Statistical Machine Learning - Assignment 1 Report
SML-Pro

1. Introduction

In this project, we identified three categories of features from two points of view. We compared these features with a baseline classifier and then used this most appropriate feature set to compare several classifiers. Besides the machine learning methods, we also experimented with a non-machine learning method as well as the hybrid method. For our final prediction, we chose linear SVM as the classifier with the combined features. The training accuracy achieved 29.48%, and the accuracy on Kaggle reached 31.93%.

2. Feature Engineering

Related researches indicated two effective aspects of identifying authorship. Rexha (2018) and Bozkurt (2007) both implemented content-agnostic and stylometric features to predict the author. Therefore in this project, by following their approaches, we considered both stylometric and contextual features. We used specific methods such as TF-IDF and Doc2Vec, to allow the machine to extract features itself for the contextual features, while we manually selected features for stylometry. We combined both kinds of features in our final feature selection.

To extract the semantic features, we deployed a sequence of canonicalization. First, we tokenized sentences by removing useless whitespaces and punctuation, while remaining and canonicalizing the multiple symbols to represent some user habits because some people particularly favored multiple periods and exclamation marks. We also lowercased words except those consisting of all upper characters, which are the representational features in informal writing. Besides, we eliminated the stop words and unnecessary tags such as “http.” Finally, we implemented lemmatization.

1.1 Stylometry

According to Rexha (2018), the stylistic distinction lies in features such as “average word length, average sentence length, average sentence char length, upper chars ratio, type-token ratio, and vocabulary richness.”

Given the characteristics of tweets, we combined some extra special stylistic features, calculate the number of 1) multiple punctuations like ‘!!!!!’; 2)special expressions such as ‘coooool’ or ‘BABABABA’; 3) total sentences in one tweet.

To select the useful features, we started from only implementing a single feature, then added more features step by step to filter features.

1.2 Contextual Features

TF-IDF

Before generating TF-IDF vectors, we also implemented N-gram to extract meaningful phrases. The first step we took was to include every feature that TF-DF generated. However, having more than 400 thousands of features was unrealistic to compute. Then we started from a small number of features by setting a threshold on the document frequency and gradually lowering this threshold. In this way, we added more features per iteration and compared different feature sets to get effective features. Similarly, we also set the threshold to specify the maximum number of features ranked by the normalized term frequency. At last, we combined length of tweets as stylometry feature with TF-IDF features, and we found that 200 thousand of features extracted by this combined method yielded the highest accuracy with several experiments.

Doc2Vec

Doc2Vec is a method based on Word2Vec, but the difference is that this method turns the paragraphs or documents into numeric representation for the features. For this model, we tried with the Distributed Bag of Words algorithm which the document vectors were gained from the result of trained neural network. The task of the neural network is to predict a probability distribution of words in a document with a given random word from the document. Hence the features are the inferred vectors from the DBOW model.

2. Learner

This multi-class text classification problem belongs to a supervised machine learning (ML) problem in which we have an author as the label for each tweet. However, compared with other multi-class problems, this task was more challenging in that there were more than 9k authors. Apart from that, it was hard to extract the representative pattern because of casual syntax. Also, we had to consider the limit of computational capability.

Given those challenges, there was some trade-off between features, models, and computational ability.

As we have conducted some preprocessing to shrink the size of features, to select appropriate features, we had to compare the three categories of features with a baseline classifier logistic regression. The results can be found in table 1:

Type of TF-IDF	Training Acc.	Kaggle Acc.
Doc2Vec	16.30%	0.047%
TF-IDF	16.1%	18.79%
Stylometry	5.9%	N/A
Combined(TF-IDF & Stylometry)	17.63%	19.04%

Table 1: Accuracy for different features with baseline

From the table, we found that combined features performed best. Then we decided to train these features with four conventional binary classifiers with default hyperparameters for comparison: Logistic Regression, SVM, Naïve Bayes, and Decision Trees. However, due to the high number of features, the Naive Bayes and Decision Trees require impractical RAM to compute. Because SVM and Logistic Regression require less computational resources while having higher accuracy, so we had to focus on SVM and Logistic Regression.

3. Model selection

By trying without regularization, we observed that there was overfitting for logistic regression. When we added more regularizations than default to the model, the accuracy is around 12.5%.
On the other hand, we tested SVM with linear, polynomial, and RBF kernels. The result showed that linear and RBF kernel achieved a close result which was much more accurate than the polynomial kernel. A good explanation of the close result was that with a large number of features, the features are already in high dimensions, so there was no need to map the data to a higher-dimensional space. The possible reason why polynomial performed poorly was that we did not find the right parameters for the polynomial kernel.
By comparing the accuracies, SVMs performed better than linear regression. Since linear and RBF kernel had similar performance, but the linear kernel was much faster, we ultimately decided to conduct with linear SVM. To explore the best performance of the linear SVM, we tuned the penalty parameter C to adjust the margin size. Below are the results.

Model	Acc. C=1e5	Acc. C=100	Acc. C=1	Acc. C=0.01	Acc. C=1e-5
Linear SVM	17.63%	25.23%	29.48%	24.92%	19.1%

Table 2: Training accuracies with tuned parameter C

4. Exploration with Non-ML Methods

4.1 Vector space model (VSM) with bm25 weights

Similar to the TF-IDF, we also implemented VSM by using BM25 weights and then constructed an inverted index list to predict the label. Each tweet is considered as a document, and the labels are the titles of the documents. By summing up the query score in each document and returning the ranking with the top highest BM25 scores, we can get the most similar tweet and use its label as the prediction result. Preprocessing steps were conducted and compared before constructing the VSM to get better performance. We did cross-validation by using 10% of the dataset as a test set and the other 90% to build the inverted index.
By applying this method, finally, we got 28.57% of accuracy on the cross-validation test set and 27.51% accuracy on the Kaggle set. The results are in Table 3:

Experiments	Training Accuracy	Kaggle Accuracy
Raw text with TweetTokenizer	23.74%	26.168%
Raw text with WordTokenizer	26.36%	26.375%
Remove stop word	26.49%	26.375%
Remove stop word & lowercase	28.57%	27.513%

Table 3: Appearance for Non-ML Method

4.2 Combination of VSM and ML

An interesting thing we have observed from our results is that the recall for top 10 is 34.87% and for top 15 is 38.82%, which suggests that if we have a good methodology and re-ranking this top 10 set in some way. We may essentially improve our performance of accuracy. Also, this is the motivation that we want to combine BM25 VSM with ML methods.
In the experiment, we combined the BM25 vector space model with linear SVM. The linear SVM would predict the result based on the tweets with labels set provided by BM25 inverted index. Again, we test our methods by cross-validation on both the training set and the Kaggle set. The results are listed in table 4:

Ranking set size	Dataset	BM25 Acc.	BM25 Recall	Final Acc with ML
Top 10	Training	28.5%	34.87%	31.62%
Top 10	Kaggle	-	-	26.61%
Top 15	Training	28.5%	38.82%	34.27%
Top 15	Kaggle	-	-	26.59%

Table 4: Appearance for the combined method

5. Critical analysis

5.1 features

We tried the stylistic features on different sizes of data set and plotted the corresponding accuracy. The chart indicated an anti-monotone relationship between the size of training examples and accuracy. This scenario was actually foreseeable, for the number of parameters was too small compared with training examples and the dataset was not linear separable under that low dimension.

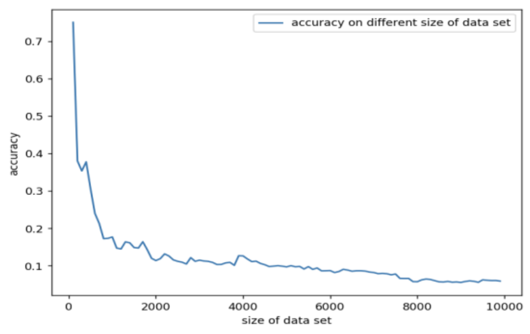


Figure 1: accuracy of different size of the data set

To overcome the problem of underfitting while still taking stylometry into account, we combined stylistic and contextual features in training models. The reasons why this combined method gives us the best appropriate features can be concluded in two aspects.

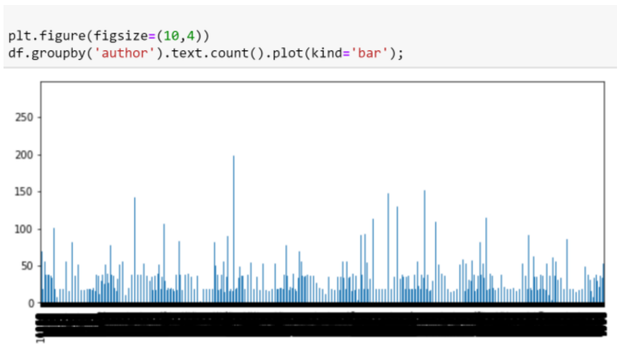
First, with TF-IDF, the uniqueness of features is guaranteed. When identifying authors for tweets, the goal is to find the uniqueness of the author for a given tweet. For example, if everyone type 'http' in their tweets, we cannot infer which author writes this tweet because this is nothing special. What helps us identify the author is the unique domain name. However, one big challenge is that the content is not fixed, and, with fixed length limitation, it is hard to identify a pattern of structures of the tweet. By using TF-IDF, even though the structures of tweets may not be so different from each other, we can get a large number of unique features as these features reflect how important they are for the tweets.

As for the second reason, the combination of stylometry and TF-IDF covers the feature diversity. For example, while TF-IDF handled the contextual part, the stylometry covered the structural part. Although there may not be many structural patterns as addressed above, using stylometry features such as the length of the tweet can still help. For instance, some people like short tweets, while others particularly like longer tweets.

On the other hand, although Doc2Vec belongs to a contextual perspective, the reason for its bad performance might be that the feature vectors are affected by the small single document size. For example, each tweet has 140 words limitation. For such a small corpus, the vectors learned from Doc2Vec are small, and the same words can be learned multiple times with different meanings. What's more, rare words are treated as noise in this model, so the preprocessing procedure should be redesigned in the future when using Doc2Vec. What's more, there is significant overfitting, and the penalty parameters should be carefully selected as well.

5.2 Model

From the results of models, SVM with linear kernel performed better than Logistic Regression. The reason could be that the logistic regression tried to find the best solution while the SVM tried to give an optimal solution. To be more specific, with the large margins, the SVM had lower variance than logistic regression which meant outliers did not affect the results much. The logistic loss function diverged faster than hinge loss, so it was more sensitive to outliers. From the results of logistic regression, we could find that the dataset was non-linear separable, which meant that more outliers were there for the models. Also, according to figure 2, the dataset was imbalanced, which caused the classifiers to be biased towards the majority classes. Thus, the low variance SVM did better than Logistic Regression in this project.



```
plt.figure(figsize=(10,4))
df.groupby('author').text.count().plot(kind='bar');
```

Figure 2: Number of Tweets grouped by authors

5.3 Exploration

From BM25 inverted index results, we observed that this method gave a good prediction result for the raw text even with less preprocessing. The performance of WordTokenizer was better than TweetTokenizer because the granularity of WordTokenizer was smaller than TweetTokenizer as it tried to split every punctuation, while TweetTokenizer tried to retain some special string tags like "#abc," and web URLs like "http://bit.ly/ssdasda". Usually, retaining the special string works because it keeps the important features as it should be. However, in BM25, the scores are only based on the string slot counts of appearance in the tweet. So with the granularity decreasing, the possibility that a specific string matches each other would increase, leading to the promotion of the final document score. For example, for the string "#abc", the TweetTokenizer just tokenizes as "#abc," while the WordTokenizer would separate "#" and "abc". If the author has the habit to hash someone, then the additional score of "#" will be considered in WordTokenizer. On the other hand, only "#abc" will be counted in TweetTokenizer. Furthermore, removing the stop words and lowercasing words can also improve the appearance of the system, because deleting these words can make us concentrate on the important words. The purpose of lowering the case of words is to prevent the same words from being counted differently since its format is in the count-based model.

We also found that when we combined BM25 with linear SVM, the accuracy was obviously improved in the training dataset but showed no improvements in the Kaggle set. As we cannot access to full Kaggle set, we don't know how exactly this happens. However, after communicating with Ben, we came up with the hypothesis that one of the differentiation reasons was because, in the Kaggle 30% test data, it may contain more authors whose tweet appear rarely in the training set. However, when we do cross-validation by randomly splitting the dataset into 90% training and 10% testing, authors who write tweets frequently could be more likely to appear in the test set. These tweets are much easier to recognize correctly due to their large data resource set.