COMP90051 Assignment 1: Who Tweeted That?

Group 14: Amar Babuta, Asouv Bahl

1. Introduction

Authorship attribution is a common task in Natural Language Processing (NLP) applications, such as academic plagiarism detection and potential terrorist suspects identification on social media. As for the traditional author classification task, the training dataset usually includes the entire corpus of the author's published work, which contains a large number of examples of standard sentences that might reflect the writing style of the author. However, when it comes to the limited text on social media like Twitter, it brings some challenging problems, such as informal expressions, a huge number of labels, unbalanced dataset and extremely limited information related to identity. In this project, we attempt to find an appropriate solution for such a massively multiclass classification task from a Kaggle competition based on various machine learning techniques.

2. Feature Engineering

The summary of the training dataset and testing dataset are shown in Table 1.

	Number of Authors	Number of Tweets
Training Dataset	9297	328932
Testing Dataset	Unknown	35437

Table 1. The Training and Testing Dataset

First, we explore the distribution of the classes from the training dataset. The statistical boxplot and distribution of tweets numbers are shown in Figure 1 and Figure 2 below.

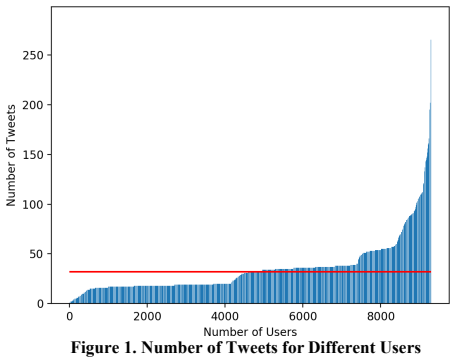

Figure 1. Number of Tweets for Different Users

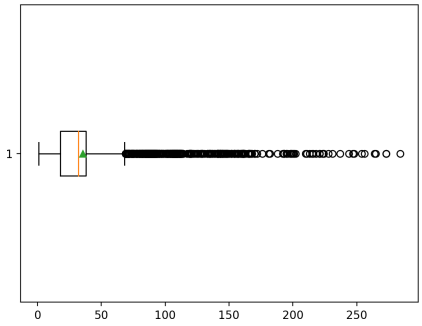

Figure 2. Boxplot for Number of Tweets

The median of the number of tweets from the same user is 32, which is the red line shown in Figure 1. The first quartile and third quartile are 18 and 38, respectively. It can be seen that this is an unbalanced massively multiclass dataset with a long-tailed distribution.

The preprocess for the original tweet text is applied on both training and testing dataset and we use the NLTK package to deal with it. First, we use regular expression to remove the non-English characters, including the punctuation marks and the emoji. Then all the alphabetical characters in the text are transformed into lowercases and the text are split into tokens for stemming and lemmatization. After that, we remove the stopwords with the built-in English-stopwords list in NLTK before recombine the processed tokens into sentences.

We use scikit-learn package to implement vectorization and TF-IDF algorithm. The processed text is transformed into vectors based on the frequency of tokens. Then we use the TfidfTransformer to give the token vectors with different weights based on the relevance between them, which would be helpful to extract some information related to the authors from the tokens. Finally, we get two matrices for both training and testing dataset. The columns are the TF-IDF weights of processed tokens and the rows represent the individual tweets.

3. Learners

We select four typical machine models for multi-classification in this task, including the MultinomialNB (Multinomial Naïve Bayes), KNN (K Nearest Neighbors), LinearSVC (Linear Support Vector Classification) and MLR (Multiple Logistic Regression). However, due to the limitation of data and computing resources, we did not try the DeepLearning such as CNN and BERT even they have proven to perform well on massively multiclass classification task.

The results are shown in Table 2 below:

MultinomialNB	KNN(K=1)	KNN(K=2)	KNN(K=3)	LinearSVC	MLR
20.534 %	15.012 %	9.267 %	8.758 %	30.128 %	17.100 %

Table 2. The Accuracy Results for Different Models

4. Critical Analysis and Model Selection

It can be seen that the linearSVC achieves the best performance with 30.128% while the KNN(K=3) has the lowest accuracy of only 8.758 %.

Reasons for better result of Linear-SVC is that it has a higher fault tolerance and regularization effects. Based on the distribution of training samples, Linear-SVC searches for the best linear classifiers. The samples that determines the classification boundary position are not all training data but two different samples of two categories that make the biggest interval between their categories space. Therefore, it is possible to screen a small number of training samples that are most effective for predictive tasks in massive or even high-dimensional data. The generalization ability of linear-SVM is mainly reflected in the modeling of structural risk when modeling, which means the model will reduce over-fitting after adding regularity.

In this experiment, KNN performs not good for all the three K values. Moreover, with K increasing, the performance tends to be worse. This is because the imbalanced distribution of classes in training set and KNN is greatly affected by those categories with extremely limited number of samples. A lot of classes in our training set only contains one or two samples, therefore, in the classification process for a new sample, if $K > 1$ and the K similar samples of this sample belong to different categories, the deviation of the classification effect will be increased, and the more the categories of these K samples, the more likely the error occurs.

As for the result of MultinomialNB (20.534 %), if the possibilities of each feature are assumed to be independent, then MultinomialNB is expected to have a good performance. However, it is not the real situation in most NLP tasks since there are some potential relationship between different words in sentences. Also, in our training set, it is not appropriate to assume that features are independent of each other, because there may be interaction between Twitter users, resulting in similar data of different users. Meanwhile, most prior distributions are assumed by experience. Therefore, Naive Bayesian is more adjusted to incremental training of smaller scale of data, which could be done by separating data into batches.

In the case of LMR, since the logistic regression model take all training samples into consideration of the parameters during the training process, if there are too many abnormal points, it will have a greater impact on the decision boundary, and finally the result will tend to be biased.

In addition, ensemble learning is a powerful technique to increase accuracy on a most of machine learning tasks. In this project, we try a simple ensemble approach called weighted voting to avoid overfitting and improve performance. The basic thought of this method is quite simple. For each prediction from the results of different models, we give them a weight corresponding to their individual accuracy in the previous stage. If the predicted labels of two models are the same, we just add their weight together. Then we select the prediction with highest weight as the final prediction. Considering the individual performance of the previous models, we try three different combinations: **linearSVC + MultinomialNB + KNN(K=1)**, **linearSVC + MultinomialNB + MLR** and **linearSVC + MultinomialNB + MLR + KNN(K=1)**. The results are shown in Table 3.

LinearSVC + MultinomialNB + KNN(K=1)	LinearSVC + MultinomialNB + MLR	LinearSVC + MultinomialNB + MLR + KNN(K=1).
30.100 %	30.100 %	21.898 %

Table 3. The Accuracy Results for Different Models

Unfortunately, it seems that the ensemble approach does not achieve our expectation and the performance is even a bit lower than that of individual LinearSVC. There are two main reasons. The first one is the limited number of selective models. Even in the combination with maximum models, we only have four models for voting. Another reason is that the individual performance for different models are varying widely, for example, the accuracy of KNN(K=1) (15.012 %) and MLR (17.100 %) are nearly only half as much as that of LinearSVC (30.128 %). The final result we submitted on Kaggle is based on the individual LinearSVC model.

5. Other Related Works

We also tried the Deep Neural Network LSTM. The entire training dataset is divided into 7:3 for training set and test set respectively. Because the training set is relatively small, we chose 5 epochs to train the model, but the final effect is still very poor. The first epoch to the fifth epoch only rose the accuracy from about 1% to about 6%. Considering that LSTM belongs to deep neural network, a relatively complex model, which contains a large number of parameters, requires a large amount of data for training, in order to ensure a good performance. However, because our training data set is too small, the neural network cannot be well trained, which leads to its unsatisfied performance. The results are shown in Table 4.

Epoch	Time	Loss	Accuracy
1/5	357s	8.3999	1.120 %
2/5	350s	7.7809	2.850 %
3/5	351s	7.4458	4.100 %
4/5	349s	7.1828	5.110 %
5/5	349s	6.9572	5.830 %
Total Loss	7.81	**Final Accuracy**	5.000 %

Table 4. The Results for LSTM

In summary, for such a massively multiclass classification with extremely limited text information, since the relatively small number of data sets, with relatively complex models such as deep neural networks, it cannot achieve good results. Consider applying the above model to real life. If the training sample is sufficient, the deep neural network will have better generalization ability, because the model contains a large number of features and has better memory effect. But if there are too few training samples, the performance of the neural network will be very poor. If the training samples are small and not balanced enough, and there are many categories, and the data is not completely linearly separable (such as this experiment), LinearSVC will have better performance.

6. Future Work

Due to the limitation of time, we have some ideas which might be worthy but have not yet to try. The first one is about how to deal with the unbalance problem of the dataset. Back to the Figure 1 and Figure 2, it can be seen that the tweets number for a lot of users does not even achieve the first quartile value which is 18. Massive users only post one or two tweets while some users have hundreds of tweets, which lead to an unfair situation for different classes in training. To soothe such case, a common approach, called oversampling, is to generate some data for the classes with extremely limited training data and the SMOTE algorithm is a popular method in this case. Figure 3 explains its basic thought briefly.

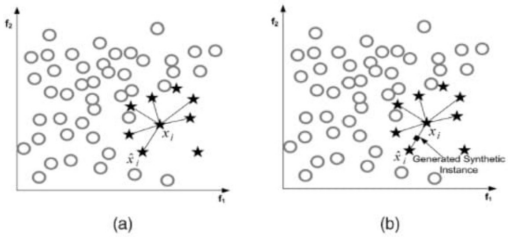

Figure 3. How SMOTE Works

Imaging that a data vector is just a point in a n-dimensional space. For each point from users whose number of tweets is less than the first quartile value, similar to KNN, we random choose some nearest points and link them to it with lines. Then we find the midpoints on those lines and take them with some noise as the generated data point for the class. However, how to determine the approximate number of generated data points is based on experimentation.

Another aspect is related to ensemble learning. One common approach is based on Bagging which is similar with the same notation in Random Forest algorithm. Bagging is to use the method of returning to sample, to build a sub-model with sampled samples, and to train the sub-model. This process is repeated several times and finally merged. Probably divided into two steps:

(1) Repeat K times:

- Repeated sampling modeling
- Training sub-models

(2) Models Ensembling

- Classification problem: voting
- Regression problem: average