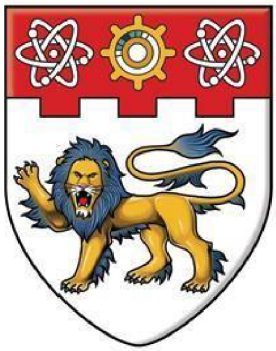


NANYANG TECHNOLOGICAL UNIVERSITY

Twitter Sentiment Analysis of the Three Constituencies (Sengkang, Hougang and Aljunied) Won by the Workers' Party in 2020 GE

Submitted By

Wang Zhiyuan G1901655F

Ji Chunxiao G1901552B

Xiao Renlin G2002298D

[2997 Words]

Table of Contents

1. Introduction

The 2020 GE of Singapore was a general election held on 10th July 2020 in Singapore. It elected members of parliament (MP) to the 14th Parliament of Singapore. For those Singaporeans who were 21 or older as of 1st March 2020, voting was compulsory (Elections Department Singapore, 2020). The election was vehemently contested by 192 candidates from 11 different parties across the island, which is the highest ever in Singapore's history since independence in 1965. Not surprisingly, the ruling party, the People's Action Party (PAP), secured its 15th consecutive term in the government because of the solid support from the masses over the years, with winning 83 out of the 93 parliamentary seats. Nevertheless, it is also observed that the largest opposition party, the Worker's Party (WP) maintained their Aljunied group representation constituency (GRC) and Hougang single member constituency (SMC) wards very successfully and seized the newly formed Sengkang GRC in the 2020 GE (ChannelNewsAsia, 2020), as the Figure 1 below demonstrates. Since it is a pretty significant change for the atmosphere of Singapore politics, this project is to study and analyze the sentiment of the general public when they talk about Sengkang, Hougang and Aljunied on Twitter two months after the election.

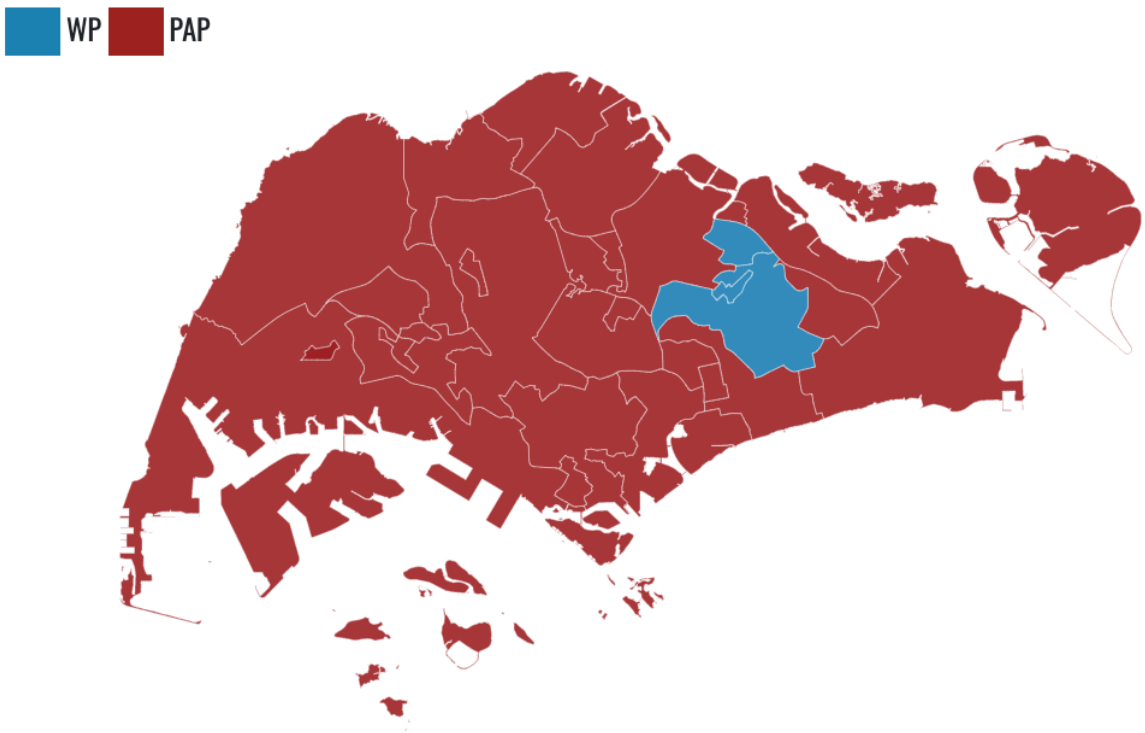


Figure 1. Constituencies won by People's Action Party (PAP) and Worker's Party (WP)

2. Tweets Scraping and Cleaning

Tweepy, is utilized to crawl those tweets related to Sengkang, Hougang and Aljunied from Twitter.com. Tweepy is a Python library that furnishes access to Twitter API. However, Tweepy does impose certain limitations on the free version with default setup, in which only tweets from past one week can be scrapped. There were some other off-the-shelf libraries for crawling tweets data for more than 7 days, such as the commonly used GetOldTweets3, but most of them are no longer working after recent updates to its API by Twitter to strengthen regulations to protect its data (StackOverFlow, 2020). Totally, we collected 472 unique tweets for Sengkang, 374 for Hougang, and 92 for Aljunied. Sengkang has more tweets because Sengkang is a new town, with many young families (Mothership, 2020) who like sharing their thoughts on social media.

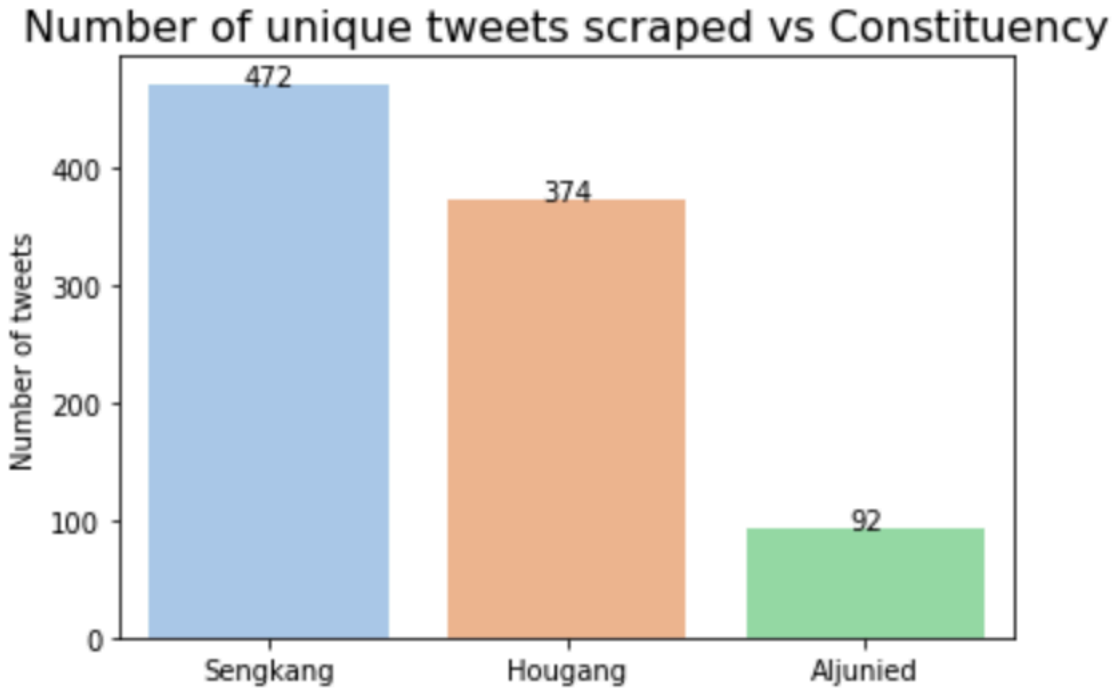


Figure 2. Number of unique tweets scraped for each constituency

Then, we cleaned the scraped tweets by transforming them into a more digestible format, so that machine learning algorithms can work better. As detailed in attached Preprocess_Tweets.ipynb, here is the cleaning sequence once a tweet is passed to the clean_text() function.

- Applying ord() function to find the integer ordinal of each character in the tweet text and only concatenate ASCII values (i.e., 128 unicode code points)

- With the help of BeautifulSoup, replacing HTML escape characters with actual ones, in other words those " to ‘’
- Removing the common-used @names from text, for instance removing ‘@wangzhiyuan’
- Removing punctuations
- Using greedy regex matching to remove web URLs, like http://... or simply www…
- Using regex to replace number with empty space
- Tokenizing the text into a list of tokens by importing nltk.tokenize
- Removing stop words from the tokens
- Stemming the tokens using PorterStemmer. Here we compared PorterStemmer, SnowballStemmer, WordNetLemmatizer, and finally selected PorterStemmer because it processes faster given that we have 1.6 million training tweets to clean as well.
- De-contract the frequently used phrases into their full forms. Such as, ‘won't’ to ‘will not’.

Note that the same sequence and steps listed above are applied to both the downloaded sentiment140 dataset with 1.6 million tweets (Kaggle, 2017) and the scraped tweets for each constituency.

3. Training of Various Classifier Models

In this section, various classifier models, namely, multilayer neural networks (MNN), convolutional neural networks (CNN), multinomial naive bayes, logistic regression, decision tree and adaboost are trained when using the cleaned sentiment140 dataset, which has very balanced sentiments labelled, i.e., half positive and half negative (Figure 3).

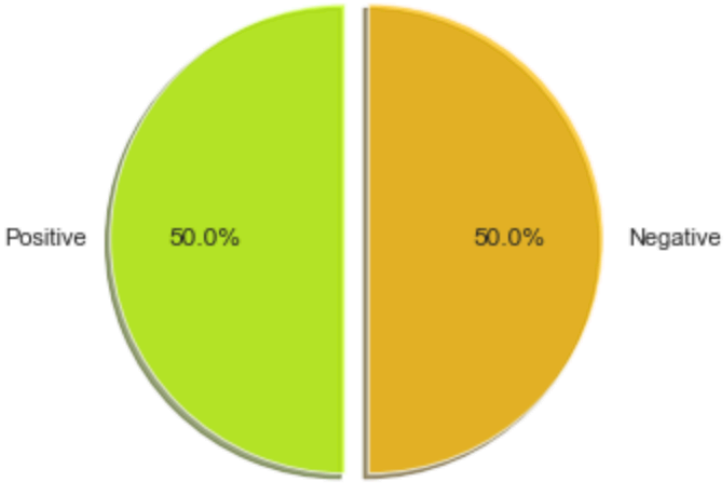


Figure 3. Percentage of positive and negative tweets for the sentiment140 dataset

We then split 70% of the dataset for training, 15% for validation and the remaining 15% for testing (Figure 4).

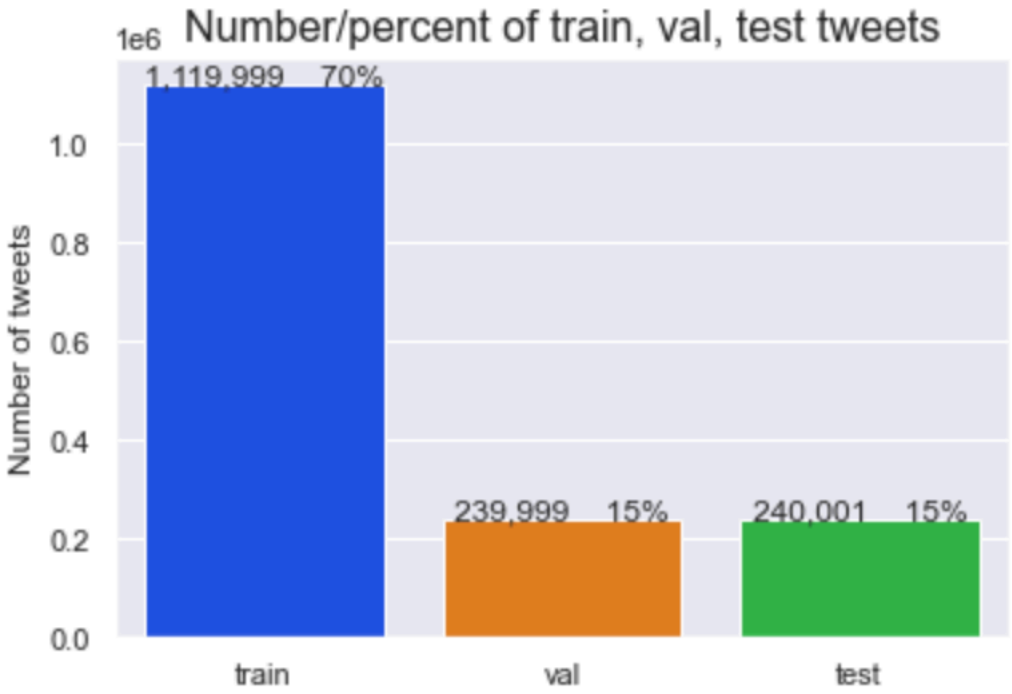


Figure 4. Number and percentage of train, val and test tweets from sentiment140

3.1. Multilayer Neural Networks (MNN)

For this model, we used code in 3_5_Classifying_Yelp_Review_Sentiment.ipynb of mlp-sentiment as our base code (Na, 2020) and further modified on top of it to find the hyperparameters that can lead to the highest test accuracy. Since this MNN model is only

trained in my personal computer with CPU, which cannot handle such a huge dataset (i.e., 1.6 million tweets), we decided to take 2% of the original dataset for training (Figure 5 & 6)

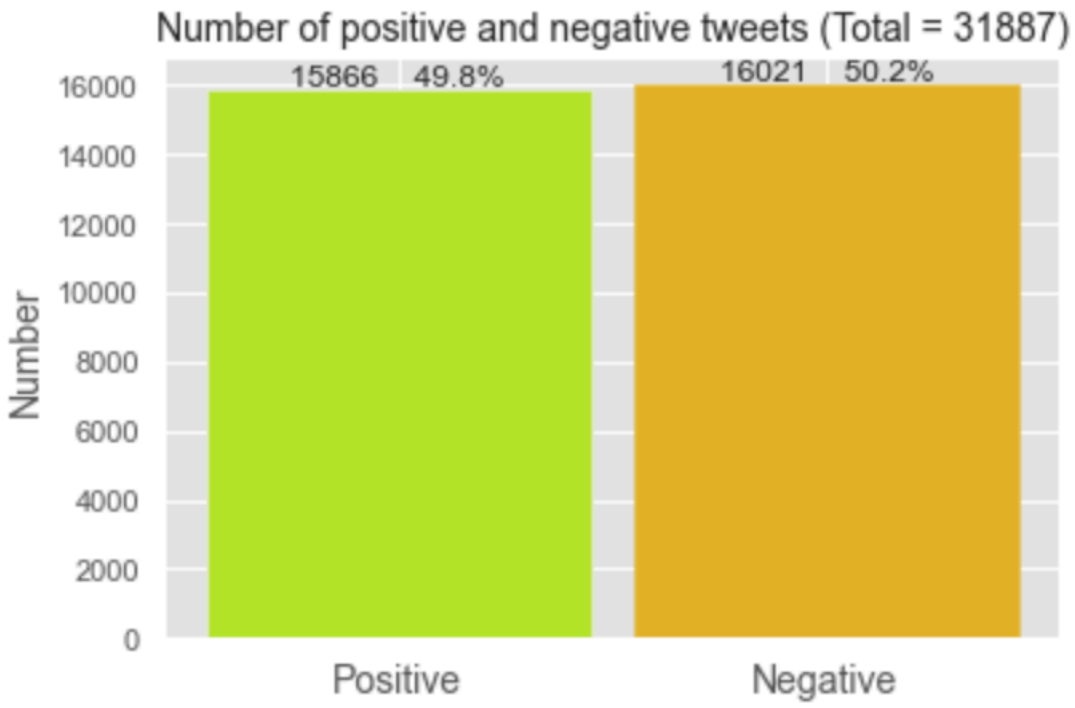


Figure 5. Number of positive and negative tweets after taking 2% from the original

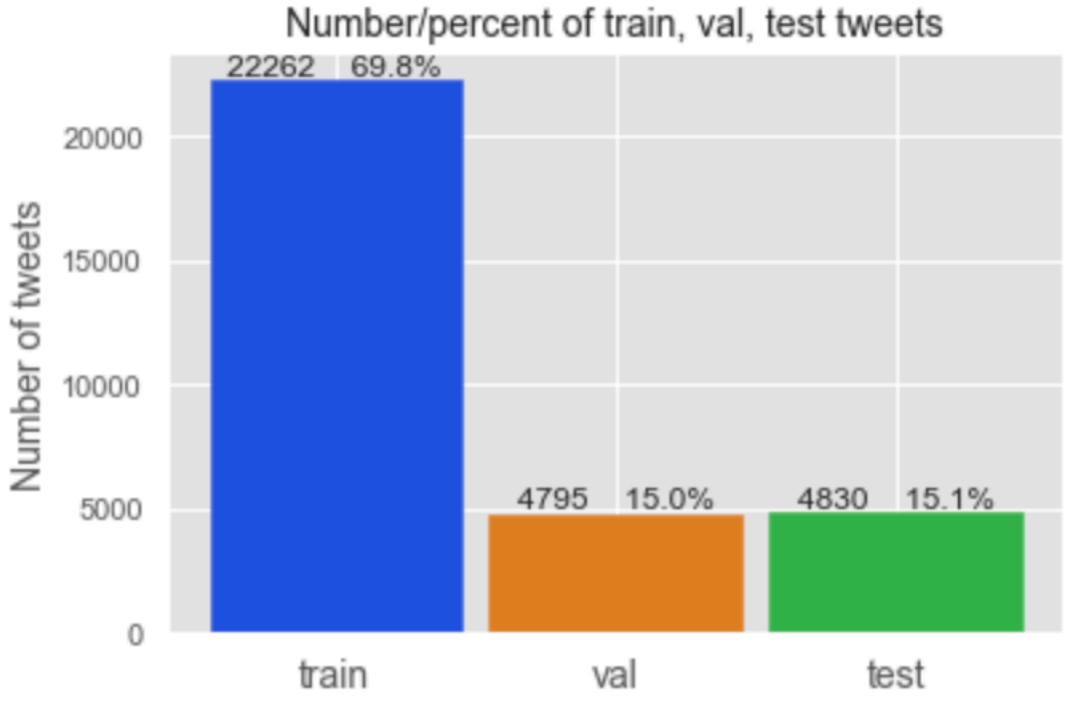


Figure 6. Number and percentage of train, val and test tweets after taking 2% from the original

3.1.1. Study the Effect of Adding Bigrams and/or Trigrams

To study the impact of adding bigrams and/or trigrams, first, we created vectorizer and one-hot vector based on single words (i.e., unigrams) only. Second, use both single words and bigrams. Lastly, use single words, bigrams and trigrams. See the from_dataframe() and vectorize() functions in TweetVectorizer class for the codes. After that, train a classifier for each case and determine the train, val and test accuracies. The results is shown in Figure 7.

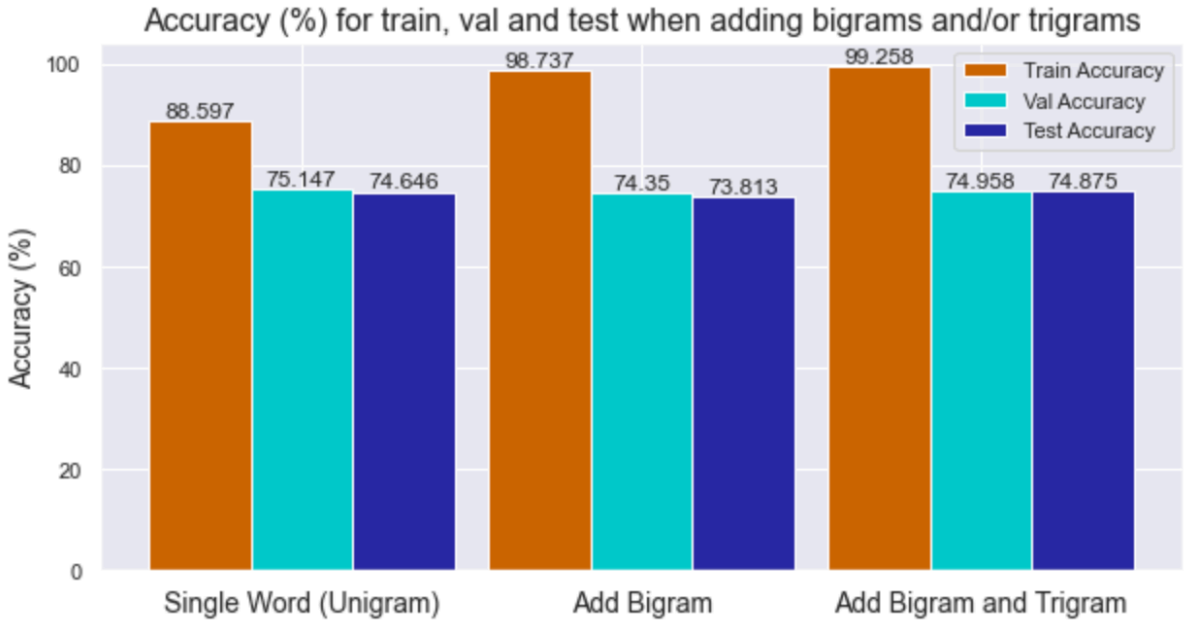
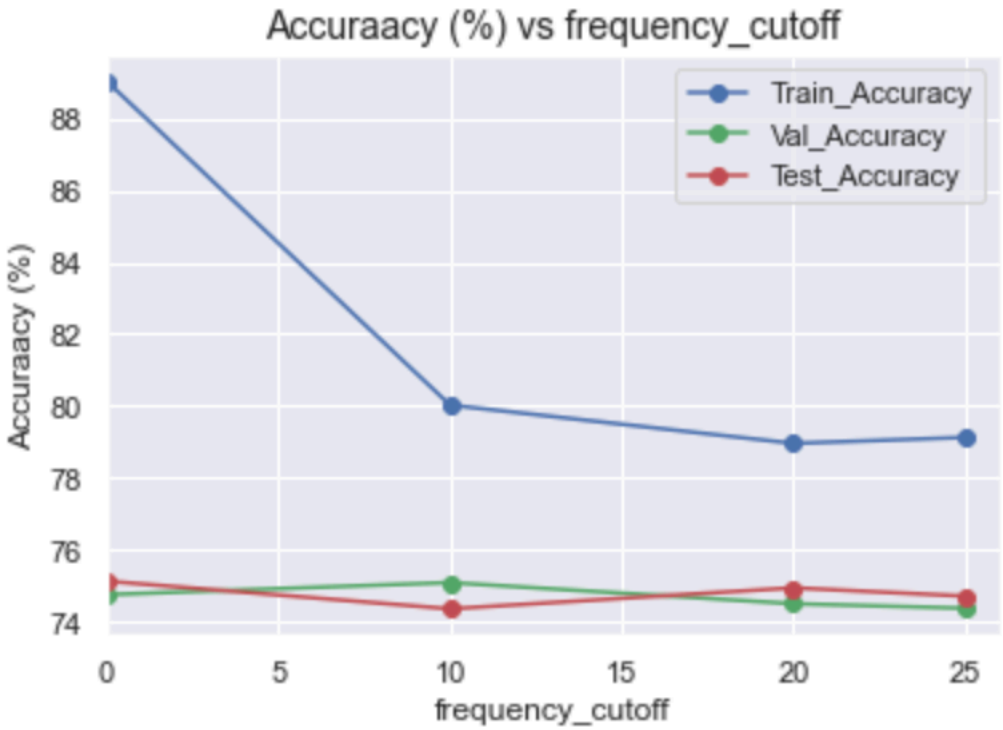


Figure 7. Accuracy (%) for train, val and test when adding bigrams and/or trigrams

As observed, adding bigrams and/or trigrams can further improve the train accuracy, from 88.597 to 98.737, and then further to 99.258. Nevertheless, it does not help much on val and test accuracies, and even slightly lower them. This phenomenon is because the classifier tends to memorize the training dataset when details are furnished. Additionally, adding bigrams and/or trigrams causes a huge surge of training time, almost 10 times in comparison with using the vectorizer made from unigrams only. Hence, single words vectorizer will still be employed.

3.1.2. Study the Effect of Increasing Frequency_Cutoff When Creating Vectorizer

A frequency_cutoff value is used when creating the vectorizer in the from_dataframe() function. Only when the frequency of a word is greater than the cutoff value, will it be considered as a valid token. The default cutoff value is set as 0.



frequency_cutoff	Train_Accuracy	Val_Accuracy	Test_Accuracy
0	89.038	74.748	75.125
10	80.022	75.084	74.354
20	78.957	74.497	74.938
25	79.123	74.371	74.708

Figure 8. Accuracy (%) vs frequency_cutoff

As can be seen from Figure 8, when frequency_cutoff value is increasing from the default 0, val and test accuracies are comparatively stable, whereas the train accuracy significantly drops. Hence, frequency_cutoff == 0 will be used.

3.1.3. Find the Best Learning Rate

Learning rate is one of the critical hyperparameters in training the MNN classifier model using gradient descent. It controls how fast the model can be adapted to the problem.

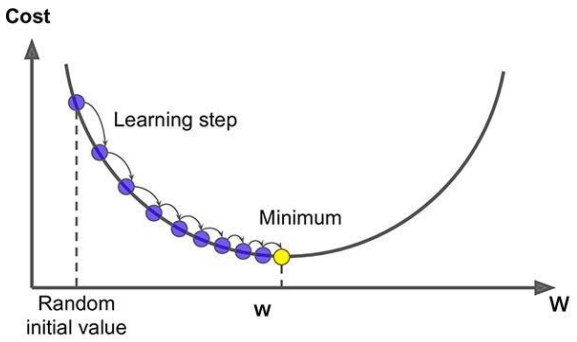
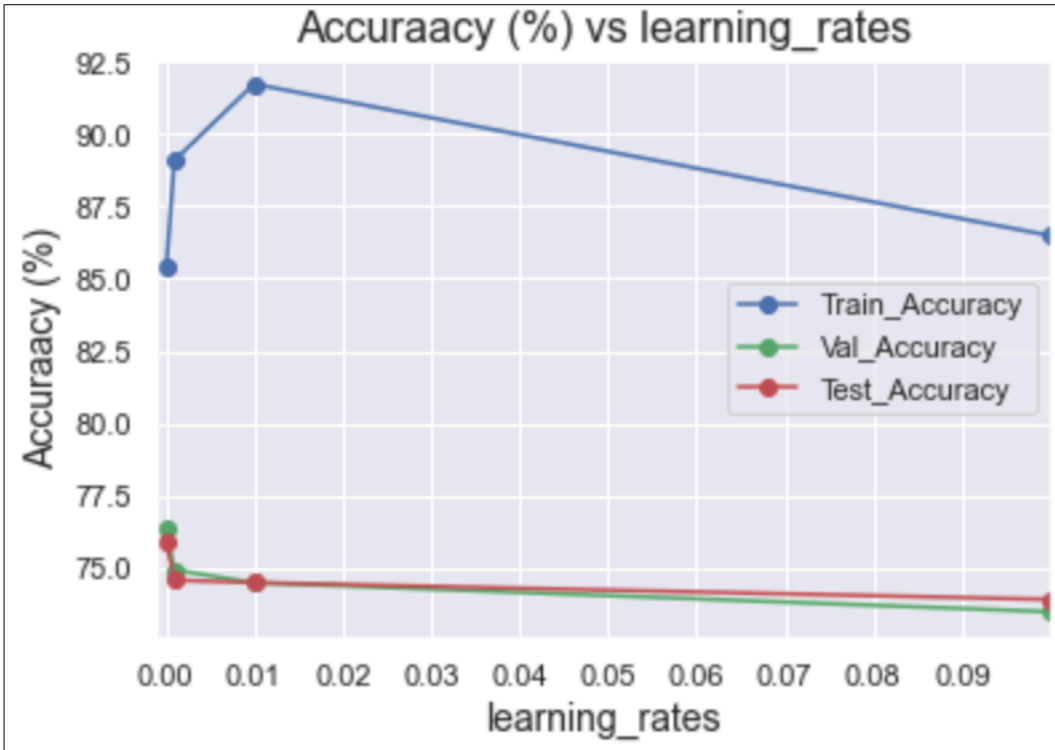


Figure 9. Gradient descent illustration (Gorman, 2017)



learning_rates	Train_Accuracy	Val_Accuracy	Test_Accuracy
0.0001	85.414	76.342	75.854
0.001	89.092	74.916	74.562
0.01	91.736	74.476	74.479
0.1	86.488	73.469	73.896

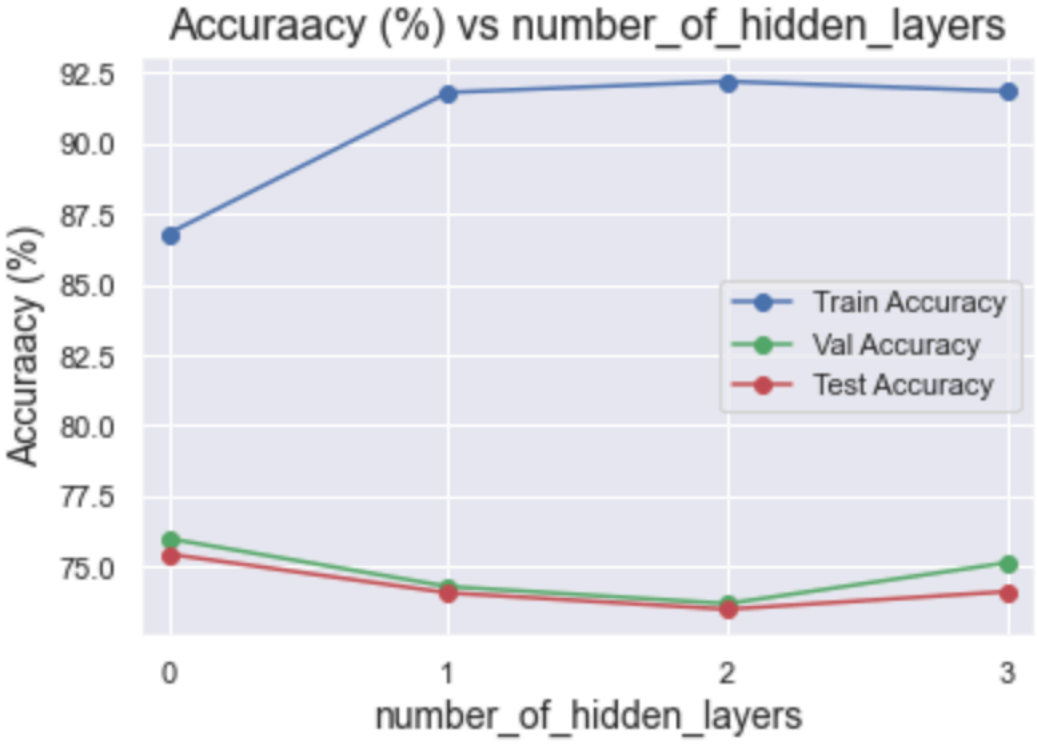
Figure 10. Accuracy (%) vs learning_rates

It is seen from Figure 10, when learning_rate == 0.0001, the highest val and test accuracies can be reached, but train accuracy becomes the lowest. When learning_rate == 0.001,

a second higher train, val and test accuracies are obtained. When learning_rate == 0.01, with a very small compromise of val and test accuracies, train accuracy can be further improved to 91.736. Hence, learning_rate == 0.01 will be used.

3.1.4. Find the Best Number of Hidden Layers

Except the input layer and output layer, the other layers in the middle are called hidden layer. We target to find the optimal number of hidden layer.



num_hidden_layers	Train_Accuracy	Val_Accuracy	Test_Accuracy
0	86.812	75.986	75.438
1	91.812	74.287	74.062
2	92.203	73.679	73.479
3	91.857	75.126	74.104

Figure 11. Accuracy (%) vs number_of_hidden_layers

As observed from Figure 11, when num_hidden_layers == 2, the highest train accuracy can be reached, but val and test accuracies becomes the lowest. when num_hidden_layers ==

0, val and test accuracies can reach the highest, but train accuracy becomes the lowest. when num_hidden_layers == 3, second highest of train, val and test accuracies can be obtained. Hence, num_hidden_layers == 3 will be applied.

3.1.5. Find the Best Number of Hidden Units in Each Hidden Layer

Hidden units is the number of neurons at each hidden layer of the neural network. We target to find the optimal number of hidden units.

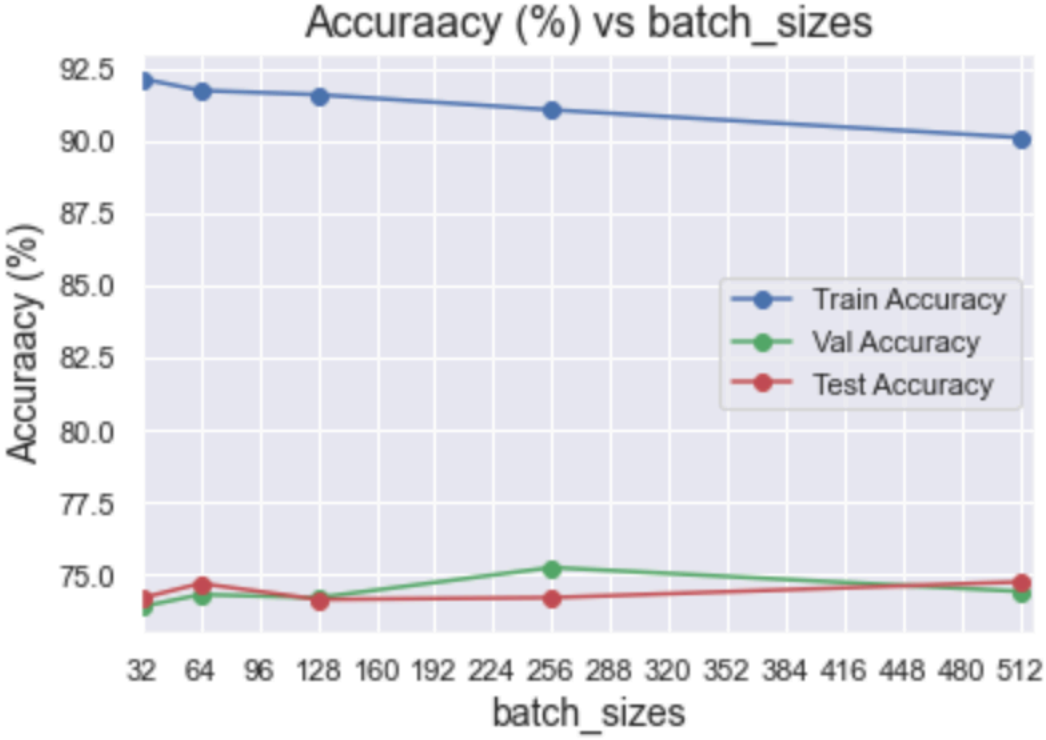
hidden_units	Train_Accuracy	Val_Accuracy	Test_Accuracy
5	87.361	74.224	74.604
10	90.823	74.518	74.187
20	92.343	73.448	74.521
30	93.291	74.266	73.313

Figure 12. Accuracy (%) vs hidden_units

As clearly demonstrated in Figure 12, when hidden_units == 20, second highest train and test accuracies can be achieved. The others will have more compromise either on train or test accuracies. Hence, hidden_units == 20 will be utilized.

3.1.6. Find the Best Mini Batch Size

Rather than implementation of gradient descent on the whole training dataset, we split our training set into smaller sets, also called mini batches, which will make the training faster.



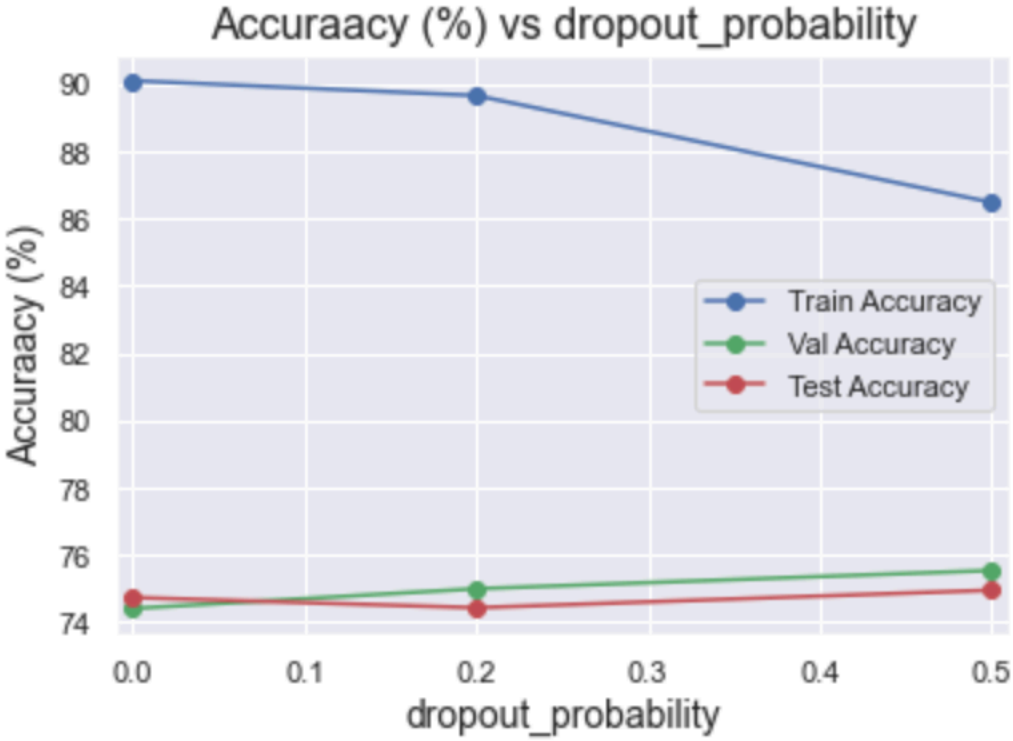
batch_sizes	Train_Accuracy	Val_Accuracy	Test_Accuracy
32	92.127	73.888	74.188
64	91.719	74.303	74.687
128	91.578	74.198	74.134
256	91.052	75.239	74.197
512	90.098	74.414	74.740

Figure 13. Accuracy (%) vs batch_sizes

As Figure 13 presents, when batch_sizes == 512, the highest test and second highest val accuracies can be achieved, with a minor drop of train accuracy, which is still above 90. Hence, batch_sizes == 512 will be applied.

3.1.7. Find the Best Dropout Probability

Dropout regularization is a simple yet effective way to prevent overfitting for the neural networks.



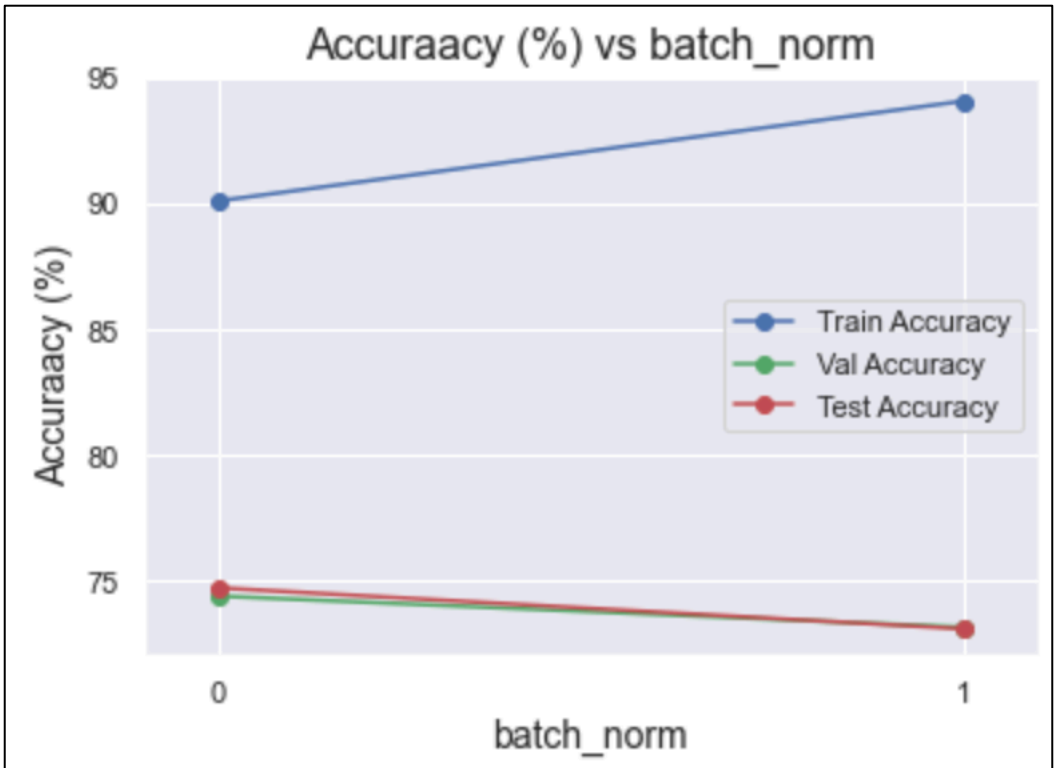
dropout_probability	Train_Accuracy	Val_Accuracy	Test_Accuracy
0	90.098	74.414	74.740
0.2	89.658	75.000	74.436
0.5	86.483	75.543	74.957

Figure 14. Accuracy (%) vs dropout_probability

As observed from Figure 14, Adding dropout can lead to a slight increase of val and test accuracies, but cause a bigger drop for train accuracy. Hence, the default setup with dropout_probability == 0 will still be applied.

3.1.8. Study the Effect of Adding Batch Norm

Batch normalization is a method standardizing the inputs to a layer for each mini-batch to be used.



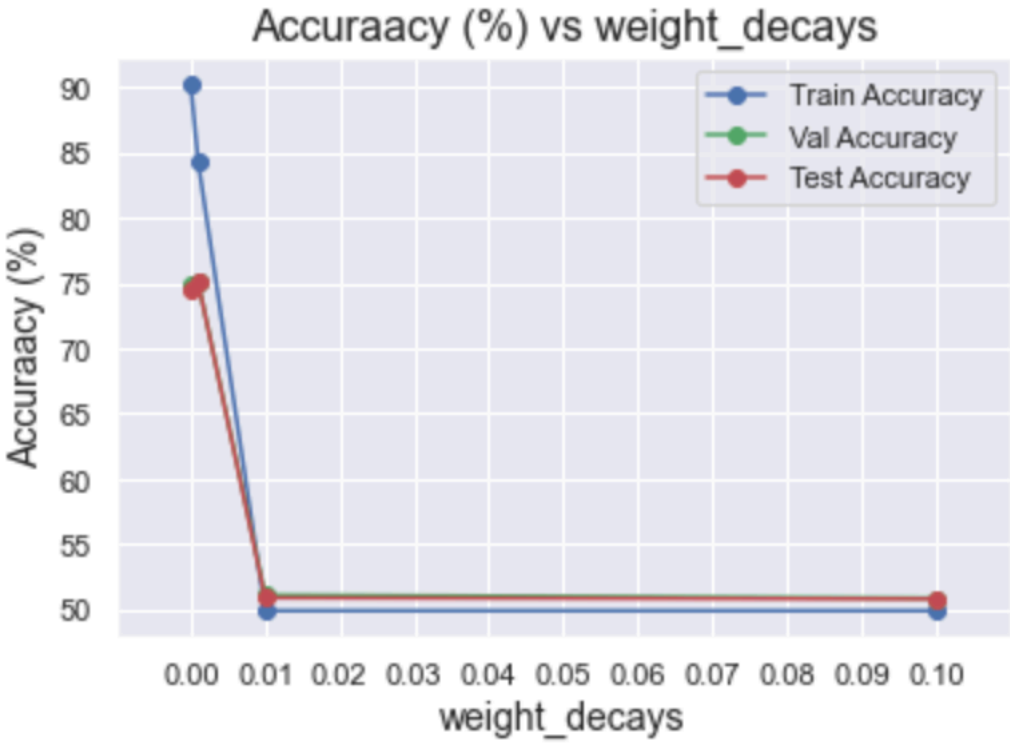
batch_norm	Train_Accuracy	Val_Accuracy	Test_Accuracy
0	90.098	74.414	74.740
1	94.063	73.177	73.112

Figure 15. Accuracy (%) vs batch_norm

It can observed from Figure 15 that adding batch norm can result in an increase of train accuracy, from 90.098 to 94.063, which is quite high. However, both val and test accuracies are reduced by more than 1%. Hence, No batch norm will be applied. In fact, in text mining field, batch norm is not that critical as compared to data mining, because we used the one-hot vectors that consist only of many 0s and 1s.

3.1.9. Find the Best Weight Decay

With L2 regularization, after each update, the weights are usually multiplied by a factor slightly less than 1 to prevent overfitting. This is called weight decay.



weight_decays	Train_Accuracy	Val_Accuracy	Test_Accuracy
0	90.321	74.848	74.523
0.001	84.348	75.130	75.109
0.01	49.923	51.128	50.911
0.1	49.918	50.890	50.825

Figure 16. Accuracy (%) vs weight_decays

As observed from Figure 16, adding weight_decay will cause the decrease of train, val and test accuracies. Thus, weight_decay == 0 will be used.

3.1.10. Apply the Best Hyperparameters Found to Train the MNN Classifier

The summary of the findings from the subsections 3.1.1 to 3.1.9 is as below:

- Only use single words (unigrams) for vectorizer and one-hot vector

- frequency_cutoff == 0 when creating vectorizer
- learning_rate == 0.01
- number_of_hidden_layers == 3
- hidden_units == 20
- batch_size == 512
- no dropout
- no batch_norm
- weight_decay == 0

With these, the train accuracy is found to be 90.039%; the val accuracy is found to be 74.154%; the test accuracy is found to be 74.175% as the Figure 17 below shows.

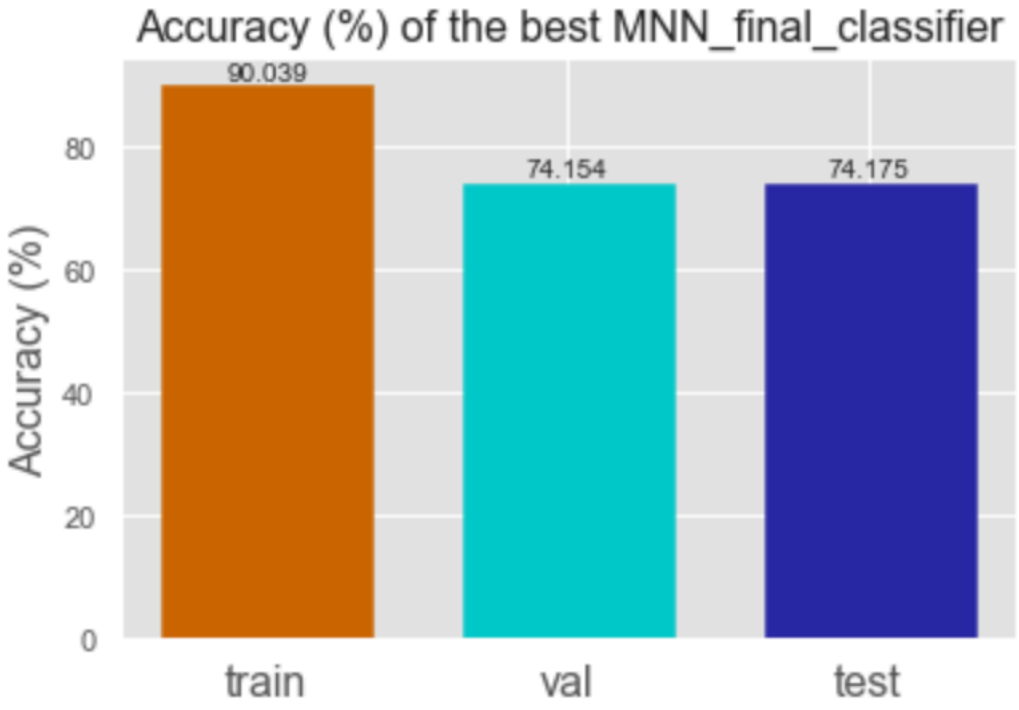


Figure 17. Accuracy (%) for train, val and test when applying optimal hyperparameters

Inference examples using the trained MNN model:

```
Long time ago.. when I was in sengkang.. there were many road work when I cross traffic light..
I was wondering.. why so many work?
 -> negative

WP MPs like Pritam Singh and Jamus Lim put up a strong proposal for a $1,300 monthly minimum wage
to support low-income workers. Sengkang MP Raeesah Khan has also spoken up on this issue as
someone who has worked with those who earn below livable wage
 -> positive
```

3.2. Convolutional Neural Networks (CNN)

In this subsection, we trained a CNN classifier and found the hyperparameters that can lead to the highest test accuracy. As mentioned above, since our computer cannot handle huge dataset, the same training dataset for MNN model is applied to train CNN model. Three CNN models with different training setting are introduced. The difference between three models is whether they allow word embedding learned from scratch. The allowance of such will lead to overfit the training data a little bit. Nonetheless, the result is still acceptable.

3.2.1. CNN Parameters Tuning

The main parameters for CNN layer are the layer activation of each layer and the batch size (Kim, 2018). Three tables below are the tuning records. CNN01 is fed with pre-trained word sectors as weights initialization. CNN02 is the basic CNN model. CNN03 allows self-trained word embedding on the top of CNN01.

After tuning the parameters, the highest test accuracy of 77% can be achieved when employing CNN01 with the highlighted setting in Table 1.

Table 1. Parameters tuning for CNN01 that has pre-trained word sectors as weights initialization

Conv1D	Dense layer1	Dense layer2	Batch Size	MaxAccuracy
relu	relu	relu	64	0.69
sigmoid	relu	relu	64	0.67
softmax	relu	relu	64	0.62
relu	sigmoid	relu	64	0.69
relu	softmax	relu	64	0.64
relu	relu	sigmoid	64	0.71
relu	relu	softmax	64	0.68
relu	relu	sigmoid	128	0.69
relu	relu	sigmoid	32	0.77
relu	relu	sigmoid	16	0.75

Table 2. Parameters tuning for CNN02 basic CNN model

Conv1D	Dense layer1	Dense layer2	Batch Size	MaxAccuracy
relu	relu	relu	64	0.71
sigmoid	relu	relu	64	0.62
softmax	relu	relu	64	0.65
relu	sigmoid	relu	64	0.61
relu	softmax	relu	64	0.7
relu	relu	sigmoid	64	0.65
relu	relu	softmax	64	0.77
relu	relu	softmax	128	0.71
relu	relu	softmax	32	0.72

Table 3. Parameters tuning for CNN03 that allows self-trained word embedding on the top of CNN01

Conv1D	Dense layer1	Dense layer2	Batch Size	MaxAccuracy
relu	relu	relu	64	0.74
sigmoid	relu	relu	64	0.61
softmax	relu	relu	64	0.68
relu	sigmoid	relu	64	0.67
relu	softmax	relu	64	0.58
relu	relu	sigmoid	64	0.7
relu	relu	softmax	64	0.72
relu	relu	relu	128	0.72
relu	relu	relu	32	0.72

3.3. Other Classifier Models From Scikit-learn Library (Multinomial Naive Bayes Classifier, Logistic Regression Classifier, Decision Tree Classifier, AdaBoost Classifier)

Scikit-learn Library, one of the most popular open source machine learning libraries for Python, provides many algorithms for classification (Scikit-learn, 2020). Here, we imported 4 of them, namely, Multinomial Naive Bayes, Logistic Regression, Decision Tree and AdaBoost. Besides, we also imported TfidfVectorizer to create the vectorizer based on the train data and generate train and test input vectors. After the trainings, the aforementioned MNN and CNN are compared with them in terms of train and test accuracies as the Figure 18 and 19 below describe.

Figure 18. Train Accuracy (%) for all models

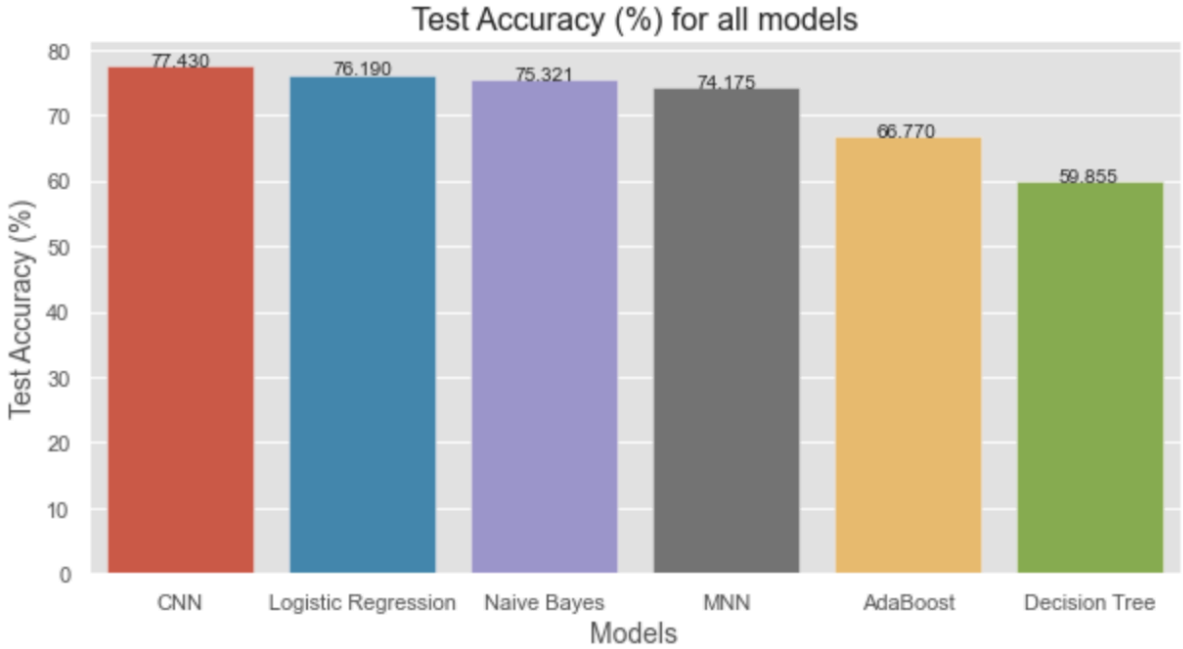


Figure 19. Test Accuracy (%) for all models

It is observed that Multinomial Naive Bayes Classifier has the highest train accuracy and 3rd highest test accuracy. Logistic Regression Classifier has the 2nd best train and test accuracies. Performances of AdaBoost Classifier and Decision Tree Classifier are not comparable with the rest. CNN reaches the highest test accuracy, but its train accuracy is

relatively low. MNN preforms well, with good train accuracy, as well as test accuracy that is only 3% lower than the highest.

Then, we chose the Logistic Regression Classifier to find some of its optimal hyperparameters by importing GridSearchCV from sklearn

- Step 1 : decide the most optimal solver to be used. As per the official API, there are several solvers, 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'. After the iteration, newton-cg is selected.
- Step 2 : find the best C (inverse of regularization strength). When C == 2.2, result is optimal.
- Import cross_val_score from sklearn to assess the predictive performance of the model; see how it performs when applying to new data.
- Apply these tuned parameters and train the Logistic Regression Classifier again.

As demonstrated in Figure 20, while using the updated parameters, its train accuracy is increased by 5% and test accuracy maintains.

Figure 20. Train and test Accuracy for Logistic Regression Classifier using default and optimal hyperparameters

4. Sentiment Prediction and Results Visualization

We applied both the optimal logistic regression classifier and best MNN model obtained from Section 3.1 to predict the sentiment for tweets discussing about the three constituencies. The results are fairly similar, as shown in Figure 21 and 22.

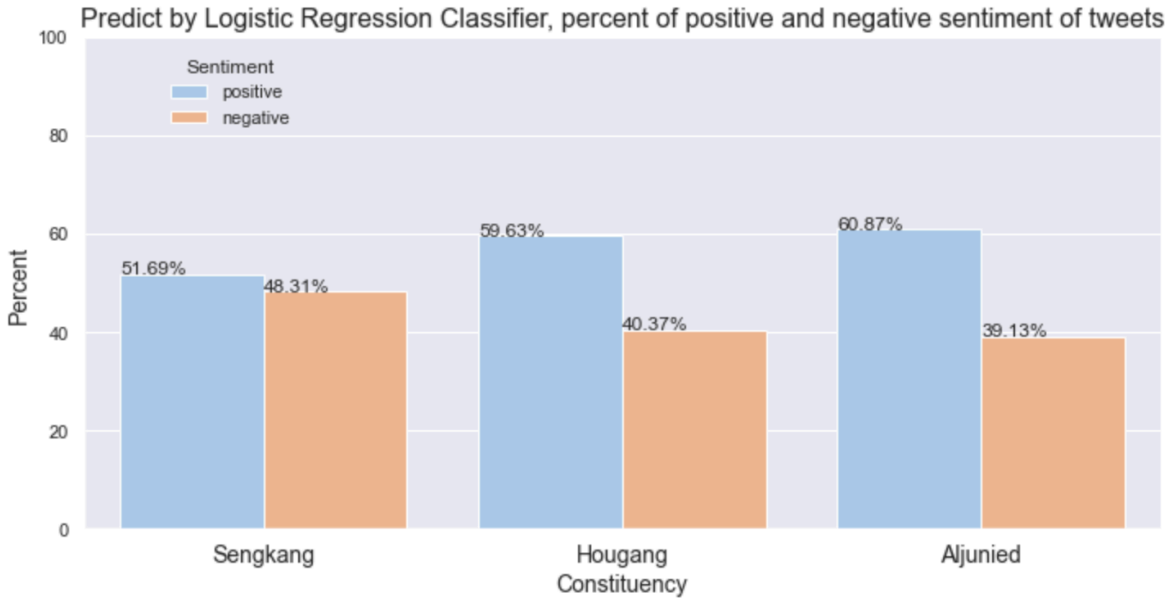


Figure 21. Predict by Logistic Regression Classifier, percent of positive and negative sentiment of tweets for each constituency

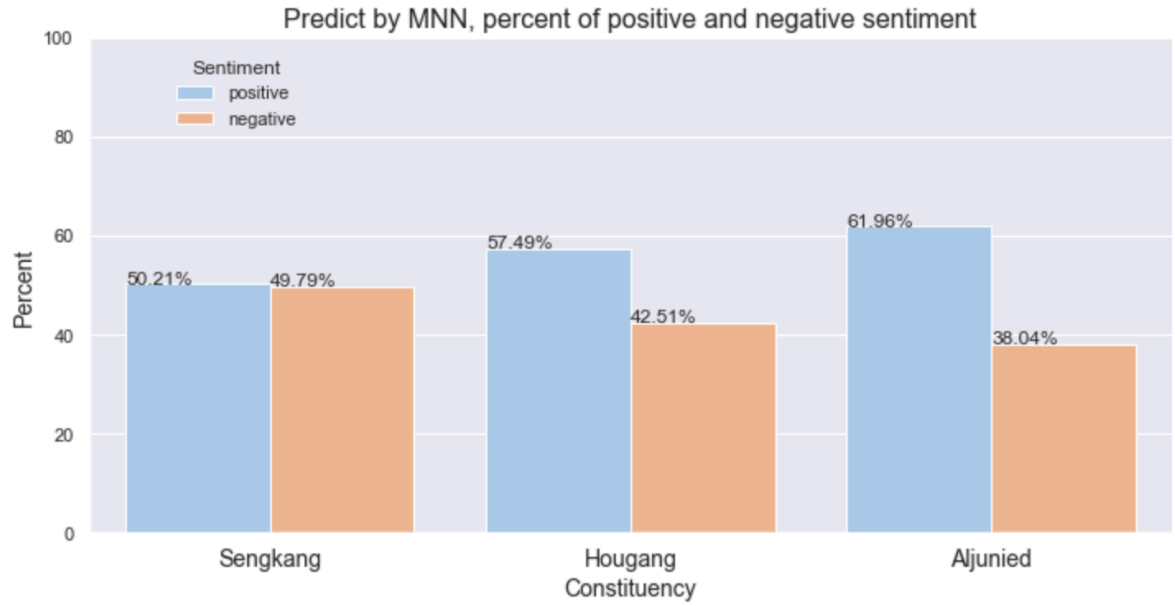


Figure 22. Predict by MNN, percent of positive and negative sentiment of tweets for each constituency

Sengkang has quite balanced sentiment, and both Hougang and Aljunied are more towards to positive. It is partially because Sengkang is only won by WP this year, whereas Hougang and Aljunied have been held by WP since 1991 and 2011 respectively.

Additionally, it would be interested to find what the common positive and negative words are when users mention the three constituencies.

Sengkang positive words

Sengkang negative words

Figure 23. Wordcloud of frequent positive and negative words when talking about Sengkang on Twitter

Hougang positive words

Hougang negative words

Figure 24. Wordcloud of frequent positive and negative words when talking about Hougang on Twitter

Aljunied positive words

Aljunied negative words

Figure 25. Wordcloud of frequent positive and negative words when talking about Aljunied on Twitter

It is also strikingly found that users use Noun much more frequently, which accounts for 54.6% of the count for all tags (Figure 26).

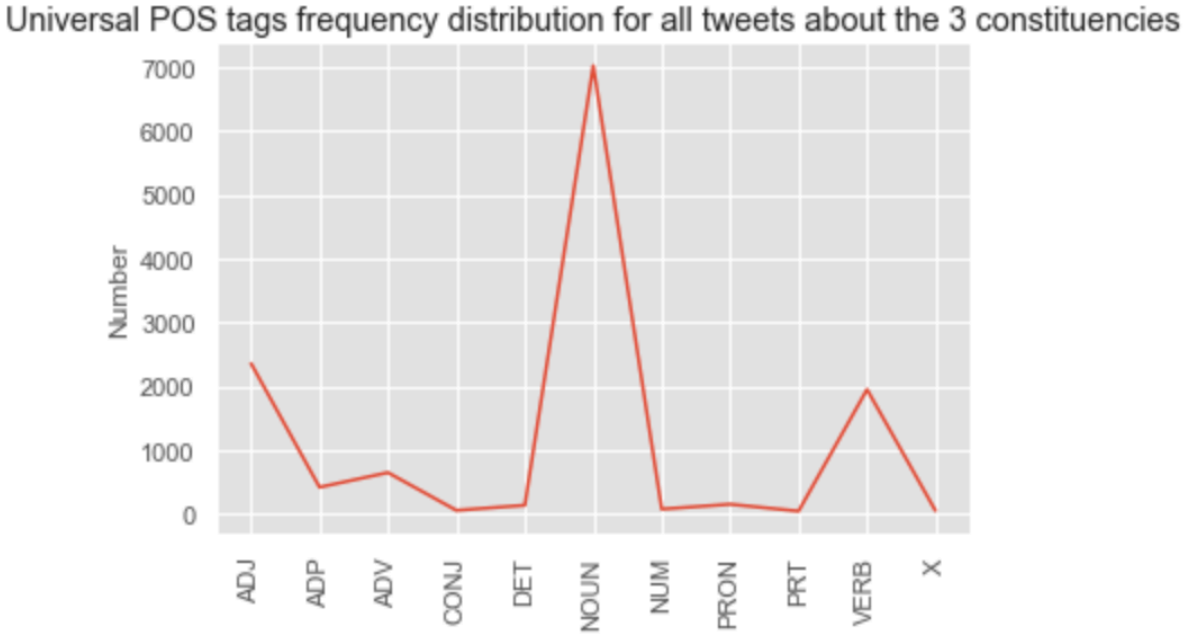


Figure 26. Universal POS tags frequency distribution for all tweets about the 3 constituencies

5. Conclusion

Through this project, tweets talking about the three Constituencies (Sengkang, Hougang and Aljunied) won by the Workers' Party are crawled via Tweepy. Total 6

classification models, namely, Multilayer Neural Networks (MNN), Convolutional Neural Networks (CNN), Multinomial Naive Bayes, Logistic Regression, Decision Tree and AdaBoost, are trained using the sentiment140 dataset. MNN, CNN and Logistic Regression are further tuned to find their optimal hyperparameters. From the sentiment prediction results, it is found that Sengkang has quite balanced sentiment, and both Hougang and Aljunied are more towards to positive. It is partially because Sengkang, as a newly created GRC with many young families, is only won by WP this year, whereas Hougang and Aljunied have been held by WP since 1991 and 2011 respectively. By doing this project, we employed web crawling, text pre-processing, natural language processing, data visualization techniques, and more importantly on training and optimizing multiple machine learning classification models.

6. Bibliography

ChannelNewsAsia. (2020). *GE2020 results*. Retrieved from https://www.channelnewsasia.com/news/specialreports/sg-votes/results

StackOverFlow. (2020). Retrieved from https://stackoverflow.com/questions/64030528/getoldtweets3-an-error-occured-during-an-http-request-http-error-404-not-fou

Mothership. (2020). Retrieved from https://mothership.sg/2020/07/sengkang-young-singaporeans/

Na, J. (2020). *3_5_Classifying_Yelp_Review_Sentiment.ipynb.*

Scikit-learn. (2020). Retrieved from https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

Kim, R. (2018). Retrieved October, 2020, from Another Twitter sentiment analysis with Python — Part 11 (CNN + Word2Vec): https://towardsdatascience.com/another-twitter-sentiment-analysis-with-python-part-11-cnn-word2vec-41f5e28eda74

Kaggle. (2017). Retrieved October, 2020, from https://www.kaggle.com/kazanova/sentiment140

Gorman, B. (2017). *A Kaggle Master Explains Gradient Boosting*. Retrieved October, 2020, from Available: http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting/

Elections Department Singapore. (2020). Retrieved October, 2020, from https://www.eld.gov.sg/