

# **Information Retrieval and Web Search**

By

**Sritej Reddy - 17ucs094**

**Course Coordinator**

**Dr. Suvidha Tripathi**



Department of Computer Science Engineering  
The LNM Institute of Information Technology, Jaipur

April 2021

## Contents

Chapter	Page
1 Part- B . . . . .	1
1.1 Overview . . . . .	1
1.2 Methodology . . . . .	1
1.2.1 Input documents in dictionary . . . . .	1
1.2.2 Tokenizing the data . . . . .	2
1.2.3 Classes and their probabilities . . . . .	3
1.2.4 Terms Occurrences in a class . . . . .	4
1.2.5 Parameters Calculation . . . . .	5
1.3 Construction of Classifier . . . . .	6
1.4 Evaluation of Classification results . . . . .	7
1.4.1 Confusion matrix . . . . .	7
1.4.2 Precision, Recall, F1-score . . . . .	8

## Chapter 1

### Part- B

#### 1.1 Overview

In this report application of Multinomial Naive Bayes Classifier which is based on Bayes' theorem in classifying a test document is shown.

#### 1.2 Methodology

##### 1.2.1 Input documents in dictionary

Taking docIDs as keys and data in the documents as values in the dictionary.

##### Listing 1.1 Input dataset

```
docs={}
docs[1]="Kyoto_Osaka_Taiwan"
docs[2]="Japan_Kyoto"
docs[3]="Taipei_Taiwan"
docs[4]="Macao_Taiwan_Shanghai"
docs[5]="London"
docs[6]="Taiwan_Taiwan_Kyoto"
```

### 1.2.2 Tokenizing the data

*Word\_tokenize* method from the *nltk* module has been utilised to output tokens from text data. Here 't' is list type output of docID 'i' which has tokens in it assigning this list to *key* 'i' of tokens dictionary.

**Listing 1.2** Tokenizing & storing in tokens dictionary

```
from nltk import word_tokenize
tokens={}
for i in range(1,7):
    t=word_tokenize(docs[i])
    tokens[i]=t
```

Key	Type	Size	
1	list	3	['Kyoto', 'Osaka', 'Taiwan']
2	list	2	['Japan', 'Kyoto']
3	list	2	['Taipei', 'Taiwan']
4	list	3	['Macao', 'Taiwan', 'Shanghai']
5	list	1	['London']
6	list	3	['Taiwan', 'Taiwan', 'Kyoto']

**Figure 1.1** screenshot of tokens dictionary.

### 1.2.3 Classes and their probabilities

Storing the given class for training set

**Listing 1.3** Assigning classes for training set

```
cIsJapan = {}  
cIsJapan[1]=1  
cIsJapan[2]=1  
cIsJapan[3]=0  
cIsJapan[4]=0  
cIsJapan[5]=0
```

**Listing 1.4** Calculating the *probability of Japan as the class* for a document.

```
cAsJapan=0  
for i in range(1,6):  
    if (cIsJapan[i]==1):  
        cAsJapan+=1  
  
ProbOfClass=[0]*2  
ProbOfClass[1]=cAsJapan/5  
ProbOfClass[0]= 1-ProbOfClass[1]
```

Here *cAsJapan* variable is taken to count the number of documents with Japan as class.  
*ProbOfClass* is a list of size 2,  
0th index stores the probability of the document to not have japan as its class, Whereas 1st index stores the probability of the document to have japan as its class.

Probability of class 0(NotJapan) and 1(Japan) has came out to be **0.6** and **0.4** respectively.

### 1.2.4 Terms Occurrences in a class

'TotalCountTokens' is a list of size 2 which stores the total number of tokens in dataset of each class.

'counts' is the dictionary in which counts w.r.t to each term in a class is stored, with key being the term and values being the list of size 2 that.

For every doc firstly checking its class. If its 1(i.e Japan) then adding the tokens count in it to value of TotalCountTokens[1].

UniqueTokens gives total number of unique terms.

#### Listing 1.5 code for forming TotalCountTokens and counts

```
TotalCountTokens=[0]*2
counts={}
for i in range(1,6):
    c=0
    if (cIsJapan[i]==1):
        c=1
        TotalCountTokens[1]+=len(tokens[i])
    else:
        TotalCountTokens[0]+=len(tokens[i])
    for t in tokens[i]:
        if t not in counts:
            counts[t]=[0]*2
        counts[t][c]+=1
UniqueTokens=len(counts)
```

Kyoto	list 2	[0, 2]
London	list 2	[1, 0]
Macao	list 2	[1, 0]
Osaka	list 2	[0, 1]

**Figure 1.2** screenshot of **counts** dictionary. The term 'Kyoto' has occurred twice in class 1(Japan) and 0 times in 'class 0'

### 1.2.5 Parameters Calculation

**condProb[term][c]** is nothing but  $P(\text{term} \mid c)$  i.e probability of term given class  $c$  (here  $c$  can be either 0 or 1). Conditional probability is calculated for only those terms that are present in the test set.

Laplace's smoothing is applied by adding 1 to the numerator and count of unique terms to denominator

**Listing 1.6** calculation of Probability parameters

```
condProb={}
for t,v in counts.items():
    if t in tokens[6]:
        condProb[t]=[0]*2
        for i in range(2):
            laplaceNum=counts[t][i]+1
            laplaceDenom=TotalCountTokens[i]+UniqueTokens
            condProb[t][i]=laplaceNum/laplaceDenom
```

Key	Type	Size	
Kyoto	list	2	[0.071, 0.231]
Taiwan	list	2	[0.214, 0.154]

**Figure 1.3** In the above screenshot of **condProb** dictionary, probability of **Kyoto** given **class 0** is **0.071** and given **class 1** is **0.231**

### 1.3 Construction of Classifier

In the list '**ProbForTest**' index-1 has value as probability of the test set to be class 'Japan' and vice versa at 0th index . Taking maxScore after calculating the score for each class.

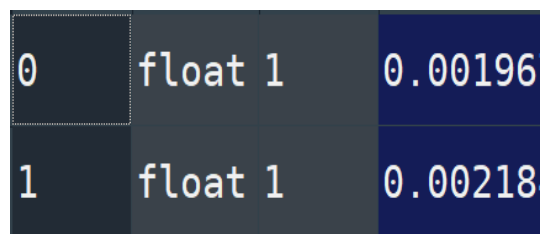
**Listing 1.7** Calculation of scores for each class for testset

```
ProbForTest=[0]*2
for c in range(2):
    score=ProbOfClass[c]
    for t in tokens[6]:
        score*=condProb[t][c]
    ProbForTest[c]=score

maxScore=max( ProbForTest )

Predicted_class=""
if ( ProbForTest.index( maxScore )==1 ):
    Predicted_class="Japan"
else :
    Predicted_class="NotJapan"
```

'**Predicted \_ class**' says which Class is predicted for the test set.



0	float 1	0.00196
1	float 1	0.00218

**Figure 1.4** screenshot of **probForTest** Probability of **class 0** is **0.0196** and for **class 1**(Japan) it is **0.0218**

Since it is maximum for class 1, **Predicted class** is **Japan**.



## 1.4 Evaluation of Classification results

### 1.4.1 Confusion matrix

		True Class	
		Positive	Negative
Predicted Class	Positive	TP	FP
	Negative	FN	TN

**Figure 1.5** A 2\*2 matrix is constructed according to the idea shown in above screenshot

#### Listing 1.8 Construction of confusion matrix

```
confusion_matrix = [ [ 0 for i in range(2) ] for j in range(2) ]
TrueClass="Japan"
if ( Predicted_class==TrueClass and TrueClass=="Japan" ):
    confusion_matrix [0][0]+=1

if ( Predicted_class=="Japan" and TrueClass=="NotJapan" ):
    confusion_matrix [0][1]+=1

if ( Predicted_class=="NotJapan" and TrueClass=="Japan" ):
    confusion_matrix [1][0]+=1

if ( Predicted_class=="NotJapan" and TrueClass=="NotJapan" ):
    confusion_matrix [1][1]+=1
```

0	list 2	[1, 0]
1	list 2	[0, 0]

**Figure 1.6** screenshot of confusion matrix. Except tp all other values (fp,fn,tn) are 0, since there is only 1 test doc and it is predicted correctly.

**Listing 1.9** Assigning values

```
true_positive = confusion_matrix[0][0]
false_positive = confusion_matrix[0][1]
false_negative = confusion_matrix[1][0]
true_negative = confusion_matrix[1][1]
```

### 1.4.2 Precision, Recall, F1-score

Precision, Recall, F1 score are 1.

All these metrics will be 1 when all the positively predicted values are actually positive leaving no actual positive value predicted as negative.

**Listing 1.10** calculating evaluation metrics

```
precision = true_positive / (true_positive + false_positive)

recall    = true_positive / (true_positive + false_negative)

f1_score  = 2*precision*recall / (precision+recall)
```