**Natural Language Processing**

**Text Classification**

**dc027884 Xie Yuan**

# Introduction

## What is it about?

**My project is mainly related to text classification in natural language processing. I implement a naïve-Bayes text classification model, which will preprocess the data, then perform word frequency statistics, feature extraction, word probability calculation, and finally classify the text .**

## What did I achieve?

1. **Preprocess the data in the json file (data format: there are n lists in a large list, these n lists represent text data, and the first item of each list represents the data number, and the second data represents the data category , the third data is text) that removes some useless data, such as "\n", "#" and other characters, and then divides the text data, converts all to lowercase and then performs stem extraction, and finally in the form of a list Store and write the final processed data into json files -train.preprocessed.json and test.preprocessed.json.**
2. **Perform word counting on the preprocessed text, and output the result to a text file: word\_count.txt.**
3. **Perform feature extraction on word\_count.txt, choose the most frequent 10,000 words as the feature words, and use the frequency obtained in word\_count.txt to calculate the total word frequency in each class.**
4. **using the Bayes rule to calculate the posterior probability of each feature word and the prior probability of the class and then output the result to the file: word\_probability.txt.**
5. **Implement the naive Bayes classifier to assign class labels to the documents in the test set, and store the file\_id and class label to 'classification\_result.txt'.**
6. **Use the F1 score to assess the performance of the implemented classification model, and output F1 score.**

## Programming Environment

**Programming language: python**

**Python version:3.11.2**

**code editor: visual studio code**

**Operating system: windows**

# Description of my project

## Function 1：def preprocess(inputfile,outputfile):

### Description

**Pass in a json file, the content of the file is as follows: (the list is nested in the list, the first item of each sublist is the text number, the second item is the text class, and the third item is the text content), after processing by this function, remove For the unused symbols in the third item, English letters are reserved and all lowercase, and then word segmentation and stem extraction are performed on them, and finally written into the output file.**

### Approach

**Use json.load (inputfile) to read the json file, then use NLTK.word\_tokenize() to tokenize the setence, then convert all characters to lowercase, then use nltk.PorterStemmer to stem the words, and then use json.dump(contents , outputfile) writes contents to outputfile.**

#### nltk.tokenize

**Import RegexpTokenizer from nltk.tokenize, and set the form of splitting the string in RegexpTokenizer, and then use it for characters.**

****

****

#### nltk.PorterStemmer

**Use this method for stemming**

****

#### json.load(inputfile) and json.dump(file\_contents,outputfile)

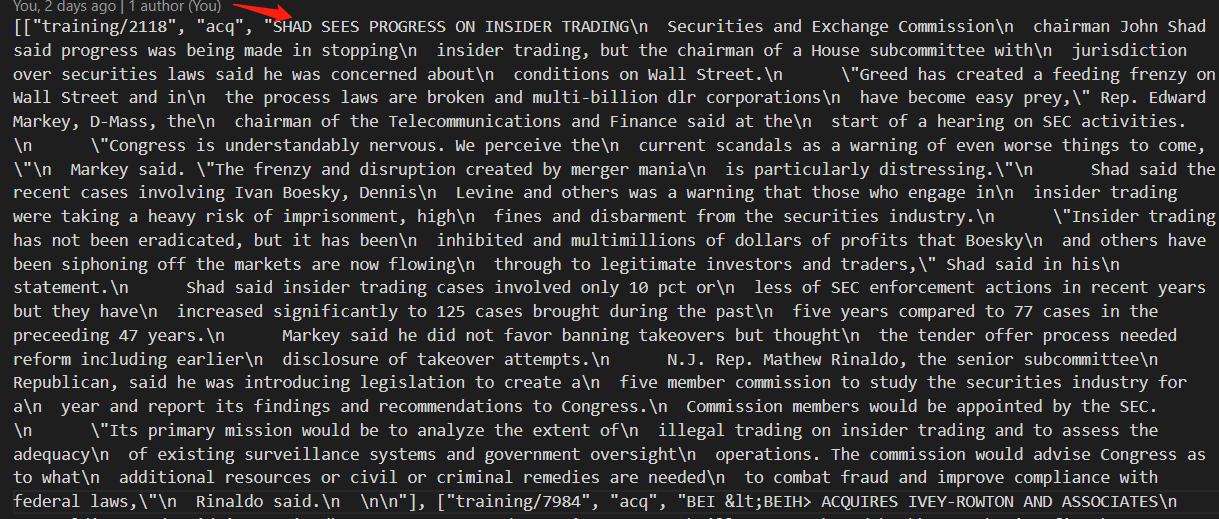
****

**Use json.load(inputfile) to read inputfile.**

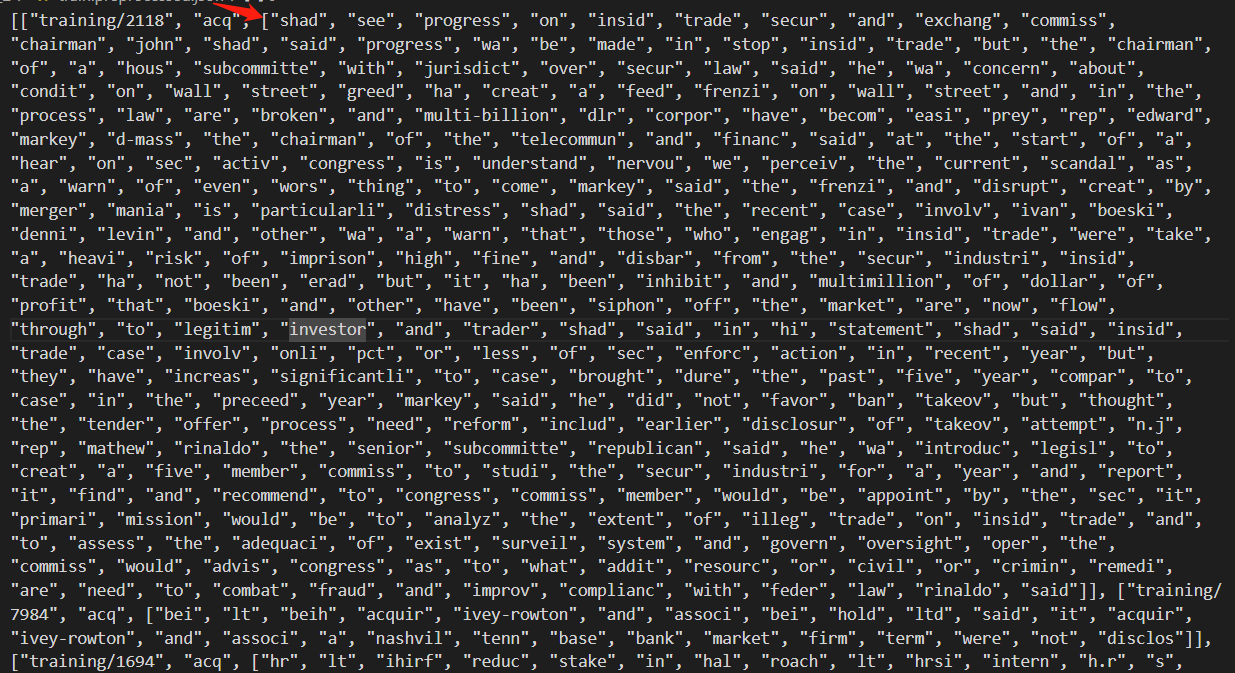
**Use json.dump(file\_contents, outputfile) to write the content file——contents into outputfile.**

### 2.1.3 Result

**Take a text content in a json file as an example:**

**Before processing：**

**After processing：**

****

## def count\_word(inputfile,outputfile):

### 2.2.1. Description

**This method uses a dictionary to count the words of each category and the number of each category of words in different categories, and output them to the outputfile in order.**

### 2.2.2. Approach（dictionary）

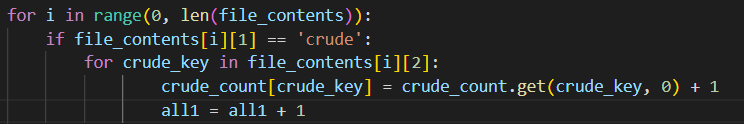
**Using the dictionary method, count the words that appear, The first line contains the total document frequency in five classes, in the order of: 'crude', 'grain', 'money-fx', 'acq' and 'earn', separated by a space. In the following lines, each line contains a word and its frequency in five classes, in the same order of the classes, separated by space.**

#### 1. dist(Counter（c）)

**Use this method to convert the list c into a dictionary.**

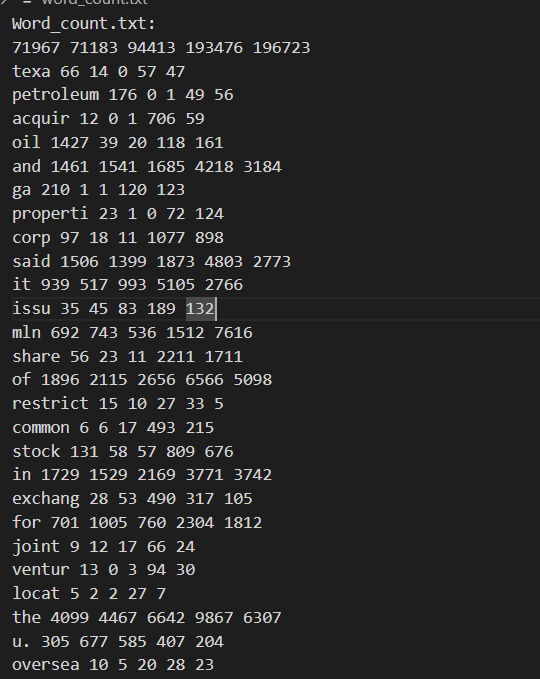
#### 2. C[key] = C.get(key,0)+1

**If the number of keys in the dictionary is 0, return 1, and key.value+1**

****

### 2.2.3 Result

**The first line contains the total document frequency in five classes, in the order of: 'crude', 'grain', 'money-fx', 'acq' and 'earn', separated by a space. In the following lines, each line contains a word and its frequency in five classes, in the same order of the classes, separated by space.**

****

## def feature\_selection(inputfile,threshold,outputfile)

### description

**Choose the most frequent 10000 words(defined by threshold) as the feature word then using the frequency obtained in 'word\_count.txt' to calculate the total word frequency in each class.**

**when calculating the word frequency, only words recognized as features are taken into consideration.**

**Output the result to the output file in the format required.**

### Approach

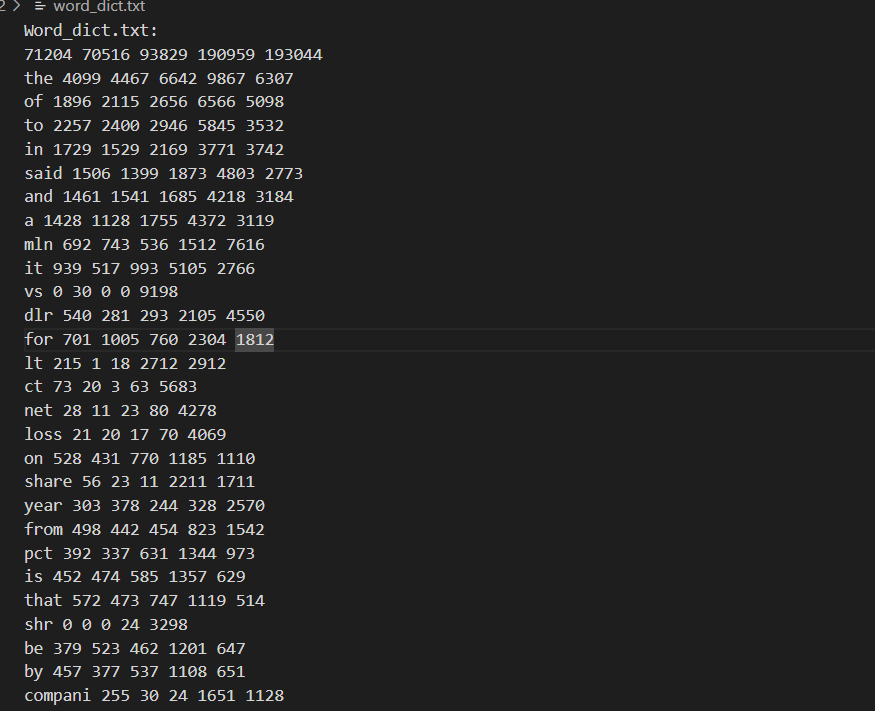
**Use a tuple to summarize the number of words that appear in word\_count, then sort them, select the 10,000 that appear the most, and then output them in a specific format.**

#### Sort the tuples

****

### Result

**The first line contains the total feature word frequency in five classes, in order of: 'crude', 'grain', 'money-fx', 'acq', and 'earn', separated by a space. In the following lines, each line contains a word and its frequency in five classes, in the same order as above, separated by space.**

****

## calculate\_probability(word\_dict,word\_count,outputfile)

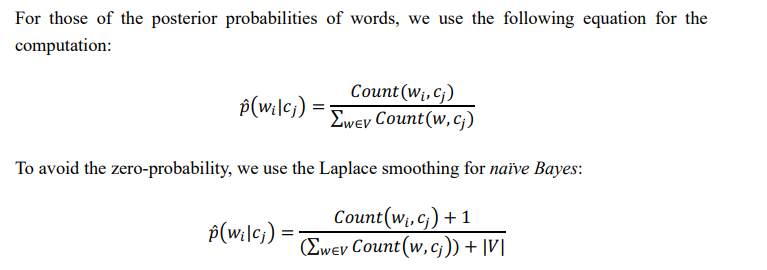
### Description

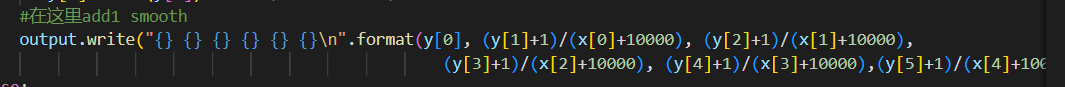
**Calculate the posterior probability of each feature word, and the prior probability of the class.**

**Output the result to the output file in the format required**

**Use 'word\_count.txt' and 'word\_dict.txt' jointly.**

### Approach(the Laplace smoothing for naïve Bayes)

****

****

### 2.4.3 Result

**The second line of probability is the prior probability of the class.**

**The probability of the following line is the posterior probability of the word.**

****

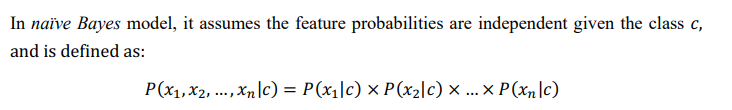
## def classify(probability,testset,outputfile)

### 2.5.1. Description

**Implement the naïve Bayes classifier to assign class labels to the documents in the test set.**

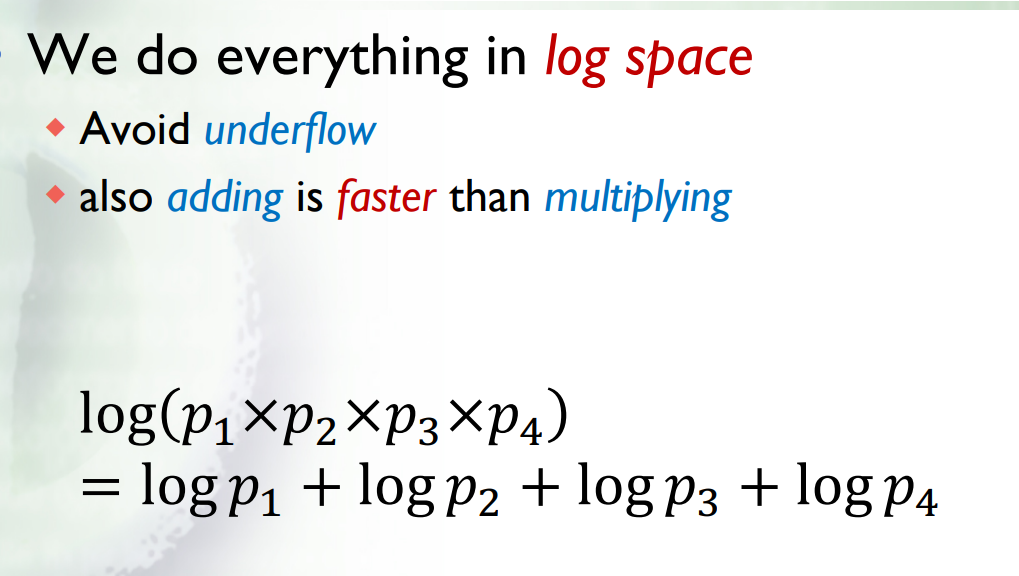
**Output the result to the output file in the format required.**

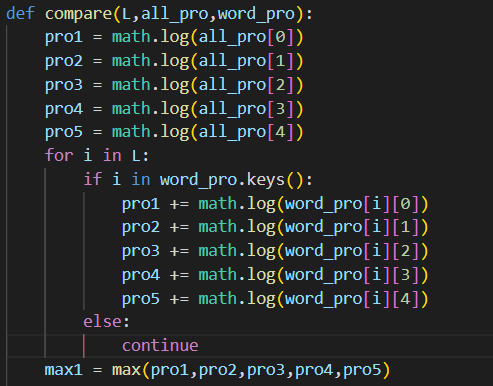
### 2.5.2. Approach

****

**Through the above formula, find the value of the characteristic probability and compare it, and assign it the class with the highest corresponding probability.**

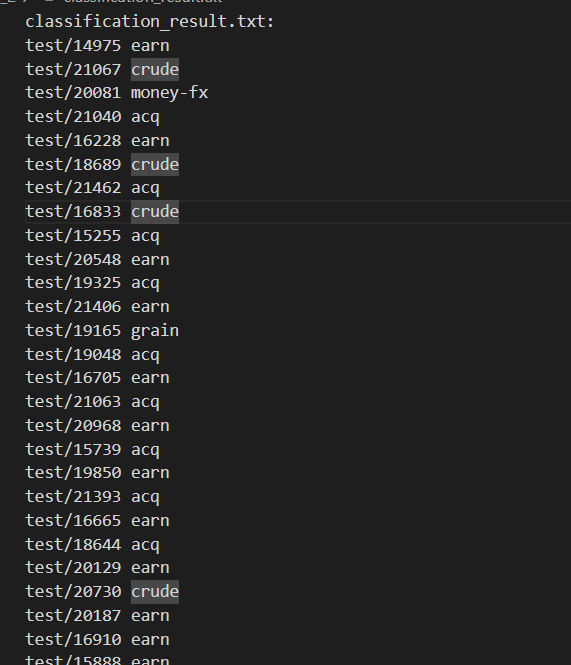
**It should be noted that in order to prevent data overflow, you should use:**

****

****

### 2.5.3. Result

**The first line is the text number, and the second line is the matching class result.**

****

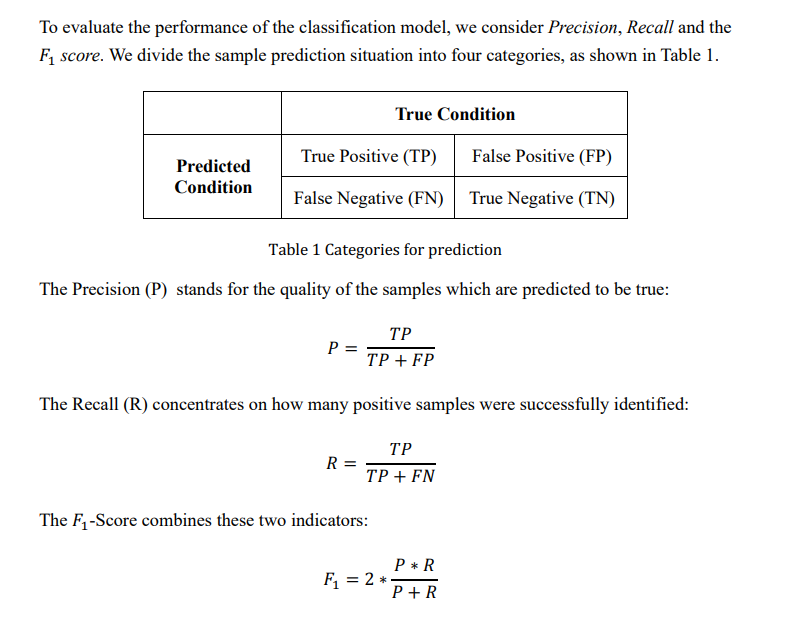
## 2.6. f1\_score(testset, classification\_result)

### 2.6.1. Description

**Use the F1 score to assess the performance of the implemented classification model，then the return value should be a float object.**

### 2.6.2. Approach

**Use the method described in the figure below to calculate F1. It should be noted that for multiple classes, micro-F1 or macro-F1 needs to be used. In this time, I used macro-F1-first get the TP , FP , FN value, and then calculate the Precision and Recall of each category, and then calculate the average of the Precision and Recall values of all categories, and finally obtain the overall F1 Score through the F1 formula.**

****

### 2.6.3. Result

****

# 3. Conclusion

**By completing this project independently, I not only gained a deeper understanding of the Naive Bayesian text classification model, but also learned a lot of small knowledge, such as reading and writing json files, and preprocessing strings (use NLTK .word\_tokenize() to tokenize the sentence and use nltk.PorterStemmer to stem the words), use the dictionary for word frequency statistics, calculate macro-F1, etc.**

**The difference in my project may be that the word segmentation method I used is different. I imported PorterStemmer for text segmentation. For word frequency statistics, I also learned and used dist(Counter()) to deal with it. . Besides, for calculating the class attribution possibility of the documents, I used logP1 + logP2+logP3 instead of P1\*P2\*P3.**

**In the end, I also hope that I can gain more knowledge in this class and at the same time achieve an ideal result, which is very important for my future studies in the junior year.**