# COMP30027 Report

**anonymous**

## 1 Introduction

This project is about to pick suitable machine learning strategy to predict the geotag based on the text generated by user. In this project, I will discuss my work from the following perspective: data pre-precessing[3], Models[5] that I explored.

## 2 Related Work

Previous works on

## 3 Data Preprocessing

During the project, I found that it is really important to have a good approach to transfer the plain text to useful and suitable new features to feed into the model. Better data preprocessing strategy always produce more usable information and more choice for our later feature engineering.

### 3.1 Why Not *-top* Files

Those files were originally provided to us. Even though those words are highly correlated (in terms of all the words) to our labels, up to 78% of the instances contain only 0 for all the attributes. If I could correctly predict over 70% of the non-zero labels data and use 0-R to assign the others to one label, 34.9% accuracy could be reached overall. However, based on my discovery, it's even impossible to achieve 40% on the non-zero part. Also, the real world data wouldn't be evenly distributed. In this case, the data could not be very useful.

### 3.2 Raw File Text Precessing

Another choice becomes to use the raw tweets and process it to usable data. However, plain text can be hardly used but words are easier. Therefore, the basic step is to split the text by space. Next, there are still a lot of noise and redundancy in the feature space. The primary thing to do is to remove all the punctuation which are widely used but uninformative. After that, by sampling some text from the tweets, I notice that the word after # and might be useful as they may contains information which hight correlated to one of the label but rarely occur, for instance, ″BrisbaneNews9″. Those words are split into smaller words by capital letter. Notice that I might get a text like this ″#LouisWhyAreYouAnEGG″, the EGG at the end won't become [″E″,″G″, ″G″]. Instead, it will be kept as a whole word. After this process, all words are transfer to lowercase to prevent mismatching.

Up until now, the sample size are huge, throwing some less informative words is necessary. In this steps, all [1] stop words and unicode are removed. Those words are widely used but less informative. Generally, you couldn't say that people lives in Brisbane use more ″to″ than the people who lives in Melbourne. Too many of this kind features could results in overfitting.

## 4 Feature Engineering

Feature engineering is another very important part. Based on the performance of my preprocessing strategy, same set of data gain a 5% [2] boost in accuracy when comparing to using train-top100. In this section, I will discuss two feature engineering approach that I used. Their performance will be mentioned in model section. [5]

### 4.1 Frequency Based

Since the feature space are still very large, but number of instance are relative small. To avoid overfitting and variance of our model, I prefer to take the top 20000 most frequent words

[1]more will be removed in later section for testing purpose

[2]figure obtained by 10-fold CV on train-raw and dev-raw by using MNB and Logistic regression(Saga)

among all tweets. Two difference representations for this method are used, the first one is one-hot (preferred by most of the model) stored in sparse matrices, the other one use the number of ranking in frequency (start from 1) and then use 0 padding to extend to the same length (preferred by word embedding).

### 4.2 TF-IDF

TF-IDF is a commonly used feature selection tools in NLP. The TF part measure how frequent we could see a word in a document, IDF represent the incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely.(Wikipedia contributors (2019)) I process the text both in word level (with n-gram range 1-2) and char level (with n-gram range 2-6 for word to char). The feature only using word level features we call it **WORD**, and the feature combine both word level and char level features we call it **CHAR**. However, using char level features does not give us a better results but require a lot more precessing time, therefore, it was discard eventually.

## 5 Models

In this project, a wild range of different models was considered. However, none of them are used alone. Instead, a ensemble model was built on the top of them to combine the advantage of them.

Table1[1] contains part of the model I used. All of them are fine-tuned on cross-validation and eventually tested on dev set. If we only test our data on dev set, we are still trying to learn the hyper-parameters from dev set which will cause over-estimate.

| Model | WORD[4.2] | CHAR[4.2] |
|---|---|---|
| MNB | 34.3% | 34.7% |
| Logistic Regression | 33.5% | 33.4% |
| Random Forest | 33.2% | 33.1% |
| adaBoost[3] | 34.6% | 34.0% |

Table 1: Comparison of accuracy for different models

### 5.1 Multinomial Naive Bayes (sklearn)

- NB is normally the first approach people want to try when doing a NLP task. Also, It could handle multi-class problems naturally and yield some decent result. Since we know this is a multi-class prediction and the data distribution follows a multinomial distribution, we chose the MNB classifier.
- By looking at the confusion matrix, it works well on Brisbane (38%) and Sydney (36%) , but not very well on Perth(28%).

### 5.2 Logistic Regression (sklearn)

- sag/saga solver works pretty well on multi-class problems and perform fast convergence in problem with huge data set. Also, better interpretability than MLP.
- The confusion matrix looks quite similar to MNB. However, it perform really well on predicting Sydney(43%) by sacrificed the performance on Brisbane(30%) and Perth(27%). I guess that it was caused by using OVA to handle the multi-class problem.

### 5.3 Random Forest (sklearn)

- Ensemble learning normally works well and it could handle multi-class problem naturally. In Random Forest, the variance of the model could be minimized.
- Due to the randomness nature of it, error analysis could hardly been applied unless random seed is fixed. With fixed seed, It could achieve a slightly lower accuracy than MNB, whereas, a much better F1 score. In order to keep the randomness nature, I removed the hard coded seed values in later use.

### 5.4 Word Embedding (Tensorflow)

- One-hot encoding was used to transfer the text to embedding-layer-usable data to feed in a random initialized embedding layer. A Bi-direction LSTM layer was used directly blow the embedding layer for text analysis. [4] A bi-direction is a common tool for NLP task, it will look at the text both forward and backward which give the model extra chances to look at the text and against forgetting effect. However, overfitting is a big problem. Therefore, multiple dropout layers are applied.
- The result are not supper impressive, it only achieved 34.3% accuracy on dev set

[4]high level understanding of LSTM could found here Kaushik Mani (2017)

and pretty low F1 score. However, a great boost acc on Perth, which might caused by using word count features.

### 5.5 Voting (sklearn)

- Voting is a robust learning method, works well on combining multiple weak learner. Softmax decision function was used in this project to make a decision based on the how confidence (represent in probability) is the model think their predictions are correct.
- As we already had some trained model, we could just combine them and see if they yield some useful results. The final result shows that it does pretty well. MNB, Logistic Regression, Random Forest are combined, and Adaboost with one layer decision tree and with MNB are used for my voting approach, and yield a results which better than any single of them.

## 6 Conclusions

In conclusion, I found out that the most important part of this project is feature engineering. It is true that some model could have a very good pattern recognition ability, but bad preprocessing may mislead the model and make the problem non-learnable, or require extremely long time to converge. By using the same processed data, the difference between best model and worst model are smaller than 3% (compare to itself not 100%). Intuitively, better pre-processing method could extract more useful information from the raw data. Consequently, models could gain huge advantages by using this ”extra” information.

## References

Kaushik Mani. 2017. GUR’s and LSTM’s. `https://towardsdatascience.com/grus-and-lstm-s-741709a9b9b1`. [Online; accessed 19-May-2019].

Wikipedia contributors. 2019. Tf–idf — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Tf%E2%80%93idf&oldid=894112456`. [Online; accessed 18-May-2019].