

## I. Overview

For this Kaggle competition, I used TF-IDF plus Random Forest classifier as my model. This is a very basic model and can not get a good performance. It can not be accelerated by GPU, either. However, we can still learn and find out many interesting things from the experiments. Since my code explanation is included in the notebook, I am not going to explain implementation details again here, but focus on the reason and the results of my work. So, let's see the table below and have a quick overview first.

Preprocessing	Drop duplicates + Transform emoji to text
Feature Engineering	TF-IDF with N max features (N from 500 to 3000)
Training data size	20% to 50% according to experiments
Model	Random Forest
Best Result	Public 0.34839 / Private 0.33327
GPU Support	No

▲ Table 1. Overview of my work.

## II. Preprocessing

The most important part of preprocessing is that I transformed the emojis into text, such as 🤔 into "[cry]". Theoretically, this helps the model to learn the underlying logic in the content instead of just ignoring it. However, from my experiment results, the improvement of this data cleaning technique is approximately 1 to 2%. I believe this is due to the architecture of my work, since TF-IDF + Random Forest classifiers lack of the ability to make use of semantic contents. If I use BERT as the basic model, this data cleaning technique should give a better improvement.

## III. Feature Engineering

Using TF-IDF is a classical way to extract the features from text. Though it cannot grab the underlying semantic logic from the text, it shows high stability and high resistance to noise. For the Kaggle competition this time, I set max features from 500 to 3000 and test them respectively, and the result shows that with max features = 3000, the model gets great improvements compared to 500 and 1000. Moreover, the improvement does not seem to reach its limit. This implies if we have enough memory and time to build more features from the massive corpus, the performance can be further improved. However, due to the resource limits of Kaggle, I can only stop at 3000 and

try to improve other parts of my model.

## IV. Model and Training

I chose Random Forest classifier as my model. This model cannot be accelerated by GPU, so it takes a lot of time. According to my experiments, even with TF-IDF set to 500 max features only, the whole model still takes more than half an hour to generate the results. This gives me an inspiration of “training data sampling”. That is, since the training data is too large, I sampled part of the data used for training for reducing the resource consumption. These of course decreases my model performance, but if I keep using this model, this is a necessary evil.

From my experiment results, unsurprisingly, 20% of the sampling size gets the worst performance. 30% the second, and 50% is the best. The performance difference between the best and the worst is roughly 5%. I’ve also tried 100% of the training data, but it takes too long, so I just interrupted it and stopped at 50%.

This is not the end of training adjustment. I also used stratified split to split the dataset into training and validation. This is from my observation of the given dataset. The distribution of each class is actually imbalanced. By using stratified split, I can ensure that the distribution can be roughly good for my model training.

## V. LB and CV Strategies

LB (Leader Board) and CV (Cross Validation) are two phrases commonly mentioned in Kaggle competition. The main concept is to trust CV more than LB. This is because public LB only uses a small part of data to calculate the score. If we train the model to get higher LB score, our model will be biased toward public LB about 20% to 30%. Therefore, a better strategy is to do K-fold (or its variant) and train for a better CV score. This ensures the model performs good generally, even with private datasets. Though I did not do K-fold in this competition, this idea inspired me to do stratified split during training, which is mentioned in **Part IV**.

## VI. Interesting Findings

Before implementing the model, I observed the distribution and the features of the emotion dataset, and found out that about one-third of the dataset is emotion “joy”, and about one-sixth of the dataset is emotion “anticipation”. Therefore, I did some interesting experiments to check out the “private dataset”.







The first experiment is uploading a csv file such that each data is predicted as “joy”.

If the private dataset is also biased toward joy, I should get scores approximately 0.35. To my surprise, I only get 0.14525 in private dataset. This score is very close to one-seventh (0.142857), which gives me the second assumption: the private answers are all balanced.

To verify my second assumption, I wrote a code such that for each data, randomly pick one emotion as the predicted answer, and the private score is 0.13960, which is between one-eighth and one-seventh. This gives me some insight of the private data.

## Appendix:

### Snapshot of some of my submission results

 <b>submission (3).csv</b> Complete (after deadline) · 1h ago	0.13960	0.13802	<input type="checkbox"/>
 <b>submission (2).csv</b> Complete (after deadline) · 1h ago	0.14525	0.14188	<input type="checkbox"/>
 <b>submission (2).csv</b> Error (after deadline) · 1h ago			
 <b>submission (1).csv</b> Complete · 4d ago · Test	0.14525	0.14188	<input type="checkbox"/>
 <b>submission (1).csv</b> Error · 4d ago · Test			
 <b>submission.csv</b> Complete · 5d ago · Test Cleaning + 3000 Features	0.33327	0.34839	<input checked="" type="checkbox"/>