IRWS PROJECT PART B

PRIYANK SONKIYA 17DCS009

APRIL 2021

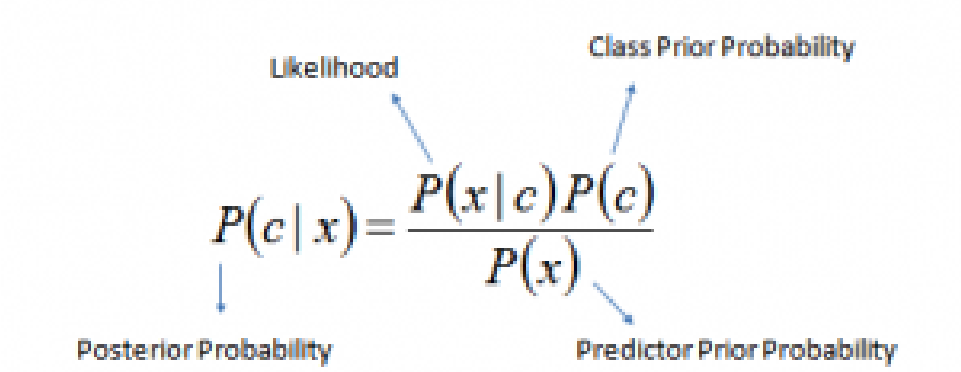
Contents

0.1 INTRODUCTION

This the project report for PART B as a component of the IRWS course even semester 2020- 2021 at The LNM Institute Of Information Technology, Jaipur. A multinomial Naive Bayes Classifier is implemented on the provided data and text classification is done .

0.1.1 THE NAIVE BAYES RULE

The naive bayes rule for text classification looks like -



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Here P(c/x) denotes the posterior probability that is the probability of document x belonging class = c.
P(x/c) signifies the likelihood that with what probabilities tokens of a document are distributed for a class c and finally P(c) refers to the prior probability that is the basic information we have about the system or in other words the prior belief .
We focus on which of the probabilities out two or more is larger in number and accordingly the class label of test data is decided , hence we can ignore the P(x) as that is same for each class and doesn't change.

0.1.2 WHAT WE WANT TO ACHIEVE?

We have to make a model on the basis of our training data and then perform a test case of document - "Taiwan Taiwan Kyoto" and classify it to be present in the "yes" class or the "no" class .

0.2 IMPORTING NECESSARY LIBRARIES AND PACKAGES

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
import nltk
from nltk.tokenize import word_tokenize
import numpy
nltk.download('punkt')
```

The above shown libraries are imported and relevant packages are also imported.

0.3 DATA PREPARATION

```
columns = ['Words in Doc', 'class']
rows = []

rows = [['Kyoto Osaka Taiwan', 'yes'],
        ['Japan Kyoto', 'yes'],
        ['Taipei Taiwan', 'no'],
        ['Macao Taiwan Shanghai', 'no'],
        ['London', 'no']
        ]

training_data = pd.DataFrame(rows, columns=columns)
```

So I have copy pasted the data as it is from the assignment and made it a data frame - training_data with the target variable as class = "yes" or "no".
The data frame looks like -

	Words in Doc	class
0	Kyoto Osaka Taiwan	yes
1	Japan Kyoto	yes
2	Taipei Taiwan	no
3	Macao Taiwan Shanghai	no
4	London	no

0.4 PRIOR PROBABILITIES

As we can see that 2 documents belong to class "yes" and the other 3 belong to class "no" . With this information we can calculate the probabilities - class_yes and class_no -

```
yes_count = 0
no_count = 0
for q in range(len(rows)):
  if rows[q][1] == 'yes':
    yes_count=yes_count+1
  if rows[q][1] == 'no':
    no_count=no_count+1

class_yes = yes_count/len(rows)
class_no = no_count/len(rows)
class_yes , class_no
```

The output is - class_yes = 0.4 and class_no = 0.6
Hence we have prior prior probabilities - 0.4 and 0.6 for the classes "yes" and "no" respectively. The code simply counts the number of yes and number of - "no" and then divides them by the total number of documents.

0.5 TERM DOCUMENT MATRICES

TERM DOCUMENT MATRIX FOR CLASS - YES

```
yes_docs = [row['Words in Doc'] for index,row in training_data.iterrows() if
                                        row['class'] == 'yes']
vec_yes = CountVectorizer()
X_yes = vec_yes.fit_transform(yes_docs)
tdm_yes = pd.DataFrame(X_yes.toarray(), columns=vec_yes.get_feature_names())
tdm_yes
```

TERM DOCUMENT MATRIX FOR CLASS - NO

```
no_docs = [row['Words in Doc'] for index,row in training_data.iterrows() if
                                       row['class'] == 'no']
vec_no = CountVectorizer()
X_no = vec_no.fit_transform(no_docs)
tdm_no = pd.DataFrame(X_no.toarray(), columns=vec_no.get_feature_names())
tdm_no
```

With the help of the above shown code we iterate over the rows of the training data and from the feature extraction module by sklearn we make our term document matrices which look like -

```
tdm_yes
```

	japan	kyoto	osaka	taiwan
0	0	1	1	1
1	1	1	0	0

```
tdm_no
```

	london	macao	shanghai	taipei	taiwan
0	0	0	0	1	1
1	0	1	1	0	1
2	1	0	0	0	0

This term document matrix neither provides any functionality nor solves any dependency in classification problem still its helps in verification of data which we see as we proceed .

0.6 WORD COUNT

We already have information about the prior probabilities and now the goal is to find out the likelihood . And to find the likelihoods we will need to find the word counts or the token counts for the documents so that according likelihood probabilities can be calculated . Following code gives the same for both the word counts for words in "yes" class and also the ones in "no" class -
WORD COUNT FOR CLASS - YES

```
word_list_yes = vec_yes.get_feature_names();
count_list_yes = X_yes.toarray().sum(axis=0)
freq_yes = dict(zip(word_list_yes,count_list_yes))
freq_yes
```

WORD COUNT FOR CLASS - NO

```
word_list_no = vec_no.get_feature_names();
count_list_no = X_no.toarray().sum(axis=0)
freq_no = dict(zip(word_list_no,count_list_no))
freq_no
```

The corresponding outputs are -

```
freq_yes
```

```
{'japan': 1, 'kyoto': 2, 'osaka': 1, 'taiwan': 1}
```

```
freq_no
```

```
{'london': 1, 'macao': 1, 'shanghai': 1, 'taipei': 1, 'taiwan': 2}
```

The feature names (the words) and the count information are extracted from the vectors that made up the term document matrix . Here two lists word_list and count_list are merged to make dictionary elements freq_yes and freq_no containing the words and there counts for both yes and no class .

0.7 DICTIONARY

We know that we have to use LAPLACE smoothing in our classifier hence the unique elements and their count will be a major requirement. A dictionary is made containing all the possible words and the total count or the length of the dictionary is also calculated which is 8.

```
dictionary = list(set(word_list_yes) | set(word_list_no))
dictionary,len(dictionary)
```

And following is the output-

```
dictionary = list(set(word_list_yes) | set(word_list_no))
dictionary,len(dictionary)
```

```
(['taipei',
  'kyoto',
  'taiwan',
  'macao',
  'japan',
  'shanghai',
  'london',
  'osaka'],
 8)
```

This code simply does a union operation on the yes class words and no class words lists and also return the length. For instance here length = 8 which means the dictionary has 8 unique words.

0.8 LIKELIHOOD PROBABILITIES

To determine - P(Xi/yes) and P(Xi/no) where Xi is the set of all the unique words.
Mutual Independence - for a document d containing tokens - X1 , X2 ..... Xn , the probability that document d belongs to class c will be denoted as - P(d/c) = P(X1/c) * P(X2/c) * .....*P(Xn/c) , where n denotes the number of token in a document . This is because of conditional independence . All the probabilities are independent of each other. Log probabilities could have been considered but assuming our system is efficient enough and no human intervention is required we skip taking the log probabilities. The following code performs this -

```
prob_yes = []
for word,count in zip(word_list_yes,count_list_yes):
  prob_yes.append((count+1)/(sum(count_list_yes)+len(dictionary)))
yes_dict = dict(zip(word_list_yes,prob_yes))
```

The yes_dict looks like -

```
yes_dict
```

```
{'japan': 0.15384615384615385,
 'kyoto': 0.23076923076923078,
 'osaka': 0.15384615384615385,
 'taiwan': 0.15384615384615385}
```

These are the likelihood for the tokens that belong to the "yes" class . Both the word lists and count lists as explained above are taken and the probability is calculate by laplace smoothing (see count + 1 in code snippet) . Also the denominator is divided by the length of the dictionary which simply denotes the number of unique words in the training data and the count of the token. Finally with both the word list and the probability list (prob_yes) a dictionary is made containing the token(words) belonging to class "yes" and their corresponding likelihood probability.
But those words(tokens) that do not belong to the "yes" class can also appear in test data and can make our probabilities - 0 . Hence to take care of that with the help of LAPLACE SMOOTHING the following code is written -

```
pro_yesno = []
difference = set(dictionary).symmetric_difference(set(word_list_yes))
yes_no = list(difference)

for word in zip(yes_no , count_list_yes):
  pro_yesno.append(1/(sum(count_list_yes)+len(dictionary)))
yesno_dic = dict(zip(yes_no,pro_yesno))
yesno_dic
```

The yesno_dic dictionary looks like -

```
yesno_dic
```

```
{'london': 0.07692307692307693,
 'macao': 0.07692307692307693,
 'shanghai': 0.07692307692307693,
 'taipei': 0.07692307692307693}
```

These are the words (london , macao , shanghai and taipei) which did not belong to the "yes" . To find their likelihood probabilities by laplace smoothing since there word counts will be 0 for "yes" class , 1 is divided by the sum of no of tokens in yes class and the number of unique words in the dictionary(length of dictionary) which is in accordance with the Laplace smoothing formulae. And as expected all the likelihood probabilities for the tokens not belonging to the yes class are same. Finally we concatenate both the likelihoods of the "yes" class to make a single likelihood for the "yes" class -

```
final_yes_probabilities=yes_dict.update(yesno_dic)
yes_dict
```

The output for the same is -

```
yes_dict
```

```
{'japan': 0.15384615384615385,
 'kyoto': 0.23076923076923078,
 'london': 0.07692307692307693,
 'macao': 0.07692307692307693,
 'osaka': 0.15384615384615385,
 'shanghai': 0.07692307692307693,
 'taipei': 0.07692307692307693,
 'taiwan': 0.15384615384615385}
```

The yes_dict denotes the dictionary for the likelihood for the token of the "yes" class which is probabilities like -
P(taiwan/yes) , P(kyoto/yes) , P(japan/yes) and so on for entire tokens.
Similarly the same is performed for class "no" and the likelihoods are -

```
no_dict
```

```
{'japan': 0.07142857142857142,
 'kyoto': 0.07142857142857142,
 'london': 0.14285714285714285,
 'macao': 0.14285714285714285,
 'osaka': 0.07142857142857142,
 'shanghai': 0.14285714285714285,
 'taipei': 0.14285714285714285,
 'taiwan': 0.21428571428571427}
```

So no_dict contains likelihoods for token belonging to the no class like -
P(taiwan/no) , P(kyoto/no) , P(london/no) and so on for all the tokens.
Now the likelihood and the prior probabilities have been calculated and hence the posterior can be calculated now given the test data.

0.9 TEST DATA

The test data is hard coded and converted into tokens (the same which was done for training data in 1st step of pre-processing) .
After converting to tokens posterior probabilities can be calculated.

```
new_sentence = 'taiwan taiwan kyoto'
new_word_list = word_tokenize(new_sentence)
new_word_list
```

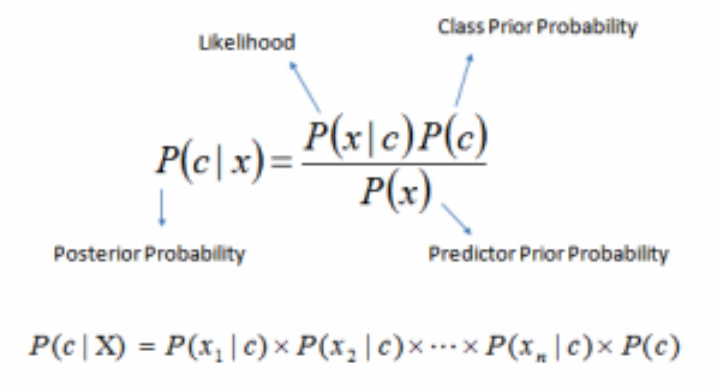
The output looke like -

```
new_word_list
```

```
['taiwan', 'taiwan', 'kyoto']
```

0.10 POSTERIOR PROBABILITIES

For determining the posterior probabilities which will be -
P(yes/test_data) and P(no/test_data) we will use the formulae that is mentioned in section 0.1.1 that is -



P(yes/test_data) =
P(yes)* P(taiwan,taiwan,kyoto/yes)=
P(yes)*P(taiwan/yes)*P(taiwan/yes)*P(kyoto/yes) (due to conditional independence)
Again stressing on the fact that the denominator is not being considered due to the fact that it remains same for both the yes and no probabilities.
All the above calculation is performed by -

```
final_probability_yes = class_yes
for j in range(len(new_word_list)):
  final_probability_yes = yes_dict[new_word_list[j]]*final_probability_yes

final_probability_yes
```

The output is -

```
final_probability_yes
```

```
0.002184797451069641
```

The first line of the above code -
final_probability_yes=class_yes means that the final probability is firstly assigned the prior value which is the class probability of yes = 0.4 , see section 0.4 for class_yes probability . After assigning the prior probability a loop is executed that goes to the likelihood probabilities for class "yes" for each token , fetches the likelihood and multiplied to the existing value of final_probability_yes which is the posterior probability .
So for each token - taiwan , taiwan , kyoto the likelihood probabilities are fetched , multiplied with already the prior assigned and final result is calculated .
The final posterior probability for class "yes" - P(yes/test_data) = 0.002184.
Similarily for class "no" we perform the exact same opearations and find out - P(no/test_data)

```
final_probability_no = class_no
for r in range(len(new_word_list)):
  final_probability_no = no_dict[new_word_list[r]]*final_probability_no

final_probability_no
```

The output looks like -

```
final_probability_no

0.0019679300291545188
```

As shown the posterior probability - P(no/test_data) comes out to be - 0.0019679 Clearly P(yes/test_data) is more than P(no/test_data) but we still implement a if logic for doing this .

```
predicted_data=[];

if(final_probability_no > final_probability_yes):
  print("the test data - Taiwan Taiwan Kyoto belongs to class NO ")
  predicted_data.append(['taiwan taiwan kyoto','no'])
  predicted_data=pd.DataFrame(predicted_data)

else:
  print("the test data - Taiwan Taiwan Kyoto belongs to class YES ")
  predicted_data.append(['taiwan taiwan kyoto','yes'])
  predicted_data=pd.DataFrame(predicted_data)


predicted_data.columns=['TEST DATA','PREDICTED CLASS']
predicted_data
```

In the above snippet the probabilities are compared and the greater probability gets to determine the test class and also the code makes a data frame - predicted_data that will be further used in evaluation for comparison. The data for the same is appended by either of the if / else loop . The final predicted data looks like -

```
the test data - Taiwan Taiwan Kyoto belongs to class YES
```

	TEST DATA	PREDICTED CLASS
0	taiwan taiwan kyoto	yes

0.11 EVALUATION

The same logic is used for creating the true test data , that is the data we want our model to predict . This is for comparison and for accuracy measurement purposes. A similar data frame like predicted_data is made that is - true_data. This is with reference to the true class that was recently updated in problem statement of the assignment. The code is -

```
test_data = [['taiwan taiwan kyoto','yes']]
true_data = pd.DataFrame(test_data)
true_data.columns = ['TEST DATA','TRUE CLASS']
true_data
```

The output is -

```
true_data
```

	TEST DATA	TRUE CLASS
0	taiwan taiwan kyoto	yes

0.11.1 MAKING THE CONFUSION MATRIX

For making the confusion matrix information like - true positives, false positives , true negatives , false negatives are needed. In this case it can be clearly seen that the predicted class is the actual class and there is only one test data so things can be calculated easily . But for generalisation purpose,so that if any data appears this code can perform efficiently the values are calculated as -

```
df =pd.DataFrame()
true_positive = 0
false_positive = 0
true_negative = 0
false_negative = 0
for i in range(len(true_data)):
  for j in range(len(predicted_data)):
   if((true_data['TEST DATA'][i] == predicted_data['TEST DATA'][j] )):
     if(true_data['TRUE CLASS'][i] == 'yes'):
       if(predicted_data['PREDICTED CLASS'][j] == 'yes'):
         true_positive = true_positive+1
       else:
         false_negative = false_negative + 1
     if(true_data['TRUE CLASS'][i] == 'no'):
       if(predicted_data['PREDICTED CLASS'][j] == 'yes'):
         false_positive = false_positive+1
       else:
         true_negative = true_negative + 1

true_positive,false_positive,false_negative,true_negative
```

The output is -

```
true_positive,false_positive,false_negative,true_negative
```

```
(1, 0, 0, 0)
```

In the above code all the confusion matrix parameters are being calculated . The two for loops are for two data frames - the predicted one and the true one. Hence on the basis of test data the class is compared from both the data frames and then accordingly one out of true positive , true negative , false positive and false negative is incremented . This process is calculated for all the test data which in this case is only 1 - taiwan taiwan kyoto. Hence only true_positive = 1 and rest all are 0 as calculated.
Having all the values the confusion matrix is made which looks like -

```
confusion_matrix=[[true_positive,false_positive],[false_negative,
                                        true_negative]]
confusion_matrix = pd.DataFrame(confusion_matrix)
confusion_matrix.columns =['ACTUAL YES CLASS', 'ACTUAL NO CLASS']
confusion_matrix.index = ['PREDICTED TO BE IN YES CLASS', 'PREDICTED TO BE
                                        IN NO CLASS']
```

The output of confusion matrix is -

```
confusion_matrix
```

	ACTUAL YES CLASS	ACTUAL NO CLASS
PREDICTED TO BE IN YES CLASS	1	0
PREDICTED TO BE IN NO CLASS	0	0

Using the below formulae - precision , recall and F1 values are calculated -

```
precision=true_positive/(true_positive+false_positive)
recall = true_positive/(true_positive+false_negative)
F1_value = 2*precision*recall/(precision+recall)

print("THE PRECISION IS:"+str(precision)+"\nTHE RECALL IS :"
+str(recall)+"\nTHE F1 VALUE IS :"+str(F1_value))
```

The output for precision , recall and F-value is -

```
THE PRECISION IS:1.0
THE RECALL IS :1.0
THE F1 VALUE IS :1.0
```

Because there was only a single test data which was classified correctly by the classifier that was implemented, hence precision , recall and f-value all are equal to 1. The project can be concluded now.

0.12 CONCLUSION

Hence we have build a classifier that classifies a particular test data to belong to class "yes" or class "no" .
This implementation may look lengthy but it is due to the fact that nothing is hard coded leaving the training and test data , so if this classifier is given some other data it will classify it correctly . So having successfully implemented the classifier - code - click me ,the project concludes.

0.13 REFERENCES

The following resources were used to complete this project -
1. Classroom Slides.
2. Multinomial Naive Bayes Classifier for Text Analysis click me
3. Implementing a Naive Bayes classifier for text categorization in five steps click me

0.14 A NOTE OF THANKS

A Big Note of Thanks to our instructors, MRS. Preety Singh mam and Mrs Suvidha Tripathi mam who constantly guided us and maintained effective communication through emails and classroom sessions in this pandemic.
I am truly grateful to have teachers like you. Thank you so much.

PRIYANK SONKIYA
17DCS009