文字探勘 HW3

資管三 B11705033 江睿宸

1. 執行環境：Jupyter Notebook
2. 程式語言：Python3
3. 程式碼說明

匯入需要的套件

In [1]:
```
import os
import nltk
from nltk.stem import PorterStemmer
import numpy as np
from collections import defaultdict
from math import log
import re
import csv

nltk.download('punkt')
```

讀取 1095 個檔案並根據 training.txt 分類

In [2]:
```
# 讀取training.txt文件
def load_training_data(training_file, data_directory):
    data = []
    labels = []
    training_indices = []
    with open(training_file, 'r', encoding='utf-8') as file:
        lines = file.readlines()
        for line in lines:
            parts = line.strip().split()
            class_id = int(parts[0])
            file_indices = parts[1:]
            training_indices.extend(int(index) for index in file_indices)
            for index in file_indices:
                file_path = os.path.join(data_directory, f"{index}.txt")
                with open(file_path, 'r', encoding='utf-8') as f:
                    data.append(f.read())
                labels.append(class_id)
    return data, labels, training_indices
```

```
# 讀取測試數據
def load_test_data(data_directory, training_indices):
    test_data = []
    test_indices = [i for i in range(1, 1096) if i not in training_indices]
    for index in test_indices:
        file_path = os.path.join(data_directory, f"{index}.txt")
        with open(file_path, 'r', encoding='utf-8') as f:
            test_data.append(f.read())
    return test_data, test_indices
```

使用 hw2 實作過的功能提取 tokens

```
stopwords = ["i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you", "your", "yours", "yourself", "yourselves", "he",

# From hw2
def preprocess_text(texts):
    processed_texts = []
    punctuation = ',.!?;:"`()_'
    punctuation += "'"
    stemmer = PorterStemmer()
    for text in texts:
        for char in punctuation:
            text = text.replace(char, '')
        tokens = text.lower().split()
        stemmed_words = [stemmer.stem(word) for word in tokens]
        processed_texts.append([word for word in stemmed_words if word not in stopwords])
    return processed_texts
```

```
training_file = 'training.txt'
data_directory = 'IRTM'
train_data, train_labels, training_indices = load_training_data(training_file, data_directory)
test_data, test_indices = load_test_data(data_directory, training_indices)

train_data = preprocess_text(train_data)
test_data = preprocess_text(test_data)
```

計算 Naïve Bayes 會用到的資料

In [3]:
```
def calculate_word_frequencies(train_data, train_labels):
    word_freqs = defaultdict(lambda: defaultdict(int))  # 保存每個類別中每個單詞的頻率
    class_counts = 15
    vocab = set()  # 所有出現在訓練數據中的單詞
    doc_counts = defaultdict(lambda: defaultdict(int))  # 保存每個類別中每個單詞有出現的文檔數

    for words, label in zip(train_data, train_labels):
        seen_words = set()  # 跟踪本文檔中已經見過的單詞
        for word in words:
            word_freqs[label][word] += 1
            vocab.add(word)
            if word not in seen_words:
                doc_counts[label][word] += 1
                seen_words.add(word)

    return word_freqs, class_counts, vocab, doc_counts

word_freqs, class_counts, vocab, doc_counts = calculate_word_frequencies(train_data, train_labels)
```

使用 frequency-based method 計算出前 500 個 term，計算中使用 hw2 統計過的 term frequency(在交出的作業已經將這段註解掉，但還是附上 dictionary.txt)

In [5]:
```
# frequency based
def load_dictionary(dictionary_file):
    term_frequencies = defaultdict(int)

    with open(dictionary_file, 'r', encoding='utf-8') as file:
        for line in file:
            parts = line.strip().split()
            term = parts[1]
            freq = int(parts[2])
            term_frequencies[term] += freq

    return term_frequencies

dictionary_file = 'dictionary.txt'# From hw2
term_frequencies = load_dictionary(dictionary_file)

def select_top_features(term_frequencies, top_n=500):
    sorted_terms = sorted(term_frequencies.items(), key=lambda item: item[1], reverse=True)
    top_features = [term for term, freq in sorted_terms[:top_n]]
    return top_features

top_500_features = select_top_features(term_frequencies, top_n=500)
```

計算 P(X=t|C)

In [10]:
```
# 計算條件概率 P(X=t|C)
def calculate_conditional_probs(word_freqs, class_counts, vocab, alpha=1):
    conditional_probs = defaultdict(lambda: defaultdict(float))
    for label in range(class_counts):
        total_words = sum(word_freqs[label].values())
        for word in vocab: #smoothing
            conditional_probs[label][word] = (word_freqs[label][word] + alpha) / (total_words + alpha * len(vocab))
    return conditional_probs

conditional_probs = calculate_conditional_probs(word_freqs, class_counts, vocab)
```

使用 NB 計算 P(C)*$\prod P(X = t|C)$，為了減少計算負擔先取 log 後相加

In [12]:
```
def classify(document, conditional_probs, class_counts, vocab):
    max_log_prob = float('-inf')
    best_class = None
    for label in range(class_counts):
        log_prob = log(1/13) #P(C)
        for word in document:
            if word in vocab:
                log_prob += np.log(conditional_probs[label][word]+1e-10)
        if log_prob > max_log_prob:
            max_log_prob = log_prob
            best_class = label
    return best_class

predictions = [classify(doc, conditional_probs, class_counts, top_500_features) for doc in test_data]
```

因為上次已經計算過 term frequency，因此使用 frequency-based method 計算出前 500 個 term，但在開始前先試過一次不選擇 500 個 term 而是整個單字集都一起計算的，認為相較之下結果不是很好，因此重新使用 chi-square 計算

In [7]:
```
def chi2_test(word_freqs, class_counts, doc_counts, vocab):
    chi2_scores = defaultdict(float)
    N = 13 * 15

    for word in vocab:
        for class_id in range(class_counts):
            TP = doc_counts[class_id][word]  # 該類別中包含該單詞的文檔數量
            FP = sum(doc_counts[c][word] for c in range(class_counts) if c != class_id)  # 其他類別中包含該單詞的文檔數量
            TN = 15 - TP  # 該類別中不包含該單詞的文檔數量
            FN = N - (TP + TN + FP)  # 其他類別中不包含該單詞的文檔數量
#             print(TP, ", ", TN, ", ", FP, ", ", FN)

            if (TP + FP) * (TN + FN) * (TP + TN) * (FP + FN) > 0:  # 防止分母為0
                chi2 = N * ((TP * FN - TN * FP) ** 2) / ((TP + FP) * (TN + FN) * (TP + TN) * (FP + FN))
                chi2_scores[word] += chi2

    return chi2_scores


chi2_scores = chi2_test(word_freqs, class_counts,doc_counts , vocab)
top_500_features = sorted(chi2_scores, key=chi2_scores.get, reverse=True)[:500]
# print(f"選擇的前500個特徵: {top_500_features}")
```

結果比整個資料集算出的模型還要更好