1. Introduction

The task of project 2 is to, based on the content of a tweet only, predict the geolocation from one of the four cities: Melbourne, Brisbane, Perth and Sydney. It is a classification problem with long string text as input (each tweet is a string with less than 140 characters). In the training set, each class has 25841 instances(tweets), making a total of 103364 instances.

2. Data Cleaning

To ensure that the tweets in the training data is useful and can be further analysed and processed for model training, noisy data such as meaningless and useless texts need to be removed. UTF-16 strings (e.g. '\u0e44'), which exists in the raw training data, are non-understandable strings incorrectly converted from their original and meaningful form to their current form when they are collected from Twitter and stored in tsv file, probably because of the inconsistency between different encoding and decoding systems. UTF-16 strings in the training set can be regard as data collecting 'errors'. And for a human, they are garbled characters. What we should have in the training set are strings which are original, useful and human-understandable, not those errors or garbled characters (i.e. UTF-16 strings). One possible solution is to convert UTF-16 strings back to their 'proper' form. This is not ideal though: some UTF-16 strings, if encoding/decoding properly, represent text in other languages such as Arabic (e.g. the 12th instance). As will be mention later in the pre-processing section, we converted the training tweets into a more useful format using a particular method under the assumption that tweets are in English. Converting non-English words in the same way is at the risk of generating more noisy or useless and even misleading data. Thus, even when UTF-16 strings can be transformed back to its 'proper' form, we chose to delete them instead.

After filtering UTF-16 strings out, excessive quotation marks and whitespaces in tweets are removed as well.

3. Data Preprocessing

Another issue with the training data is that it can not be directly used without pre-processing. In the train-raw.tsv file provided for project 2, each instance is just a string text directly extracted from Twitter. There is no way that we can directly feed text string to our model since each string is unique. Therefore using the whole string as the input to feed the model will give the model only one feature where every instance is unique(except for identical tweets). The solution to deal with this problem is using CountVectorizer from sklearn library. We firstly fit the training set using CountVectorizer to find all words appeared in the whole training set and construct a pool of words (i.e. tokens). The pool contains every word which appeared in the training set. Then, we transform each tweet in the training set(also those in the dev set and test set for the purpose of consistency) into its corresponding token counts(i.e. counts how many times each token(word) appears in a tweet).Instead of a unique, long string text(catastrophic for machine learning as an instance since this instance would have only one feature). By using CountVectorizer, each instance now is represented by a row of values of different features(ideal form for machine learning) and can be directly fed to machine learning models.

4. Feature Selection

4.1 Motivation

Due to the huge amount of string texts in the training set, the number of tokens in the pool CountVectorizer generated is also significant (in our case, there are 162860 words identified and therefore 162860 features for each instance). The transformed training set(162860-dimension) will almost certainly face the curse of dimensionality. As the dimension increases, the volume of the space increases so fast that data become sparse. Sparse data will lead to increase of bias and variance and consequently lower accuracy for classifiers. Also, too many features will expand the volume of computation needed to train a model. The model that suffers most severely from the curse of dimensionality is the K-nearest-neighbour classifier whose accuracy is expected to be pretty low. It can not even be trained in a reasonable amount of time.

Therefore, there seems to be an urgent need to reduce the number of features used in the training set. SelectKBest method from sklearn is used to find the top K words with highest chi-square scores. In other words, features (i.e. occurrence count of words) which give most information about which class an instance belongs to will be fed to classifiers.

4.2 Error Analysis

Surprisingly, this method does not work as expected. Unlike most other training datasets with large amount of attributes, choosing top k words for training dataset for project2 can even reduce the accuracy of classifiers. The reason why this method 'behaves' differently might be that attributes in the training set of project2 (i.e. occurrences of all words in the pool of words) are different from attributes of most other training datasets. For a more common training dataset with many features (e.g. those we have seen in practice), different features describe or capture different aspect of an instance. For example, when predicting the housing price using a training set, the features in that training set could contain both useful features(e.g. location , age, number of rooms, etc.) and useless features (e.g. names of pets living in that house). Note that all these features are different aspects of the house. By choosing k best features, the dataset can keep the most useful features and get rid of useless ones. By contrast, in training dataset for project 2, the attributes are occurrences of different English words. Even if English words are different, all attributes describe the same aspect of the tweet-occurrence (s of different English words). If we think collectively, occurrences of all words in the pool of words, can be regard as the meaning of the tweet (it is represented not in human's but in computer's way). Therefore, if we were to choose only top k best attributes('words'), the single aspect (occurrences , or meaning of tweet) of the instance(tweet) is not fully represented. If we think classifiers as a person, then someone who knows only 'k best words' of a particular language will find it difficult to understand that language (he/she can still guess though). Thus, to help classifiers know as much as possible about the ’language’(not guessing), all features (occurrences count of all words) should be kept and therefore SelectKBest is abandoned after several trials.

This idea might still be fairly abstract, Another perspective may help to understand. If we regard those features we generated from CountVectorizer as sub-features of the only feature-the tweet text. SelectKBest can choose the sub-features with highest ‘weight’, but we are not supposed to remove those with less weight since the only meta-feature (tweet text) will then not be fully represented.

5. Building Models

There are two stages for building our models/systems. In the first stage, we explored various base classifiers using codes imported from sklearn library trained with our training set. All models are trained using the training set generated in the way discussed above and then tested on dev set. For each type of classifier, RandomSearch or GridSearch method is used to find the best parameters. As a result, MultinomialNB stands out among those classifiers.

We also tried different K values to choose the top k words before we realised that SelectKBest does not work as expected. What we found is that as K increases, the accuracy of each model also increases. The best K is 162860-the total number of attributes(i.e. choose the all atrributes).

In the second stage of building our model. We have tried stacking and voting ensemble methods. We tried different combinations of models. However, there seems no combination of ensemble model meet our expectation of increase the accuracy by a considerable margin. It is even more surprising when we find out that no combination of ensemble model can beat the MultinomialNB. Although ensemble methods do not increase the accuracy in our specific case, they are theoretically used to boost accuracy in machine learning and expected to have better accuracy than any of the metamodels, therefore they are still worth exploring.

Thus, the three models we think worth analysing are one base classifier: MultinomialNB and two ensemble classifiers: stacking model and voting.

6. Model Performance Analysis and Summary of Related Literatures

As a surprising result, stacked ensemble model and voting seem to have much lower performance(accuracy with dev datasets) than expected. Since ensemble models contain the prediction results of multiple learning models, they are commonly used to boost accuracy. In our specific application of stacking, the poor performance and the lack of diversity of the models might be the reason for low accuracy according to Dietterich (2000)'s emphasis, "a necessary and sufficient condition for an ensemble model to be more accurate than any of its individual members is if the classifiers are accurate and diverse". Based on our result on test accuracy: all of our five base classifiers have poor performance. Funda Güneş, Russ Wolfinger, and Pei-Yi Tan state that overfitting and leakage are also common and universal concerns in stacked ensemble models(2017). As the number of models increases, the trained model may largely bias on the training data and give over optimistic results. Another challenge for stacked ensemble models in practice is hyperparameter tuning. Since there are a set of hyperparameters for each base learners, the number of possible stacked models is infinite(Güneş, Wolfinger, and Tan, 2017).

Our best training model in this project uses multinomial naïve Bayes method. The multinomial model records word frequency in each document and predict the location, i.e. the class, based on the frequency(McCallum and Nigam, 1998). Our model with multinomial method from sklearn package gains the highest accuracy over all the models attempted even when the twitter length is not taken into consideration. Although McCallum and Nigam assert that "including document length is necessary because document length specifies the number of draws from the multinomial"(1998). Naïve Bayes often produces surprisingly good results(Friedman 1997; Friedman et al. 1997; Sahami 1996; Langley et al. 1992), and a multinomial naïve Bayes suits our project the most and brings us relatively satisfying accuracy.

Therefore, the best system we built is the Multinominal NB model due to the fact that it does not only can be trained in a short period of time, but also has higher accuracy while testing with dev dataset.

7. Parameter Tuning

This is a time-consuming processing due to the size of datasets. During the process of parameter tuning, we have found that the time required to train different models differs significantly. Given the large number of attributes in each instance and the large number of instances in the training set, the data size is so enormous that we have to put the time required to train a model at priority. Therefore, the time required to train the data with the model should also be considered as an important part of the evaluation of the model, as well as accuracy and other metrics.

The fastest model to train with is MultinomialNB model, which takes around only 10 seconds to train. While the slowest model should be KNN model which takes a ridiculous time to finish even one training(therefore we could not train the data and cannot provide a csv predicted file in submission).

8. Conclusion

In conclusion, the three main models we selected are multinomial Naïve Bayes, stacking and voting ensemble models. Our best model for this project is multinomial Naïve Bayes, with the accuracy of 34.6%. It is not an ideal accuracy for machine learning because our training set does not have enough indicative features for the class prediction.

Even though our best model's accuracy increased from that of the baseline classifier (one-r classifier) for almost 10 percent, there is still much room for improvement. One of the possibilities is to generate more features, for example, we can count the length of each tweet. It could enhance the performance of Multinomial NB because it determines the number of draws from the multinomial. Cross validation and bagging may help improve the ensemble models, theoretically.

References

Dietterich, T. 2000. Ensemble Methods in Machine Learning. Lecture Notes in Computer Science 1857:1–15.

Güneş, F., Wolfinger, R., & Tan, P. Y. 2017. Stacked ensemble models for improved prediction accuracy. In SAS Conference Proceedings.

McCallum, A., & Nigam, K. 1998, July. A comparison of event models for naive bayes text classification. In AAAI-98 workshop on learning for text categorization (Vol. 752, No. 1, pp. 41-48).

Nir Friedman, Dan Geiger, and Moises Goldszmidt. 1997. Bayesian network classifiers. Machine Learning, 29:131– 163.

Jerome H. Friedman. 1997. On bias, variance, 0/1 - loss, and the curse-of-dimensionality. Data Mining and Knowledge Discovery, 1:55–77.

Mehran Sahami. 1996. Learning limited dependence Bayesian classifiers. In KDD-96: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, pages 335–338. AAAI Press.

Pat Langley, Wayne Iba, and Kevin Thompson. 1992. An analysis of Bayesian classifiers. In AAAI-92.