COMP90051 project1: Who Tweeted That?

Yaoyi Chen 970325 yaoyic@student.unimelb.edu.au
Yibo Wang 922200 yibow5@student.unimelb.edu.au
Yuxuanchen 1035457 yuxuchen@ student.unimelb.edu.au

1. Introduction

In recent years, with the development of artificial intelligence, text classification technology based on machine learning has received extensive attention and development. This technique implements the classification of test texts through five steps: text preprocessing, feature extraction, dimensionality reduction, classification, and model evaluation. The goal of this project is to extract and classify the tweets with author information on Twitter, use the terms in the tweet text as feature items, and select the best representative features from this. The dimension of the feature space, and the training model based on the extracted features which can finally achieve accurate classification of the tweets without author information.

2. Data Set

The data set used for this report is short messages that posed to Twitter. The data set consist of two parts which are train_tweets.zip and test_tweets_unlabeled.txt. The training data set contains 328k tweets containing author IDs. The test data set are more than 35k of tweets published by the same author but not labeled with ID information. However, in the training set, on average, each user sent 34 tweets, but in fact, nearly half of the users sent less than 17 tweets.

2. Related Work

2.1 Feature Engineering (version 1)

We use the TF-IDF method for feature extraction and get a feature data set of 230K dimensions. Due to the large dimension of the feature dataset, we consider data cleaning from the following four aspects.

Id	Original URL	Extracted URL
8746	http://bit.ly/1UstFu	bit.ly
8746	http://bit.ly/Jplists	bit.ly
8746	http://tinyurl.com/yks92ww	tinyurl.com
8746	http://tinyurl.com/damuxk	tinyurl.com

Table 1 URL feature processing

Id	Original hashtag	Extracted hashtag
3039	#Cowboys	Cowboystag
8746	#anntaylor	anntaylortag
1017	#enewsnow	enewsnowtag
2764	#WhoDat	WhoDattag

Table 2 hashtag processing

- URL: We found that different users like to visit different websites, so we processed the URL and treated the results as a feature of the user classification.

The different pages of the same website in TF-IDF are treated as different features. For example, in Table 1, the two pages of “bit.ly” are treated as two different features, but in fact, we think that only domain name in one URL can be treated as a feature to effectively distinguish the characteristics of the user, other information has no value, so we extract the domain name and reduce the feature dimension from 230k to 130k.

- Hashtag: Some users like to post tweets with specific hashtags, so we think that the hashtag's topic information can be used as a feature to distinguish users.

In TF-IDF, "#" will be deleted, and the contents of the hashtag will be changed into ordinary text information. In contrast, we want to highlight the information in the hashtag as a valid feature for distinguishing users, so we add the suffix "tag" to the hashtag text. For example, in Table 2, turn "cowboys" into "cowboystag".

- Long word: Considering the different writing habits of users, some users prefer to use long and complex words in tweets, while others prefer to use abbreviations or short words. We have processed the data for this situation.

By extracting the number of long words in each tweet message as a feature, and splitting the 10k messages into a training set and a test set in a ratio of 9:1, the SVM algorithm is used to calculate the accuracy, and the result is shown in Figure 1. The “longword (7)” indicates that the number of letters is greater than or equal to 7 is a long word, and “baseline” indicates the state before this feature is added. As can be seen from the Figure 1, the feature of long words is not good for the accuracy of classification.

- Punctuation and emoji: Users has their own habit of using punctuation and emoji, so punctuation and emoji are extracted as features.

Take the same data set as the third part of the long word, and train different punctuation marks and emojis. The accuracy rate is shown in Figure 2. It can be seen that punctuation and emoji help with tweet message classification. (Sebastiani, 2002)

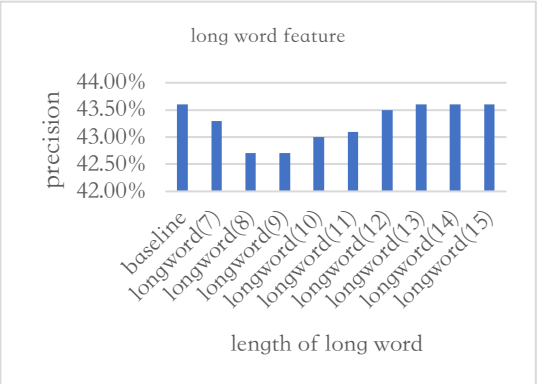

Figure 1 long word feature

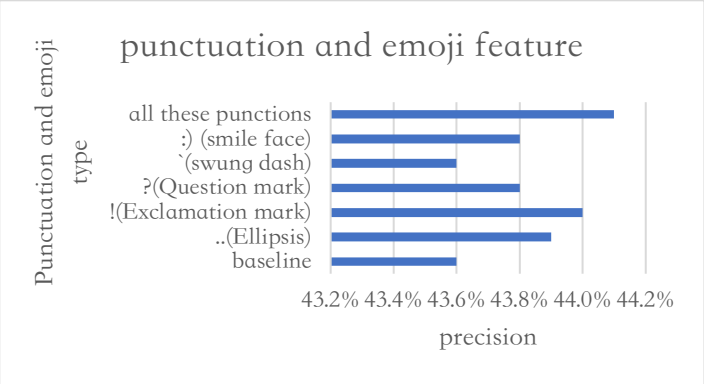

Figure 2 punctuation and emoji feature

2.2 Feature Engineering (version 2)

We firstly apply TF-IDF to the raw tweets and use KNN (with n_neighbor=1) algorithm to predict the result as a benchmark. The unlabeled accuracy result is 12%. Also we find that with KNN, the result is no different when the numbers of trained feature exceed 1200. So we use PCA on the tf-idf matrix to check the feature variances and find out there are nearly 200 features have relatively high variances (over 0.001). So we reduce the tf-idf matrix to 200 features and use that as the tf-idf document-to-vector matrix. We further apply Spacy pre-trained model and Bert pre-trained model on the tweets data. The unlabeled prediction results do shows improvement. With KNN, **Bert=18% > Spacy=15% > Tf-idf**. Therefore, we use the joint matrix of tf-idf (200 features), spacy, bert and apply some internal calculation to get another doc-vector matrix (3000 features in total) Hence we generate two different ML training set; version 1 has **130k features** and version 2 has **3k features**.

3. Model Selection and Analysis

We have tested several ML methods (all bellow percentages are perdition accuracy of unlabeled data set):

KNN: It is the most simple and fast model but generally gives pretty good results. The best result we can get is **22%**. We test several numbers of n_neighbor and find out the best is n_neighbor=1. This implies the doc-vectors of this tweet data are truly have a messy distribution, the distance of vectors with the same class is not close. The reasons why KNN had quite nice results is that KNN do not really 'train' and 'classify' labels, so it doesn't fit noise and does not 'force' the model to separate data points, this may be good for data set with huge number classes and messy distributed data points like this tweets data. But the capability of KNN has a clear bound in this project. The reasons of this may also due to KNN does not really train anything so it never gets any implicit information about data. Secondly, as the number of dimensions(features) scaling up, the Euclid distance becomes less significant. Furthermore, KNN suffers from imbalanced data (like this project) while many other ML learners such as SVM have the same problem.

SVM: This ML learner gives the best result **27%** (trained on 130k features matrix). We try different kernel methods. The kernel 'poly' and 'rbf' has too much computational complexity due to a large number of features (the dot calculation is too expansive). This two kernels, also kernel 'sigmoid', show a very low accuracy on training data. The 'One-vs-Rest' strategy has way better performance than 'One-vs-One'. So we finally choose 'linear' kernel and 'One-vs-Rest' method. We think the main reason why Linear SVM has the best result is that with careful feature engineering and selection, the more meaningful features are adding, the higher the accuracy achieved. Whereas Linear SVM also suffers from the imbalance of data. (Schuldt et al, 2004)

Decision Tree (DT): DT generally shows low accuracy in prediction (under **8%**). DT instinctually can handle features of various scales and sometime may even benefit from it. However, since we use doc-vector with pre-trained models (for 3k features matrix); basically the scales of all features are similar. The classes' distribution is disordered, the size of the total data set and feature space are pretty large. Whereas the size of samples for each class is quite small. Therefore, for most features, the information gain or impurity decrease in a split is very low. All these problems make the training complexity(time) extremely high(long). Even if we try to tune the hyperparameters like min_samples_split or min_samples_leaf to smooth the model, it

doesn't really help. If we just fix the structure, such as set max_depth, the results are generally very bad. But we believe we don't use DT well enough; the tree-based model can be a direction of our future improvement.

Random Forest (RF): RF has the same problems as DT. Since DT performs very high bias, RT is basically hard to improve this problem. If we want to reduce model variance, we may need the incredible number of estimators. Despite computational issue, it simply exceeds our RAM capability. We don't have the chance to test the whole data set on RF, but the result of a subset data is really unacceptable.

Adaboost: We use shallow trees (stump or depth 2) as base learners of Adaboost. It doesn't require that much RAM as RT, but the computational complexity is higher (time expensive). So, we have to limit the training iterations. While the results are not surprising; only under **3%**. Because we use shallow trees as base learners, the error rates are pretty high; often greater than 0.5. If we want to increase the base learners' accuracy, we may need more complex structures, then we just back to the same problem as RT.

LogisticRegression(LR): In LR, we want to test the difference between classification strategy 'OvR' and 'OvO' also the difference among internal solvers. The 'One-vs-One' strategy in this project seems too computational complicated. Data have the number of class n >9000 and it requires n*(n-1)/2 comparisons. However, 'OvO' is not efficient either, it tends to fit the noise more. For data with messy distribution, 'One-vs-Rest' can benefit more from tolerating relatively high loss to gain a better result. The test results in LR give us more confidence to use 'OvR' in our best Linear SVM model. As solvers, 'lbfgs' is too computationally expensive for our machine's capacity; 'liblinear' does pretty well while Stochastic Average Gradient descent('sag') does better and faster in such a large size data like this project. The best result we gain from LR is **16%**.

Neural Network (NN): We don't have the time and machine capability to formally test NN. But we believe NN may not really useful because for data set with huge feature space like 3k (not to mention 130k) and a large number of classes; the number of samples for each class in this data set then is too small. When we try CNN and LSTM, the loss is always high (over 6) and very hard to converge. The best result we can have is around **12%**. (Soon et al, 2001)

4. Future Work

In the feature engineering part, for version 1

emoji	:)	;)	:(	:D	:-)	:P	;-)	(:	"=D"	:-(
number	7640	1731	1583	1348	1278	589	575	331	254	180

Table3 emoji type and the corresponding number of occurrences in the training set

As shown in Table 3, different emoji features can be further tested, and then different features can be combined to try to achieve the best tweet message classification effect.

For version 2 matrix, we originally want to use the pre-trained models more flexible. For example, use advanced model, careful fine-tuning on pre-trained models, more complex matrix internal calculation, larger feature space and so on. But the limit of time and machine capacity restrict us to go there.

In the ML model part, the tree-based models may not actually beat SVM in this project but it should gain much better results than we had now. By using more meaningful and representative features, restriction of feature space, effective smoothing method like preparing and try other faster ensemble model like XGBoost to achieve this goal. On the other hand, NN may not be the best solution for this project, but it should perform better too. With better and smaller feature space, we think after sophisticated network design, such as more layers and different activation function combination, the LSTM may have the same nice result as SVM.

Reference

Schuldt, C., Laptev, I., & Caputo, B. (2004, August). Recognizing human actions: a local SVM approach. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* (Vol. 3, pp. 32-36). IEEE.

Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, *34*(1), 1-47.

Soon, W. M., Ng, H. T., & Lim, D. C. Y. (2001). A machine learning approach to coreference resolution of noun phrases. *Computational linguistics*, *27*(4), 521-544.