Programming Assignment 3

執行環境: Visual Studio Code
程式語言:Python 3.11.5
作業架構:

```
R12725048
├── report.pdf
├── training.txt
├── pa3.py
├── data/
```

執行方式

- 使用VS code跑pa3.py檔
- 需要下載的套件有:
  - pip install stop_words:刪除不太帶有資訊的單詞所需
  - pip install nltk: 使用Porter’s algorithm.所需
- 直接按全部執行即可

處理邏輯

- Step 1 : import 所需套件

```python
import math
from nltk.stem import PorterStemmer
import re
from stop_words import get_stop_words
import csv
```

處理邏輯

- Step 2 : 根據HW2 一樣先依據上次的news dataset建立class TF_TFIDF()，並建立其 object : tf_idf
  - ➢ 此步驟會做資料的前處理、建好dictionary、tf-idf table
  - ➢ 皆是上次作業的內容，因此簡報就不多花篇幅敘述

```
# step1:藉由HW2先根據所有data建立dictionary&tfidf table
file_path = "./IRTM"
num_docs = 1095
tf_idf = TF_IDF(file_path, num_docs)
```

處理邏輯

- Step 3 : 讀取training.txt，並將內容分別儲存成以下形式
- training_dataset = [1,2,3,...700,730,...,1019] → 將所有要拿來訓練的文章id儲存起來並照id大小排序
- training_list = {1:[11,19,29,113,...],2:[1,2,3,...],...}→ 用dict儲存每個class各別需要訓練的doc有哪些
- classes = [1,2,3,...13] → 用list紀錄所有的類別

```python
training_dataset = [] #用list來儲存所有要拿來training的doc id
training_list = {} #用dict分別儲存每個class中有那些training的doc id
classes = [] #用list來儲存class有哪些
f = open('training.txt')
for line in f.readlines():
    input = line.split(' ')
    if '\n' in input:
        input.remove('\n')
    input = [int(i) for i in input]
    training_list[input[0]] = input[1:]
    training_dataset.extend(input[1:])
    classes.append(input[0])
f.close
training_dataset.sort()
```

處理邏輯

- Step 4(1) : 重要features挑選

➢ 採取Log likelihood ratio的方式來挑features

➢ 而又因為作業為Multiple Classifiers，所以使用select the top k/n features for each n classifiers的方式來挑選500個features

➢ 要把最後的features union起來，因為不同類別可能挑到相同的feature

```python
# step2: 根據training dataset先做重要term的篩選
selected_features = FeaturesSelection(training_list, classes, int(500/13))
selected_features = set(selected_features)
```

```python
def FeaturesSelection(training_list:dict,classes:list, k:int) -> list:
    vocabulary = []
    score_list = {} #用來儲存每個class中的每個term的llr score e.g.score_list={'apple':0.025,'banana':3.521,...}
    #針對各個class各挑k個重要features最後合併起來
    for c in classes:
        V = ExtractVocabulary(training_list[c])
        score_list = likelihood_ratio(c, V)
        #將score_list依照values(llr score)排序
        sorted_score = dict(sorted(score_list.items(), key=lambda item: item[1], reverse=True))
        #每個class挑最高分的k個features合併起來
        vocabulary.extend(list(sorted_score.keys())[:k])

    return vocabulary
```

處理邏輯

- Step 4(2) :

likelihood ratio

```python
def likelihood_ratio(c:int, V:list) -> dict:
    score = {}
    for term in V:
        n11 = 0
        n01 = 0
        n10 = 0
        n00 = 0
        for doc in training_dataset:
            #若此doc為此類別(on topic)
            if doc in training_list[c]:
                #若term為present
                if term in ExtractTokensFromDoc(doc):
                    n11+=1
                #若term為absent
                else:
                    n10+=1
            #若此doc為不屬於此類別(off topic)
            else:
                #若term為present
                if term in ExtractTokensFromDoc(doc):
                    n01+=1
                #若term為absent
                else:
                    n00+=1
        pt = (n11+n01)/len(training_dataset)
        p1 = n11/(n11+n10)
        p2 = n01/(n01+n00)
        H1_likelihood = (math.pow(pt,n11)) * (math.pow((1-pt),n10)) * (math.pow(pt,n01)) * (math.pow((1-pt),n00))
        H2_likelihood = (math.pow(p1,n11)) * (math.pow((1-p1),n10)) * (math.pow(p2,n01)) * (math.pow((1-p2),n00))
        llr = (-2)*(math.log(H1_likelihood)-math.log(H2_likelihood))
        score[term] = llr
    return score
```

處理邏輯

- Step 5:建立class NaiveBayes()，並建立其object : NB，將NB的attributes&methods初始化

```python
# step3: 用Naive Bayes進行training
NB = NaiveBayes(tf_idf, classes, training_dataset, training_list)
prior, condprob = NB.TrainMultinomialNB(selected_features)
```

```python
class NaiveBayes():
    def __init__(self, tf_idf:TF_IDF, classes:list, training_dataset:list, training_list:dict):
        self.tf_idf = tf_idf
        self.classes = classes
        self.training_dataset = training_dataset
        self.training_list = training_list

```

處理邏輯

- Step 6 : 進行training

```python
# step3: 用Naive Bayes進行training
NB = NaiveBayes(tf_idf, classes, training_dataset, training_list)
prior, condprob = NB.TrainMultinomialNB(selected_features)
```

```python
def TrainMultinomialNB(self, V:list) -> list | dict :
    #宣告prior陣列存放各類別的P(C)值
    prior = [0 for i in range(0, len(self.classes)+1)]
    condprob = {}
    N = len(self.training_dataset)

    for c in self.classes:
        Nc = CountDocsInClass(c)
        prior[c] = Nc/N
        text_c = ConcatenateTextOfAllDocsInClass(self.training_list[c])

        num_of_term_in_class_c = sum(list(text_c.values()))

        for term in V:

            T_ct = CountTokensOfTerm(text_c, term)
            if term not in condprob:
                condprob[term] = [1/(num_of_term_in_class_c+len(V)) for i in range(0, len(self.classes)+1)]

            condprob[term][c] = (T_ct+1)/(num_of_term_in_class_c+len(V))
    return prior, condprob
```

紀錄每個term在每個class的分數，並做smoothing處理

處理邏輯

```python
# step4: 將剩下的資料集進行apply
mapping_class_of_doc = [0 for i in range(1, num_docs+2)]
for doc in range(1,num_docs+1):
    mapping_class_of_doc[doc] = NB.ApplyMultinomialNB(selected_features, classes, prior, condprob, doc)
```

- Step 7 : 將所有dataset拿去驗證

```python
def ApplyMultinomialNB(self, V:list, categories:list, prior:list, condprob:dict, d:int) -> int:
    word_in_d = ExtractTokensFromDoc(d)
    score = [0 for i in range(0, len(categories)+1)]
    max_score = -10000000
    mapping = 0
    for c in categories:
        score[c] = math.log(prior[c])
        for term in word_in_d:
            #只計算selected_features裡的字，其他不用算分數
            if term in V:
                score[c] += math.log(condprob[term][c]) * tf_idf.tf[d][term]
        #紀錄最高分的類別是哪個
        if score[c] > max_score:
            max_score = score[c]
            mapping = c
    return mapping
```

處理邏輯

- Step 8: 將result寫到csv檔

```python
# step5: write result to  'HW3_csv.csv'
header = ['Id','Value']
data = []
for i in range(1,num_docs+1):
    if i not in training_dataset:
        data.append([i,mapping_class_of_doc[i]])


with open('HW3_csv.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)

    # write the header
    writer.writerow(header)

    # write multiple rows
    writer.writerows(data)
```

Kaggle 分數

8	**r12725048**	0.99111	14	13h

Your Best Entry!
Your most recent submission scored 0.99111, which is an improvement of your previous score of 0.99000. Great job!

Tweet this