Information Retrieval
B06406009 資管三 陳姵如

```
# IR Programming Assignment2
# 資管三 B06406009 陳姵如
```

一、環境
    python 3.7

二、library

```
import re
import os
import io
import math
import string
import numpy
from nltk import PorterStemmer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

三、讀取資料路徑
    1. 要把 sourcecode 跟 1095篇文檔放一起
    2. Training data 讀檔名為 “trainingdata.txt”

四、Code部分解釋/註解

```
#read the class index

index = []

file = open('trainingdata.txt', 'r')

for i in range (13):
    index.append((file.readline()).split())
    del index[i][0]

file.close()

for i in range (13):
    for j in range (15):
        index[i][j] = str(index[i][j]) + '.txt'

# Poter's Algorithm

ClassifiedDocument = []

class PorterStemmer:

    def __init__(self):
        self.b = ""  # buffer for word to be stemmed
```

```
        self.k = 0
        self.k0 = 0
        self.j = 0   # j is a general offset into the string

    def cons(self, i):
        if self.b[i] == 'a' or self.b[i] == 'e' or self.b[i] ==
'i' or self.b[i] == 'o' or self.b[i] == 'u':
            return 0
        if self.b[i] == 'y':
            if i == self.k0:
                return 1
            else:
                return (not self.cons(i - 1))
        return 1

    def m(self):
        n = 0
        i = self.k0
        while 1:
            if i > self.j:
                return n
            if not self.cons(i):
                break
            i = i + 1
        i = i + 1
        while 1:
            while 1:
                if i > self.j:
                    return n
                if self.cons(i):
                    break
                i = i + 1
            i = i + 1
            n = n + 1
            while 1:
                if i > self.j:
                    return n
                if not self.cons(i):
                    break
                i = i + 1
            i = i + 1

    def vowelinstem(self):
        for i in range(self.k0, self.j + 1):
            if not self.cons(i):
                return 1
        return 0

    def doublec(self, j):
        if j < (self.k0 + 1):
            return 0
        if (self.b[j] != self.b[j-1]):
```

```
            return 0
        return self.cons(j)

    def cvc(self, i):
        if i < (self.k0 + 2) or not self.cons(i) or self.cons(i-1)
or not self.cons(i-2):
            return 0
        ch = self.b[i]
        if ch == 'w' or ch == 'x' or ch == 'y':
            return 0
        return 1

    def ends(self, s):
        length = len(s)
        if s[length - 1] != self.b[self.k]: # tiny speed-up
            return 0
        if length > (self.k - self.k0 + 1):
            return 0
        if self.b[self.k-length+1:self.k+1] != s:
            return 0
        self.j = self.k - length
        return 1

    def setto(self, s):
        length = len(s)
        self.b = self.b[:self.j+1] + s + self.b[self.j+length+1:]
        self.k = self.j + length

    def r(self, s):
        if self.m() > 0:
            self.setto(s)

    def step1ab(self):
        """step1ab() gets rid of plurals and -ed or -ing. e.g.

           caresses  ->  caress
           ponies    ->  poni
           ties      ->  ti
           caress    ->  caress
           cats      ->  cat

           feed      ->  feed
           agreed    ->  agree
           disabled  ->  disable

           matting   ->  mat
           mating    ->  mate
           meeting   ->  meet
           milling   ->  mill
           messing   ->  mess

           meetings  ->  meet
```

```
        """
        if self.b[self.k] == 's':
            if self.ends("sses"):
                self.k = self.k - 2
            elif self.ends("ies"):
                self.setto("i")
            elif self.b[self.k - 1] != 's':
                self.k = self.k - 1
        if self.ends("eed"):
            if self.m() > 0:
                self.k = self.k - 1
        elif (self.ends("ed") or self.ends("ing")) and
self.vowelinstem():
            self.k = self.j
            if self.ends("at"):   self.setto("ate")
            elif self.ends("bl"): self.setto("ble")
            elif self.ends("iz"): self.setto("ize")
            elif self.doublec(self.k):
                self.k = self.k - 1
                ch = self.b[self.k]
                if ch == 'l' or ch == 's' or ch == 'z':
                    self.k = self.k + 1
            elif (self.m() == 1 and self.cvc(self.k)):
                self.setto("e")

    def step1c(self):
        """step1c() turns terminal y to i when there is another
vowel in the stem."""
        if (self.ends("y") and self.vowelinstem()):
            self.b = self.b[:self.k] + 'i' + self.b[self.k+1:]

    def step2(self):
        """step2() maps double suffices to single ones.
        so -ization ( = -ize plus -ation) maps to -ize etc. note
that the
        string before the suffix must give m() > 0.
        """
        if self.b[self.k - 1] == 'a':
            if self.ends("ational"):   self.r("ate")
            elif self.ends("tional"):  self.r("tion")
        elif self.b[self.k - 1] == 'c':
            if self.ends("enci"):      self.r("ence")
            elif self.ends("anci"):    self.r("ance")
        elif self.b[self.k - 1] == 'e':
            if self.ends("izer"):      self.r("ize")
        elif self.b[self.k - 1] == 'l':
            if self.ends("bli"):       self.r("ble") # --
DEPARTURE--
            # To match the published algorithm, replace this
phrase with
            #   if self.ends("abli"):      self.r("able")
            elif self.ends("alli"):    self.r("al")
```

```
            elif self.ends("entli"):   self.r("ent")
            elif self.ends("eli"):     self.r("e")
            elif self.ends("ousli"):   self.r("ous")
        elif self.b[self.k - 1] == 'o':
            if self.ends("ization"):   self.r("ize")
            elif self.ends("ation"):   self.r("ate")
            elif self.ends("ator"):    self.r("ate")
        elif self.b[self.k - 1] == 's':
            if self.ends("alism"):     self.r("al")
            elif self.ends("iveness"): self.r("ive")
            elif self.ends("fulness"): self.r("ful")
            elif self.ends("ousness"): self.r("ous")
        elif self.b[self.k - 1] == 't':
            if self.ends("aliti"):     self.r("al")
            elif self.ends("iviti"):   self.r("ive")
            elif self.ends("biliti"):  self.r("ble")
        elif self.b[self.k - 1] == 'g': # --DEPARTURE--
            if self.ends("logi"):      self.r("log")
        # To match the published algorithm, delete this phrase

    def step3(self):
        """step3() dels with -ic-, -full, -ness etc. similar
strategy to step2."""
        if self.b[self.k] == 'e':
            if self.ends("icate"):     self.r("ic")
            elif self.ends("ative"):   self.r("")
            elif self.ends("alize"):   self.r("al")
        elif self.b[self.k] == 'i':
            if self.ends("iciti"):     self.r("ic")
        elif self.b[self.k] == 'l':
            if self.ends("ical"):      self.r("ic")
            elif self.ends("ful"):     self.r("")
        elif self.b[self.k] == 's':
            if self.ends("ness"):      self.r("")

    def step4(self):
        """step4() takes off -ant, -ence etc., in context
<c>vcvc<v>."""
        if self.b[self.k - 1] == 'a':
            if self.ends("al"): pass
            else: return
        elif self.b[self.k - 1] == 'c':
            if self.ends("ance"): pass
            elif self.ends("ence"): pass
            else: return
        elif self.b[self.k - 1] == 'e':
            if self.ends("er"): pass
            else: return
        elif self.b[self.k - 1] == 'i':
            if self.ends("ic"): pass
            else: return
        elif self.b[self.k - 1] == 'l':
```

```
            if self.ends("able"): pass
            elif self.ends("ible"): pass
            else: return
        elif self.b[self.k - 1] == 'n':
            if self.ends("ant"): pass
            elif self.ends("ement"): pass
            elif self.ends("ment"): pass
            elif self.ends("ent"): pass
            else: return
        elif self.b[self.k - 1] == 'o':
            if self.ends("ion") and (self.b[self.j] == 's' or
self.b[self.j] == 't'): pass
            elif self.ends("ou"): pass
            # takes care of -ous
            else: return
        elif self.b[self.k - 1] == 's':
            if self.ends("ism"): pass
            else: return
        elif self.b[self.k - 1] == 't':
            if self.ends("ate"): pass
            elif self.ends("iti"): pass
            else: return
        elif self.b[self.k - 1] == 'u':
            if self.ends("ous"): pass
            else: return
        elif self.b[self.k - 1] == 'v':
            if self.ends("ive"): pass
            else: return
        elif self.b[self.k - 1] == 'z':
            if self.ends("ize"): pass
            else: return
        else:
            return
        if self.m() > 1:
            self.k = self.j

    def step5(self):
        """step5() removes a final -e if m() > 1, and changes -ll
to -l if
        m() > 1.
        """
        self.j = self.k
        if self.b[self.k] == 'e':
            a = self.m()
            if a > 1 or (a == 1 and not self.cvc(self.k-1)):
                self.k = self.k - 1
        if self.b[self.k] == 'l' and self.doublec(self.k) and
self.m() > 1:
            self.k = self.k -1

    def stem(self, p, i, j):
        # copy the parameters into statics
```

```
        self.b = p
        self.k = j
        self.k0 = i
        if self.k <= self.k0 + 1:
            return self.b # --DEPARTURE--

        # With this line, strings of length 1 or 2 don't go
through the
        # stemming process, although no mention is made of this in
the
        # published algorithm. Remove the line to match the
published
        # algorithm.

        self.step1ab()
        self.step1c()
        self.step2()
        self.step3()
        self.step4()
        self.step5()
        return self.b[self.k0:self.k+1]

if __name__ == '__main__':
    p = PorterStemmer()
    if len(sys.argv) > 1:
        for f in sys.argv[1:]:
            for i in range (13):
                tmp = []
                for j in range (15):
                    infile = (open(index[i][j], 'r'))
                    tmp.append(infile)
                    output = ''
                    word = ''
                    if tmp[j] == '':
                        break
                    for c in tmp[j]:
                        if c.isalpha():
                            word += c.lower()
                        else:
                            if word:
                                output += p.stem(word, 0,len(word)-
1)
                                word = ''
                            output += c.lower()
                            tmp[j] = output
                ClassifiedDocument.append(tmp)
            infile.close()

# Tokenization

TokenClassifiedDocument = [] #有重複
```

```
for i in range (13):
    tmp = []
    for j in range (15):
        tmp.append(word_tokenize(ClassifiedDocument[i][j]))
    TokenClassifiedDocument.append(tmp)

# Stopword removal
# Redundancy removal

TokenClassified = []

redundency =
['!','.','?','$',"'s","'",',',"n't","'","'ve",'would']
stop_words = set(stopwords.words('english') + redundency)
for i in range (13):
    tmp2 = []
    for j in range (15):
        tmp = []
        for w in range(len(TokenClassifiedDocument[i][j])):
            if (TokenClassifiedDocument[i][j][w] not in
stop_words) and (TokenClassifiedDocument[i][j][w].isalpha() ==
True):
                tmp.append(TokenClassifiedDocument[i][j][w])
        tmp2.append(tmp)
    TokenClassified.append(tmp2)

# Sort
for i in range (13):
    for j in range (15):
        TokenClassified[i][j].sort()

# Establish my classfied Documents

TermClassifiedDocument = [] #每個文檔去掉重複
ClassifiedDictionary = [] #13類的共用字典

for i in range (13):
    tmp2 = []
    for j in range (15):
        tmp = []
        for k in range(len(TokenClassified[i][j])):
            if TokenClassified[i][j][k] not in tmp:
                tmp.append(TokenClassified[i][j][k])
        tmp.sort()
        tmp2.append(tmp)
    TermClassifiedDocument.append(tmp2)


# Establish my classified dictionary
```

```
for i in range (13):
    for j in range (15):
        for k in range(len(TermClassifiedDocument[i][j])):
            if TermClassifiedDocument[i][j][k] not in
ClassifiedDictionary:

ClassifiedDictionary.append(TermClassifiedDocument[i][j][k])

ClassifiedDictionary.sort()

# Count the OntopicCnt and OfftopicCnt

OntopicCnt = [] // Positive True
OfftopicCnt = [] // Positive False

for i in range (len(ClassifiedDictionary)):
    tmp = []
    for j in range(13):
        cnt = 0
        for k in range(15):
            if ClassifiedDictionary[i] in
TermClassifiedDocument[j][k]:
                cnt = cnt+1
        tmp.append(cnt)
    OntopicCnt.append(tmp)

for i in range(len(ClassifiedDictionary)):
    tmp = []
    for block in range (13): # term 現在所在的類
        cnt = 0
        for j in range(13):
            if j != block: # 跳過現在所在的類別
                for k in range(15):
                    if ClassifiedDictionary[i] in
TermClassifiedDocument[j][k]:
                        cnt = cnt+1
        tmp.append(cnt)
    OfftopicCnt.append(tmp)

# Calculate the value of likelihood

LikelihoodRatio = []
pt = []
p1 = []
p2 = []

for i in range(len(ClassifiedDictionary)):
    tmppt = []
    tmpp1 = []
    tmpp2 = []
    for j in range(13):
```

```
        tmppt.append((OntopicCnt[i][j] + (180-OfftopicCnt[i][j]) +
1) / (195)) #弄懂加多少
        tmpp1.append((OntopicCnt[i][j] + 1) / (15))
        tmpp2.append((OfftopicCnt[i][j] + 1) / (180))
    pt.append(tmppt)
    p1.append(tmpp1)
    p2.append(tmpp2)

L1 = []
L2 = []

for i in range(len(ClassifiedDictionary)):
    tmpL1 = []
    tmpL2 = []
    for j in range(13):
        tmpL1.append((pt[i][j]**(OntopicCnt[i][j]+(180-
OfftopicCnt[i][j])) * (1-pt[i][j]))**((15-
OntopicCnt[i][j])+OfftopicCnt[i][j]))
        tmpL2.append((p1[i][j]**OntopicCnt[i][j] * (1-
p1[i][j])**(15-OntopicCnt[i][j]) * p2[i][j]**(180-
OfftopicCnt[i][j]) * (1-p2[i][j])**OfftopicCnt[i][j]))
    L1.append(tmpL1)
    L2.append(tmpL2)


def likelihood_ratio(L1, L2):
    return(-2*(L1-L2))

for j in range(13):
    tmpLH = []
    for i in range(len(ClassifiedDictionary)):
        tmpLH.append(likelihood_ratio(L1[i][j],L2[i][j]))
    LikelihoodRatio.append(tmpLH)

# Select the top 500 features for every class

FeatureSelected = []
tmp = []
limit = 500

for i in range(13):
    tup = []
    for j in range(len(ClassifiedDictionary)):
        tup.append((ClassifiedDictionary[j],LikelihoodRatio[i][j]))
    tup.sort(key=lambda tup: tup[1])  # sorts in place
    tmp.append(tup)

for i in range(13):
    tmp2 = []
    for j in range(limit):
        tmp2.append(tmp[i][j][0])
```

```
    FeatureSelected.append(tmp2)

# index 1095 documents

RawDocument = []
a = []
b = 0
FileNum = 1095

for i in range (FileNum):
    b = b+1
    a.append(str(b) + '.txt')

# Poter's Algorithm

ClassifiedDocument = []

class PorterStemmer:

    def __init__(self):
        self.b = ""  # buffer for word to be stemmed
        self.k = 0
        self.k0 = 0
        self.j = 0   # j is a general offset into the string

    def cons(self, i):
        if self.b[i] == 'a' or self.b[i] == 'e' or self.b[i] ==
'i' or self.b[i] == 'o' or self.b[i] == 'u':
            return 0
        if self.b[i] == 'y':
            if i == self.k0:
                return 1
            else:
                return (not self.cons(i - 1))
        return 1

    def m(self):
        n = 0
        i = self.k0
        while 1:
            if i > self.j:
                return n
            if not self.cons(i):
                break
            i = i + 1
        i = i + 1
        while 1:
            while 1:
                if i > self.j:
                    return n
                if self.cons(i):
                    break
```

```
                i = i + 1
            i = i + 1
            n = n + 1
            while 1:
                if i > self.j:
                    return n
                if not self.cons(i):
                    break
                i = i + 1
            i = i + 1

    def vowelinstem(self):
        for i in range(self.k0, self.j + 1):
            if not self.cons(i):
                return 1
        return 0

    def doublec(self, j):
        if j < (self.k0 + 1):
            return 0
        if (self.b[j] != self.b[j-1]):
            return 0
        return self.cons(j)

    def cvc(self, i):
        if i < (self.k0 + 2) or not self.cons(i) or self.cons(i-1)
or not self.cons(i-2):
            return 0
        ch = self.b[i]
        if ch == 'w' or ch == 'x' or ch == 'y':
            return 0
        return 1

    def ends(self, s):
        length = len(s)
        if s[length - 1] != self.b[self.k]: # tiny speed-up
            return 0
        if length > (self.k - self.k0 + 1):
            return 0
        if self.b[self.k-length+1:self.k+1] != s:
            return 0
        self.j = self.k - length
        return 1

    def setto(self, s):
        length = len(s)
        self.b = self.b[:self.j+1] + s + self.b[self.j+length+1:]
        self.k = self.j + length

    def r(self, s):
        if self.m() > 0:
            self.setto(s)
```

```
    def step1ab(self):
        """step1ab() gets rid of plurals and -ed or -ing. e.g.

           caresses  ->  caress
           ponies    ->  poni
           ties      ->  ti
           caress    ->  caress
           cats      ->  cat

           feed      ->  feed
           agreed    ->  agree
           disabled  ->  disable

           matting   ->  mat
           mating    ->  mate
           meeting   ->  meet
           milling   ->  mill
           messing   ->  mess

           meetings  ->  meet
        """
        if self.b[self.k] == 's':
            if self.ends("sses"):
                self.k = self.k - 2
            elif self.ends("ies"):
                self.setto("i")
            elif self.b[self.k - 1] != 's':
                self.k = self.k - 1
        if self.ends("eed"):
            if self.m() > 0:
                self.k = self.k - 1
        elif (self.ends("ed") or self.ends("ing")) and
self.vowelinstem():
            self.k = self.j
            if self.ends("at"):   self.setto("ate")
            elif self.ends("bl"): self.setto("ble")
            elif self.ends("iz"): self.setto("ize")
            elif self.doublec(self.k):
                self.k = self.k - 1
                ch = self.b[self.k]
                if ch == 'l' or ch == 's' or ch == 'z':
                    self.k = self.k + 1
            elif (self.m() == 1 and self.cvc(self.k)):
                self.setto("e")

    def step1c(self):
        """step1c() turns terminal y to i when there is another
vowel in the stem."""
        if (self.ends("y") and self.vowelinstem()):
            self.b = self.b[:self.k] + 'i' + self.b[self.k+1:]
```

```
    def step2(self):
        """step2() maps double suffices to single ones.
        so -ization ( = -ize plus -ation) maps to -ize etc. note
that the
        string before the suffix must give m() > 0.
        """
        if self.b[self.k - 1] == 'a':
            if self.ends("ational"):   self.r("ate")
            elif self.ends("tional"):  self.r("tion")
        elif self.b[self.k - 1] == 'c':
            if self.ends("enci"):      self.r("ence")
            elif self.ends("anci"):    self.r("ance")
        elif self.b[self.k - 1] == 'e':
            if self.ends("izer"):      self.r("ize")
        elif self.b[self.k - 1] == 'l':
            if self.ends("bli"):       self.r("ble") # --
DEPARTURE--
            # To match the published algorithm, replace this
phrase with
            #   if self.ends("abli"):      self.r("able")
            elif self.ends("alli"):    self.r("al")
            elif self.ends("entli"):   self.r("ent")
            elif self.ends("eli"):     self.r("e")
            elif self.ends("ousli"):   self.r("ous")
        elif self.b[self.k - 1] == 'o':
            if self.ends("ization"):   self.r("ize")
            elif self.ends("ation"):   self.r("ate")
            elif self.ends("ator"):    self.r("ate")
        elif self.b[self.k - 1] == 's':
            if self.ends("alism"):     self.r("al")
            elif self.ends("iveness"): self.r("ive")
            elif self.ends("fulness"): self.r("ful")
            elif self.ends("ousness"): self.r("ous")
        elif self.b[self.k - 1] == 't':
            if self.ends("aliti"):     self.r("al")
            elif self.ends("iviti"):   self.r("ive")
            elif self.ends("biliti"):  self.r("ble")
        elif self.b[self.k - 1] == 'g': # --DEPARTURE--
            if self.ends("logi"):      self.r("log")
        # To match the published algorithm, delete this phrase

    def step3(self):
        """step3() dels with -ic-, -full, -ness etc. similar
strategy to step2."""
        if self.b[self.k] == 'e':
            if self.ends("icate"):     self.r("ic")
            elif self.ends("ative"):   self.r("")
            elif self.ends("alize"):   self.r("al")
        elif self.b[self.k] == 'i':
            if self.ends("iciti"):     self.r("ic")
        elif self.b[self.k] == 'l':
            if self.ends("ical"):      self.r("ic")
```

```
            elif self.ends("ful"):     self.r("")
        elif self.b[self.k] == 's':
            if self.ends("ness"):      self.r("")

    def step4(self):
        """step4() takes off -ant, -ence etc., in context
<c>vcvc<v>."""
        if self.b[self.k - 1] == 'a':
            if self.ends("al"): pass
            else: return
        elif self.b[self.k - 1] == 'c':
            if self.ends("ance"): pass
            elif self.ends("ence"): pass
            else: return
        elif self.b[self.k - 1] == 'e':
            if self.ends("er"): pass
            else: return
        elif self.b[self.k - 1] == 'i':
            if self.ends("ic"): pass
            else: return
        elif self.b[self.k - 1] == 'l':
            if self.ends("able"): pass
            elif self.ends("ible"): pass
            else: return
        elif self.b[self.k - 1] == 'n':
            if self.ends("ant"): pass
            elif self.ends("ement"): pass
            elif self.ends("ment"): pass
            elif self.ends("ent"): pass
            else: return
        elif self.b[self.k - 1] == 'o':
            if self.ends("ion") and (self.b[self.j] == 's' or
self.b[self.j] == 't'): pass
            elif self.ends("ou"): pass
            # takes care of -ous
            else: return
        elif self.b[self.k - 1] == 's':
            if self.ends("ism"): pass
            else: return
        elif self.b[self.k - 1] == 't':
            if self.ends("ate"): pass
            elif self.ends("iti"): pass
            else: return
        elif self.b[self.k - 1] == 'u':
            if self.ends("ous"): pass
            else: return
        elif self.b[self.k - 1] == 'v':
            if self.ends("ive"): pass
            else: return
        elif self.b[self.k - 1] == 'z':
            if self.ends("ize"): pass
            else: return
```

```
        else:
            return
        if self.m() > 1:
            self.k = self.j

    def step5(self):
        """step5() removes a final -e if m() > 1, and changes -ll
to -l if
        m() > 1.
        """
        self.j = self.k
        if self.b[self.k] == 'e':
            a = self.m()
            if a > 1 or (a == 1 and not self.cvc(self.k-1)):
                self.k = self.k - 1
        if self.b[self.k] == 'l' and self.doublec(self.k) and
self.m() > 1:
            self.k = self.k -1

    def stem(self, p, i, j):
        # copy the parameters into statics
        self.b = p
        self.k = j
        self.k0 = i
        if self.k <= self.k0 + 1:
            return self.b # --DEPARTURE--

        # With this line, strings of length 1 or 2 don't go
through the
        # stemming process, although no mention is made of this in
the
        # published algorithm. Remove the line to match the
published
        # algorithm.

        self.step1ab()
        self.step1c()
        self.step2()
        self.step3()
        self.step4()
        self.step5()
        return self.b[self.k0:self.k+1]

if __name__ == '__main__':
    p = PorterStemmer()
    if len(sys.argv) > 1:
        for f in sys.argv[1:]:
            for i in range (FileNum):
                infile = (open(a[i], 'r'))
                tmp = infile
                output = ''
                word = ''
```

```
                if tmp == '':
                    break
                for c in tmp:
                    if c.isalpha():
                        word += c.lower()
                    else:
                        if word:
                            output += p.stem(word, 0,len(word)-1)
                            word = ''
                        output += c.lower()
                RawDocument.append(output)
        infile.close()

# Tokenization

TokenDocument = [] #有重複
Token = []

for i in range (FileNum):
    TokenDocument.append(word_tokenize(RawDocument[i]))

# Stopword removal
# Redundancy removal

redundency =
['!','.','?','$',"'s","'",',',"n't","'","'ve",'would','c']
stop_words = set(stopwords.words('english') + redundency)
for i in range (FileNum):
    tmp = []
    for j in range(len(TokenDocument[i])):
        if (TokenDocument[i][j] not in stop_words) and
(TokenDocument[i][j].isalpha() == True):
            tmp.append(TokenDocument[i][j])
    Token.append(tmp)

# Sort
for i in range(FileNum):
    Token[i].sort()

# Establish my classified Documents

TermDocument = [] #每個文檔去掉重複
Dictionary = [] #13類的共用字典

for i in range(FileNum):
    tmp = []
    for k in range(len(Token[i])):
        if Token[i][k] not in tmp:
            tmp.append(Token[i][k])
    tmp.sort()
    TermDocument.append(tmp)
```

```
## Naive Bayes using the Motinomial Model

DocScore = [] # 1095*13

ClassifiedTokenFrequncy = []
ClassifiedTokenProb = []
ClassSumFrequncy = []

for c in range(13):
    tmp = {}
    tmpCnt = 0
    for i in range(13): # 算每個classifiedToken在字典出現機率
        for j in range(15):
            cnt = 0
            for k in range(len(TokenClassified[i][j])):
                if TokenClassified[i][j][k] in FeatureSelected[c]:
                    if TokenClassified[i][j][k] in tmp:
                        tmp[TokenClassified[i][j][k]] =
tmp[TokenClassified[i][j][k]]+1
                        tmpCnt = tmpCnt +1
                    else:
                        tmp[TokenClassified[i][j][k]] = 1
                        tmpCnt = tmpCnt+1
    ClassifiedTokenFrequncy.append(tmp)
    ClassSumFrequncy.append(tmpCnt)


for c in range(13):
    ClassifiedTokenProb.append(ClassifiedTokenFrequncy[c])
    for j in range(15):
        for k in range(len(TokenClassified[c][j])):
            if TokenClassified[c][j][k] in ClassifiedTokenProb[c]:
                if
ClassifiedTokenProb[c].get(TokenClassified[c][j][k], 'not exist')
>= 1:

ClassifiedTokenProb[c][TokenClassified[c][j][k]] =
(ClassifiedTokenProb[c][TokenClassified[c][j][k]]+1)/
(ClassSumFrequncy[c]+len(FeatureSelected[c]))



for i in range(FileNum):
    tmp = []
    for c in range(13):
        probsum = 100000
        for j in range(len(Token[i])):
             if Token[i][j] in FeatureSelected[c]:
                probsum = probsum +
math.log(ClassifiedTokenProb[c].get(Token[i][j]))
```

```
            tmp.append(probsum)
        DocScore.append(tmp)

    # Classify the documents

    SuitedClass = []
    MaxScore = 0.0

    for i in range(FileNum):
        MaxScore = float('inf')
        Suited = 0
        for j in range(13):
            if DocScore[i][j] < MaxScore:
                MaxScore = DocScore[i][j]
                Suited = j+1
        SuitedClass.append(Suited)

    tmp2 = []
    for i in range(13):
        tmp2 = tmp2 + training[i]
    for i in range(FileNum):
        if str(i+1) not in tmp2:
            print(i+1,SuitedClass[i]) # print out the testing document
    ID and the suited class
```