accuracy, so we only use it for grid search and choosing the vectorization method.

Then we use k folds cross-validation (k = 4):

- Advantage: the evaluations are more reliable measures of the true performance of the model. The higher the k, the closer to true performance. However, we choose k = 4 for the same reason above, to obtain a validation set size of about 0.27.

- Disadvantage: it is quite time-consuming due to the whole process being done four times.

Yet, due to the imbalance of the training set we employ a stratifiedKfold:

- Advantage: While retaining the advantages of k folds, it also preserves the proportions of positive, negative, and neutral in the sub-training datasets the same as the proportions in the original training dataset.

- Disadvantage: same as k folds.

However, when we look at the result of the baseline model 0R on Kaggle, we realize that the data in the testing set is actually balanced because 0.34645 indicates that the majority label accounts for 0.34645 of the total.

Figure 3- the test accuracy of 0R from kaggle

This implies the difference distribution in training and testing dataset, so the sample bias problem must be fixed. This leads us to consider oversampling/undersampling.

Considering that undersampling will lose some information, we decided to use oversampling. By doing oversampling, we randomly select instances from the minority class with replacements and add them to the training dataset to obtain a balanced dataset. [1]

2.4 Hyperparameter

We used grid search to find the hyperparameters of the models.

If we used BoW as our vectorization method:

SVM:

```
from sklearn import svm, datasets
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

parameters = {'kernel':('linear', 'rbf'), 'C':[0.1, 0.5, 1, 1.5, 2], "decision_function_shape": ("ovo", "ovr")}
svc = svm.SVC()
clf = GridSearchCV(svc, parameters)
clf.fit(X_train_BoW_var_selectk_over, y_train_BoW)

print(clf.best_params_)
```

```
{'C': 2, 'decision_function_shape': 'ovo', 'kernel': 'linear'}
```

Figure 4- grid search for SVM

Multinominal Naive Bayes

```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
# Multinomial NB
parameters4 = {"alpha":[1,1.2,1.4,1.5,1.6,1.7,1.8,2],
               'fit_prior':[True, False]}
clf4 = GridSearchCV(estimator = MultinomialNB(), param_grid = parameters4)
clf4.fit(X_train_BoW_var_selectk_over, y_train_BoW)

print(clf4.best_params_)
```

```
{'alpha': 1, 'fit_prior': False}
```

Figure 5- grid search for MNB

Logistic regression

```
from sklearn import svm, datasets
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from pandas import read_csv
from imblearn.over_sampling import SMOTE
from collections import Counter
from matplotlib import pyplot
from sklearn.preprocessing import LabelEncoder

parameters2 = {'solver':('newton-cg', 'liblinear', 'sag', 'saga'), 'max_iter': [500, 2000], "C": [0.2, 0.4, 0.6, 0.8, 1
clf2 = GridSearchCV(LogisticRegression(), parameters2)
clf2.fit(X_train_BoW_var_selectk_over, y_train_BoW)

print(clf2.best_params_)
```

```
{'C': 1.5, 'max_iter': 500, 'solver': 'saga'}
```

Figure 6- grid search for LR

K-nearest neighbor

```
from sklearn.neighbors import KNeighborsClassifier
parameters5 = {"n_neighbors":[5, 10, 20, 50, 100],
               'weights': ("uniform", "distance"),
               'algorithm': ('auto', 'brute')}
clf5 = GridSearchCV(estimator = KNeighborsClassifier(), param_grid = parameters5)
clf5.fit(X_train_BoW_var_selectk_over, y_train_BoW)

print(clf5.best_params_)
```

```
{'algorithm': 'auto', 'n_neighbors': 5, 'weights': 'distance'}
```

Figure 7- grid search for KNN

If we used TFIDF as our vectorization method:

SVM:

```
from sklearn import svm, datasets
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

parameters = {'kernel':('linear', 'rbf'), 'C':[0.1, 0.5, 1, 1.5, 2], "decision_function_shape": ("ovo", "ovr")}
svc = svm.SVC()
clf = GridSearchCV(svc, parameters)
clf.fit(X_train_tfidf_var_selectk_over, y_train_tfidf)

print(clf.best_params_)
```

```
{'C': 2, 'decision_function_shape': 'ovo', 'kernel': 'rbf'}
```

Figure 8- grid search for SVM

Multinominal Naive Bayes

```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
# Multinomial NB
parameters4 = {"alpha":[1,1.2,1.4,1.5,1.6,1.7,1.8,2],
               'fit_prior':[True, False]}
clf4 = GridSearchCV(estimator = MultinomialNB(), param_grid = parameters4)
clf4.fit(X_train_tfidf_var_selectk_over, y_train_tfidf)

print(clf4.best_params_)
```

```
{'alpha': 1.2, 'fit_prior': False}
```

Figure 9- grid search for MNB

logistic regression

Figure 10- grid search for LR

K nearest neighbor

Figure 11- grid search for KNN

```
from sklearn.neighbors import KNeighborsClassifier
parameters5 = {"n_neighbors":[5, 10, 20, 50, 100],
               'weights': ("uniform", "distance"),
               'algorithm': ('auto', 'brute')}
clf5 = GridSearchCV(estimator = KNeighborsClassifier(), param_grid = parameters5)
clf5.fit(X_train_tfidf_var_selectk_over, y_train_tfidf)

print(clf5.best_params_)
```

```
{'algorithm': 'auto', 'n_neighbors': 5, 'weights': 'distance'}
```

```
from sklearn import svm, datasets
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from pandas import read_csv
from imblearn.over_sampling import SMOTE
from collections import Counter
from matplotlib import pyplot
from sklearn.preprocessing import LabelEncoder

parameters2 = {'solver':('newton-cg', 'liblinear', 'sag', 'saga'), 'max_iter': [500, 2000], "C": [0.2, 0.4, 0.6, 0.8, 1,
clf2 = GridSearchCV(LogisticRegression(), parameters2)
clf2.fit(X_train_tfidf_var_selectk_over, y_train_tfidf)

print(clf2.best_params_)
```

```
{'C': 2, 'max_iter': 2000, 'solver': 'sag'}
```

2.5 Vectorization

Vectorization is an approach to feature extraction. Since we are not given any features except tweet texts, we have to do vectorization to the tweet texts to convert text to numerical vectors. [2] The techniques we consider are the term frequency-inverse document frequency (TF-IDF) and frequency count (Bag of words).

2.5.1 BoW

BoW considers each word as a feature and counts the number of its occurrences and each word is weighted the same. This might be a potential problem as the commonly used words would have a high frequency and be considered as significant features but actually not significant. Besides, we use unigrams and bigrams to solve the situation where a word has different meanings when combined with the adjacent words like *"happy"* and *"not happy"*.

We use the best parameters of the models to see the train and validation accuracy using BoW.

```
_____________________variance threshold = 0.000100_____________________
Training accuracy:
_____________using BoW k = 5000_____________
SVM:  0.7917398945518453
 LR:  0.7770577035735208
 MNB:  0.6963605741066198
 KNN:  0.9032293497363796

Validation accuracy:
_____________using BoW k = 5000_____________
SVM:  0.5785421785421785
 LR:  0.5865683865683866
 MNB:  0.5672399672399673
 KNN:  0.3647829647829648
```

Figure 12- model accuracy using BoW

2.5.2 TFIDF

Rather than the occurrence of a word, TFIDF evaluates how relevant a word is to a document in a collection of documents. It penalizes words appearing many times in each document to make them have less weight. Same as BoW, we also use unigrams and bigrams.

We use the best parameters of the models to see the train and validation accuracy using TFIDF.

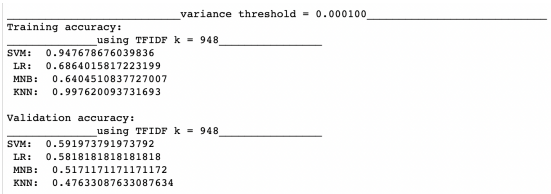

Figure 13- model accuracy using TFIDF

The comparison shows us that TFIDF gives a higher validation accuracy but the large difference between training and validation accuracy indicates a serious overfitting problem. Therefore, we decide to use BoW to do vectorization. In addition, the accuracy of KNN is too low so we do not use it for the rest of the project.

2.6 Variance Threshold

We exclude features with a very low variance which are the features with very low frequency or appear equally likely to occur in all text, we set a variance threshold of 0.0001.

2.7 Feature Selection: Select K Best

Following removing the low variance features, we select 5000 best features to train our models. F-test is particularly useful for this selection, it conducts statistical tests to check the dependence of a feature and class, each attribute has an F statistic, the higher the F statistics, the more dependency between a feature and the class.

2.8 Classifiers

2.8.1 Baseline model:

0R: It always guesses the majority class in the training set, in this case, *"neutral"*. We employ 0R before doing oversampling to the sub-training set because if the three classes are evenly distributed, there is no *"majority class"*.

2.8.2 Models:

SVM: SVM is a linear hyperplane-based classifier. It is about finding the best separating hyperplane by maximizing the margin which separates instances of different classes. We choose to use SVM because it is efficient when there are a large number of features and the data is linearly separable. In our case, we have 5000 features and therefore our data is probably linear separable as the more the features the more possible the data is linearly separable.

Multinomial Naive Bayes: it calculates the likelihood of each class given some observed features and labels the text with the class that has the largest likelihood. We utilize this model because through the research, we find that it is particularly suitable for classification with discrete features (e.g., word counts for text classification in our case) [3] and it is a popular approach in natural language processing.

Logistic Regression: Logistic regression is a supervised machine learning classification model which models the probability of a discrete outcome given an input. It is a popular approach for discrete features classification. Moreover, logistic regression has informative output, it can tell whether the feature is positive or negatively correlated with the sentiment label.

2.8.3 Ensemble model:

Stacking: Stacking is a model which is able to combine heterogeneous classifiers. We use SVM and Multinomial Naive Bayes as our level-0 base models to train the dataset and get the label as a new meta feature and adopt logistic regression as a level-1 meta classifier for the final prediction of the sentiment. Generally, Stacking is expected to have at least as good results as the best of the base classifiers.

2.9 Evaluation Metrics:

We compute training accuracy and validation accuracy. In addition, weighted precision, weighted recall, and F1 score are also computed because we believe that the weighted metrics should describe the performance of the model more accurately owing to the imbalanced dataset. Furthermore, the confusion matrix is plotted for evaluation purposes.

3. Result

	Training accuracy	Validation accuracy
0R	0.580635	0.580638
SVM	0.789386	0.577516
LR	0.776139	0.586460
MNB	0.690681	0.579992
Stacking	0.738437	0.569213

Figure 14- models accuracy

	Weighted Recall	Weighted Precision	F1
0R	0.580638	0.339865	0.427919
SVM	0.577516	0.591586	0.581317
LR	0.586460	0.599966	0.589802
MNB	0.579992	0.623235	0.587057
Stacking	0.569213	0.606733	0.576100

Figure 15- model evaluation metrics

0R having the highest validation accuracy seems to be unusual, however, this does not imply it is the best model, because it only reflects the proportion of the majority class in the validation set. Again, this accuracy reflects the data imbalanced.

All our models except 0R have a better precision than recall, according to the formulas, this indicates that they have lower false positive than false negative, which means the models do not need much evidence to say *"positive"*. The reason for that could still be the lack of training data.

baseline-0R:

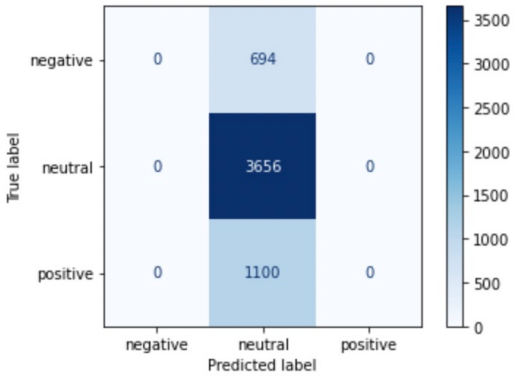

Figure 16- confusion matrix of 0R

SVM:

As we mentioned above, SVM is particularly efficient when there are a large number of features, and it does give us the best validation accuracy among the 4 classifiers. Yet, it also has an undeniable flaw which is that it is very time-consuming.

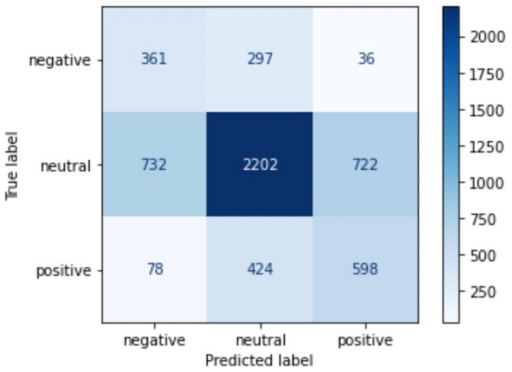

Figure 16- confusion matrix of SVM

Multinominal Naive Bayes:

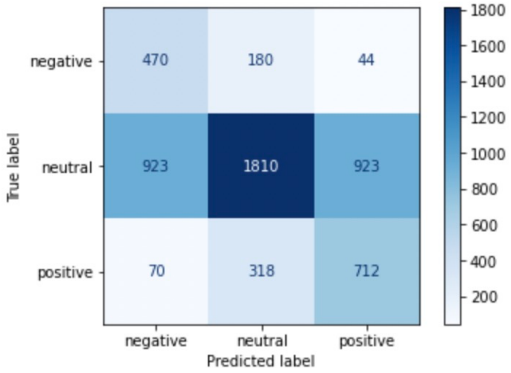

Figure 17- confusion matrix of MNB

Logistic Regression:

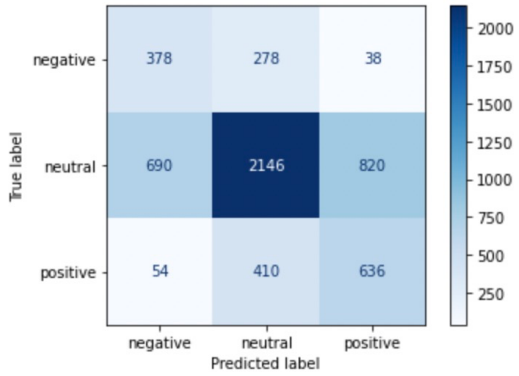

Figure 18- confusion matrix of LR

Stacking:

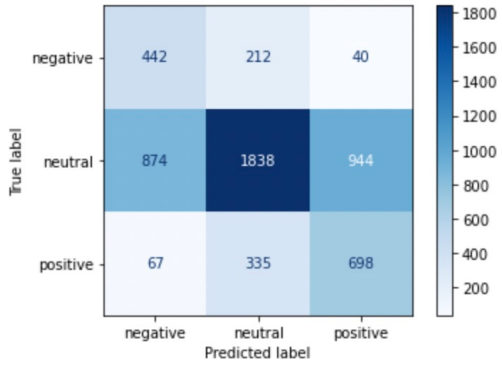

Figure 19- confusion matrix of Stacking

The confusion matrices illustrate that the base models are making similar mistakes which is an evidence of why stacking does not perform better than the best of its base model.

4. Discussion / Critical Analysis

4.1 How to resolve distribution inconsistence?

The training dataset given is imbalanced therefore, how to solve the imbalanced dataset is clearly one of the challenges. Our approach is to oversampling after splitting the training set into sub-training and validation sets to get equal number of instances of each class in the sub-training set. This approach does improve our testing accuracy, a possible reason could be the models are trained with more positive and negative instances so the ability to identify them becomes better. Another possible reason could be the testing dataset is evenly distributed according to the testing accuracy of 0R.

4.2 When to apply vectorisation?

At first, we did vectorization and feature selection to the whole training data and then split the training dataset into sub–training and validation. However, vectorization.fit_transform() would be apply to the whole training dataset so the mean and variance would be calculated based on the whole training dataset, which is a violation of the principle of validation set should not be seen by our models. Thus, we change the order so that fit_transform() is only applied on the sub-training set and the validation set uses .transform().

4.3 Oversampling

Similar to vectorization and feature selection, we use oversampling before the split but then realize we should only do oversampling to the sub-training set. If we do it before the split, the class distribution in the validation set would be altered as well. As a result of oversampling, negative, positive, and neutral are evenly distributed in the sub-training set. However, the validation set is still imbalanced due to it being excluded from the oversampling, this explains why our model performs better on the test dataset than on the validation set.

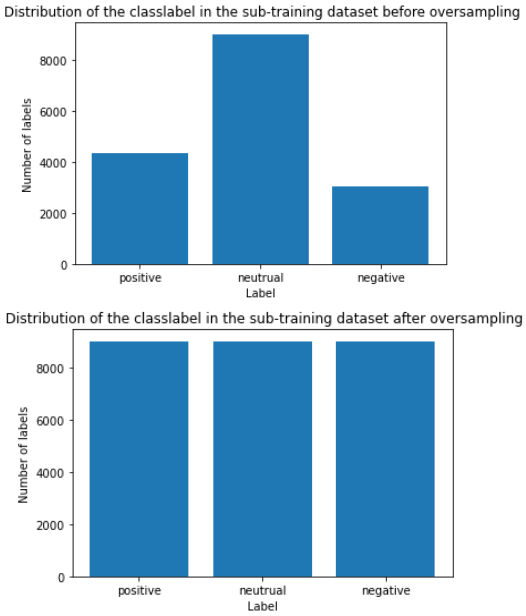

Figure 20- class labels distribution before and after oversampling

4.4 Model bias and variance

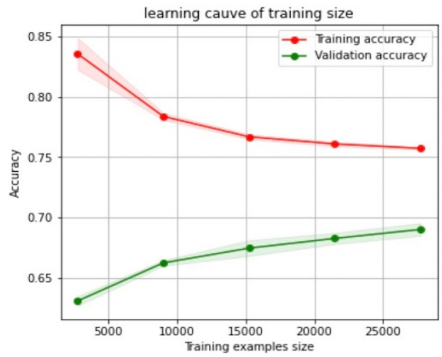

Figure 21- learning curve when using 5000 features

To analyse model bias and variance, we plot the learning curve where x-axis is the training size and y-axis is the accuracy. From the graph we can see that the curves converge to a reasonable accuracy, but the difference between the curves is relatively large. This is

an evidence of high model variance (overfitting), to reduce the variance, we reduce the number of features to 3000 and plot the learning curve again to see the difference.

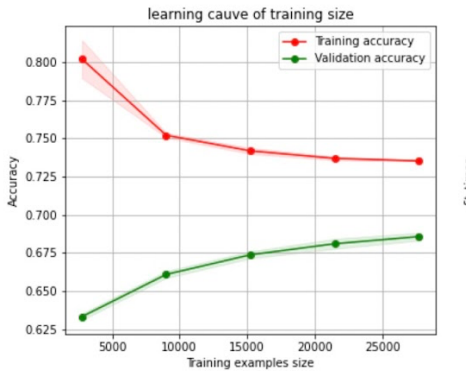

Figure 22- learning curve when using 3000 features

As the graph shows, the difference between the curves does become smaller, which indicates that reducing the number of variances is useful in reducing the variance.

4.5 Final model used to predict the testing dataset

Now, we come to the last step of the project, after diminishing the model variance, we fit the whole training dataset to get a final stacking model and use it to do the final prediction of the testing dataset.

5. Conclusions

There are several interesting findings from this project. The most learnable thing is that the oversampling approach is a handy way of dealing with imbalanced training datasets and balanced testing datasets. This project also confirms that reducing the number of features is a useful method of reducing model variance. Still, there are also some limitations to this project. For example, the performance of stacking does not meet our expectations, a possible reason for that is our base models are correlated.

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

Go A, Bhayani R, Huang L. (2009, 1(12): 2009.) Twitter sentiment classification using distant supervision[J]. CS224N project report, Stanford

[1]: Jason Brownlee 'Random Oversampling and Undersampling for Imbalanced Classification' *Machine Learning Mastery* 15, January, 2020 <https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/ >

[2]: Abhishek Jha 'Vectorization Techniques in NLP' *netuneblog* 31, December, 2021 <https://neptune.ai/blog/vectorization-techniques-in-nlp-guide >

[3]: Jason Brownlee 'Stacking Ensemble Machine Learning With Python' *Machine Learning Mastery* 10, April, 2020 <https://machinelearningmastery.com/stacking-ensemble-machine-learning-with-python/>

Abhishek Jhunjhunwala 'Is Logistic Regression a good multi-class classifier ?' *Towards Data Science* 21, feb, 2019 <https://medium.com/@jjw92abhi/is-logistic-regression-a-good-multi-class-classifier-ad20fecf1309>