CMPE 561
NATURAL LANGUAGE PROCESSING

YAŞAR ALİM TÜRKMEN
INSTRUCTOR: TUNGA GÜNGÖR

NAIVE BAYES BASED NEWS ARTICLE CLASSIFIER
PROGRAMMING PROJECT

11.12.2019

INTRODUCTION

In this project, a Naive Bayes text classifier is implemented. We are given news articles with three classes: movie, theater and sports. A system is developed which reads the news articles and automatically groups these articles into regarding topics.

Like in any machine-learning task, the dataset is split into two as training and test tests. 41 of 49 news articles are used in training and the rest is used in testing the performance of the model. There are 13 movie, 16 theater and 12 sports related articles in the training data. 2 movie, 3 sports and 3 theater related articles are used in test step.

Naive Bayes algorithm exploits the probability of occurrences of words. Therefore, the word occurrences are counted and they are weighted with respect to given weighting algorithm. Then, using the word counts the class probability scores calculated for each document in the test. The class with the maximum score is selected as the topic of that document.

PROGRAM INTERFACE

The program is written in Python3 and os, collections, math and sys modules are used. Therefore, you need 3 things to run this program:

-Python 3 (any version of it)

-A terminal program in order to run the code

-Os, collections, math and sys modules. Probably these modules will come with Python3. You will not need to install them manually.

After you install python3, the modules and a terminal program you can simply run the code with:

python3 naive_bayes.py [TRAINING_PATH] [TEST_PATH] [USE_STOPWORDS] [WEIGHTING] [THRESHOLD] [WRITE_TO_FILE]

For the details of these arguments see **Program Execution Part.**

In order to terminate the program press Ctrl+c.

PROGRAM EXECUTION

After you run the program with the desired arguments Naive Bayes algorithm will automatically trains itself and test the documents. The arguments that should be given are:

TRAINING_PATH = Path of the training files

TEST_PATH = Path of the test files

USE_STOPWORDS = Eliminate stopwords or not in processing (True or False)

WEIGHTING = The weighting algorithm (‘tf’ for term-frequence, ‘binary’ for binary and ‘tf-idf’ for term frequency-inverse document frequency)

THRESHOLD = The threshold for counting the word occurences (Give number)

WRITE_TO_FILE = Write the output of training to a file or not (True or False)

If you give missing arguments the program executes with default ones.

Default arguments:
TRAINING_PATH = <working directory>/articles/training/
TEST_PATH = <working directory>/articles/test/
USE_STOPWORDS = True
WEIGHTING = 'tf'
THRESHOLD = None
WRITE_TO_FILE = False
After the process is done it gives the output.

INPUT OUTPUT

As stated before, this program needs arguments while running. It prints some information about the corpus, given parameters and the model's performance. If the user sets WRITE_TO_FILE to True, then word counts for each class, total word counts and document counts will be written to a txt file for further usage.

PROGRAM STRUCTURE

Modules:

In the project 4 modules are used: **os, sys, collections,** and **math**.

Sys module is used for taking the arguments from the user while executing the program. Also, using this module you can exit the program with an error message.

Os is used for getting the 'working directory' and the file list from the directory.

Collections module enables us to count occurrences of words and create a set of words and number of occurrences.

Math module is used for calculating the logarithm of probabilities.

Global variables:

Stopwords is the list of stopwords. This list is taken from https://github.com/ahmetax/trstop/blob/master/dosyalar/turkce-stop-words. Using this list, stopwords in the documents are eliminated.

The arguments taken from the user while execution is starting are also used as global variables. The definitions of them are given above.

The program can be separated into 4 parts: reading and preprocessing the documents, training, test and evaluation.

Reading and Preprocessing Documents

In this part both the training and the test files are read and preprocessed before calculations. Then preprocessed documents and the regarding labels are put in lists and returned.

remove_punctuation

In this function all of non alpha-numeric characters are cleaned from the text.

read_file

Here, first the files extracted from given directory. According to their names' first characters, they are labeled with 0 for movie, 1 for theater and 2 for sport. The given files start with the first character of the class they belong to.

Then for each document a loop runs. First, the punctuation is removed from the document using remove_punctuation function. After the document is changed to lowercase, it is split into words and stopwords are cleaned. In other words, each news article is represented as a list of words.

Training

The Naive Bayes is trained using given weighting algorithms and threshold. The main function is **train**. In this function, weightings are calculated and **word_counts, total_word_counts, document_count** variables are created and returned. **Word_count** stores the occurrences of each word in each class whereas **total_word_counts** counts total frequency of each word. In **document_count** variable number of documents in each class is stored.

The processes are divided into three according to given weighting. For 'tf' just the frequency of each word in a class and in the corpus are counted. 'tf-idf' weighting needs also inverse-document-frequencies. 'tf-idf' is calculated by **calc_tf_idf** function. This function needs inverse-document-frequencies for calculation which is found by **inverse_frequency** function. For 'binary' weighting, just the presence of words are used, they are not counted.

If a threshold is given, only the words with a frequency greater than the threshold are counted.

Test

Testing is done and results are obtained in this part. It uses the training step's resultant variables are used. The probability of a document belonging to a class is calculated here. The tokenized test documents are given to **test** function and **predictions** are returned.

For each word in a document, first P(word|class) is calculated and the logarithm of this probability is added to logarithm of P(class). For unknown words Laplace Smoothing is used and by using logarithm of probabilities getting 0 probabilities for each class is avoided. Finally, the class with maximum probability score is taken for each document.

Evaluation

Evaluation of the performance of the model is shown with macro-average and micro-average precision, recall and F-1 scores. After test step, the confusion matrix is created using predicted and true labels in **create_confusion_matrix** function. This confusion matrix is used for macro-average and micro-average metric calculations which are found by **calc_macro_metrics** and **calc_macro_metrics** functions.

EXAMPLES

```
alim@alim-CX61-2PC:~/Desktop/NLP/Assignment2$ python3 naive_bayes.py ./articles/training/ ./articles/test/ True 'binary' 0 False
------------

PARAMETERS
------------

Weighting: binary
Threshold: 0
Clean stopwords: True
Number of features:4793
Number of unseen words: 594
------------

RESULTS
------------

True labels: [0, 2, 1, 2, 0, 1, 1, 2]
Predicted labels: [0, 2, 1, 2, 2, 1, 1, 2]
------------

EVAULATION
------------
micro-average scores: {'precision': 0.875, 'recall': 0.875, 'f1': 0.875}
macro-average scores: {'precision': 0.9166666666666666, 'recall': 0.8333333333333334, 'f1': 0.8730158730158729}
  Given missing arguements...
  Using default values...
  ------------

  PARAMETERS
  ------------

  Weighting: tf
  Threshold: None
  Clean stopwords: True
  Number of features:4793
  Number of unseen words: 594
  ------------

  RESULTS
  ------------

  True labels: [0, 2, 1, 2, 0, 1, 1, 2]
  Predicted labels: [0, 2, 1, 2, 2, 1, 1, 2]
  ------------

  EVAULATION
  ------------
  micro-average scores: {'precision': 0.875, 'recall': 0.875, 'f1': 0.875}
  macro-average scores: {'precision': 0.9166666666666666, 'recall': 0.8333333333333334, 'f1': 0.8730158730158729}
```

EVALUATION

The Naive Bayes model is tested with different parameters. The scores are listed below. Since the test set is very few in number it is quite hard to make comprehensive comments about them. However, it can be observed that ‘tf-idf’ gives the best results, 1.0 in best setup, and it is affected by stopword usage less than ‘tf’ weighting. The threshold usage decreases the performance of the models in particular when stopwords are not cleaned. ‘Binary’ weighting obtains very low scores when there is a threshold, as expected. Moreover, after a point some setups give worse results than random. On the other hand, threshold decreases the number of features a lot. For larger datasets, usage of a threshold is practical and sometimes we must give a threshold to make the model run faster.

When the predictions are observed, there is no obvious confusion among classes. The parameter setups don’t do the same mistake. For example, when weighting=’tf-idf’ and threshold=1 the model misclassifies different document than when weighing=’tf’ and threshold=0. Due to data size, it can’t be concluded that there is a confusion between some classes.

Parameters			Micro-average			Macro-average			Number of features
Clean Stopwords	Weighting	Threshold	Precision	Recall	F-1	Precision	Recall	F-1	

Yes	tf	0	0.875	0.875	0.875	0.916	0.833	0.873	4793
No	tf	0	0.875	0.875	0.875	0.916	0.833	0.873	4977
Yes	tf	1	0.875	0.875	0.875	0.916	0.833	0.873	1382
No	tf	1	0.875	0.875	0.875	0.916	0.833	0.873	1529
Yes	tf	2	0.75	0.75	0.75	0.866	0.722	0.787	673
No	tf	2	0.75	0.75	0.75	0.866	0.722	0.787	789
Yes	tf	3	0.625	0.625	0.625	0.833	0.611	0.705	407
No	tf	3	0.625	0.625	0.625	0.833	0.611	0.705	494
Yes	tf	4	0.625	0.625	0.625	0.833	0.611	0.705	296
No	tf	4	0.5	0.5	0.5	0.142	0.5	0.222	372
Yes	binary	0	0.875	0.875	0.875	0.916	0.833	0.873	4793
No	binary	0	0.75	0.75	0.75	0.866	0.722	0.787	4977
Yes	binary	1	0.375	0.375	0.375	0.125	0.333	0.181	554
No	binary	1	0.375	0.375	0.375	0.125	0.333	0.181	676
Yes	binary	2	0.375	0.375	0.375	0.0	0.333	0.0	103
No	binary	2	0.375	0.375	0.375	0.0	0.333	0.0	167
Yes	**tf-idf**	**0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**4793**
No	**tf-idf**	**0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**4977**
Yes	tf-idf	1	0.875	0.875	0.875	0.916	0.833	0.873	1382
No	tf-idf	1	0.875	0.875	0.875	0.916	0.833	0.873	1529
Yes	tf-idf	2	0.875	0.875	0.875	0.916	0.833	0.873	673
No	tf-idf	2	0.75	0.75	0.75	0.866	0.722	0.787	789
Yes	tf-idf	3	0.875	0.875	0.875	0.916	0.833	0.873	407
No	tf-idf	3	0.75	0.75	0.75	0.866	0.722	0.787	494
Yes	tf-idf	4	0.875	0.875	0.875	0.916	0.833	0.873	296
No	tf-idf	4	0.75	0.75	0.75	0.866	0.722	0.787	372

IMPROVEMENTS AND EXTENSIONS

The project is designed for this three class news articles classification task. Therefore, this implementation doesn't work for greater class numbers. Most of the parts in the implementation is done in a more generic way, however there are left some task-spesific parts. Those parts can be reimplement for a generic Naive Bayes text classifier.

The normalization process can be studied which is very important for frequency based algorithms. Stemming and lemmatization would increase the success of the algorithm especially for Turkish corpus due to its agglutinative structure. Also, the numbers are left as raw tokens which affects the performance negatively.

For the unseen words, Laplace-smoothing is used. However, there are better smoothing algorithms. Hence, in order to increase the performance of the model a better smoothing algorithm

can be used. Also in this project, a simple unigram approach is used, however, interpolation of different n-grams can give better results.

DIFFICULTIES ENCOUNTERED

Naive Bayes is not a complicated model. However, designing the whole architecture is not easy enough. Furthermore, it is difficult to follow the model's steps since the text files are big. For example, at the beginning I just use probability values. Then, I realized that the probabilities goes to zero after a point. I noticed that problem after examining other parts and calculations. I think, those type of problems can go unnoticed.

CONLUSION

In this project news articles with 3 different topics are classified with Naive Bayes. In Naive Bayes design Laplace-smoothing is used and the logarithm of the probabilities are calculated in order to avoid 0 probabilities.

The model is tried with different weighting algorithms. The best weighting algorithm is 'tf-idf' which gets 1.0 precision, recall and F-1 scores when stopwords are cleaned and no threshold is set. There is a negative correlation between number of features and the success of the model. Similarly threshold is negatively correlated with the performance.

This model can be fed with more data for better performance in a more realistic scenario. Also this project can be simply applied for other classification tasks. Some small changes is enough to make it more generic.

APPENDIX

```
from os import listdir
from os.path import isfile, join
import os
from collections import Counter
import math
import sys
try:
TRAINING_PATH = sys.argv[1] #Path of the training files
TEST_PATH = sys.argv[2] #Path of the test files
USE_STOPWORDS = sys.argv[3] #Eliminate stopwords or not in processing
if USE_STOPWORDS == 'False':
USE_STOPWORDS = False
WEIGHTING = sys.argv[4] #The weighting algorithm
THRESHOLD = int(sys.argv[5]) #The threshold for counting the word occurences
WRITE_TO_FILE = sys.argv[6] #Write the output of training to a file or not
if WRITE_TO_FILE == 'False':
WRITE_TO_FILE = False
```

```
except :
print("Given missing arguements...\nUsing default values...")
pwd = os.getcwd()
TRAINING_PATH = pwd+'/articles/training/'
TEST_PATH = pwd+'/articles/test/'
USE_STOPWORDS = False
WEIGHTING = 'tf'
THRESHOLD = 4
WRITE_TO_FILE = False

stopwords = open('stopwords.txt','r').read().split('\n') #The stopwords list

# Removes punctuations and other non alpha-numeric characters.
def remove_punctuation(text):
text = text.replace('\'', ' ').replace('\n', ' ').replace('.', '').replace(',', '').replace('*',
'').replace('/', ' ')
text = text.replace('?', '').replace('(', '').replace(')', '').replace(':', '').replace(';', '').replace('’',
' ')

return text

# Reads the all of the files in the given directory.
# Stopwords are eliminated from the corpus according to user's choice.
# The docs in the files are read and put into a list.
# Regarding labels are also kept in a list and these are returned.
def read_file(path, use_stopwords=True):
global stopwords

files = [f for f in listdir(path,) if isfile(join(path, f))]
labels_set = {'m':0, 't':1, 's':2} #movie:0, theatre:1, sport:2
articles = []
labels = []
vocab_set = []

for file in files:
file_path = path+file
article_raw = open(file_path, 'r').read()
article_raw = remove_punctuation(article_raw)
article_raw=article_raw.lower().split(' ')
article_processed = []

for word in article_raw:
if use_stopwords:
if word not in stopwords:
if word not in vocab_set:
vocab_set.append(word)
article_processed.append(word)
else:
if word not in vocab_set:
vocab_set.append(word)
article_processed.append(word)
```

```
articles.append(article_processed)
labels.append(labels_set[file[0]])

return articles, labels, vocab_set

# Calculates the inverse-document-frequencies of the words in given document list.
def inverse_frequency(x):
number_of_documents = len(x)
word_inv_freq = {}

for xi in x:
x_counter = Counter(xi)
for word in x_counter.keys():
if word in word_inv_freq:
word_inv_freq[word] += 1
else:
word_inv_freq[word] = 1
for word in word_inv_freq.keys():
word_inv_freq[word] = number_of_documents/word_inv_freq[word]

return word_inv_freq

# Takes tf and idf to calculate tf-idf weighting of words in the corpus.
def calc_tf_idf(inverse_frequency, word_counts):

for clss in range(len(word_counts)):
for word in word_counts[clss]:
word_counts[clss][word] = word_counts[clss][word]*inverse_frequency[word]

return word_counts

# Trains the naive bayes classifier using the given weighting algorithm.
# Returns word counts of classes, overall word counts and document counts.
def train(x, y, threshold=None, weighting='tf'):
number_of_classes = len(set(y))
training_matrix = {} # Set of all articles with their words in beloning classes
word_counts = [] # Word occurence list among classes
document_count = {} # Set of document counts in belonging classes

for i in range(0, number_of_classes):
training_matrix[i] = []
document_count[i] = 0
word_counts.append({})

for idx, xi in enumerate(x):
training_matrix[y[idx]].extend(xi)
document_count[y[idx]] += 1

if weighting == 'tf' or weighting == 'tf-idf':
```

```
total_word_count = []
for i in range(0, number_of_classes):
word_counts[i] = Counter(training_matrix[i])
total_word_count.extend(training_matrix[i])
total_word_counts = Counter(total_word_count)
if threshold:
temp_word_counts = [{}, {}, {}]
temp_total_word_counts = {}
for word in total_word_count:
if total_word_counts[word] > threshold:
temp_total_word_counts[word] = total_word_counts[word]
if word in word_counts[0].keys():
temp_word_counts[0][word] = word_counts[0][word]
if word in word_counts[1].keys():
temp_word_counts[1][word] = word_counts[1][word]
if word in word_counts[2].keys():
temp_word_counts[2][word] = word_counts[2][word]
total_word_counts = temp_total_word_counts
word_counts = temp_word_counts

if weighting == 'tf-idf':
inv_freq = inverse_frequency(x)
word_counts = calc_tf_idf(inv_freq, word_counts)


elif weighting == 'binary':

total_word_counts = {}
for i in range(0, number_of_classes):
for word in training_matrix[i]:
if word not in word_counts[i].keys():
word_counts[i][word] = 1
if word not in total_word_counts.keys():
total_word_counts[word] = 1
else:
total_word_counts[word] = total_word_counts[word] + 1

if threshold:
temp_word_counts = [{}, {}, {}]
temp_total_word_counts = {}
for word in total_word_counts.keys():
if total_word_counts[word] > threshold:
temp_total_word_counts[word] = total_word_counts[word]
if word in word_counts[0].keys():
temp_word_counts[0][word] = word_counts[0][word]
if word in word_counts[1].keys():
temp_word_counts[1][word] = word_counts[1][word]
if word in word_counts[2].keys():
temp_word_counts[2][word] = word_counts[2][word]
total_word_counts = temp_total_word_counts
word_counts = temp_word_counts
```

```
else:
print("Given unrecognized weighting. It can be 'tf', 'binary' or 'tf-idf'")
sys.exit(1)
return word_counts, total_word_counts, document_count

# Tests the trained model with the fiven files.
def test(x, y, word_counts, total_word_count, document_count):

predictions = [] # Keeps the max-probability results of given documents
number_of_types = len(total_word_count.keys())
number_of_docs = sum(document_count.values())
unseen_words = []

for xi in x:
probs = [] # List of all classes' scores
for i in document_count.keys():
p_word_d = 1.0 # p(w|d)
for word in xi:
if word not in total_word_count.keys():
unseen_words.append(word)
if word in word_counts[i].keys():
p_word_d += math.log((word_counts[i][word]+1)/(sum(word_counts[i].values())
+number_of_types))
else:
p_word_d += math.log((1.0)/(sum(word_counts[i].values())+len(total_word_count.keys())))
p_d = math.log(document_count[i]/number_of_docs) #p(d)
probability = p_d + p_word_d
probs.append(probability)
predictions.append(probs.index(max(probs)))
return predictions, set(unseen_words)

# Creates confusion matrix of the model's results.
def create_confusion_matrix (predictions, y):
number_of_classes = len(set(y))
confusion_matrix = []
for i in range(number_of_classes):
new_row = []
for j in range(number_of_classes):
new_row.append(0)
confusion_matrix.append(new_row)
for index in range(len(y)):
confusion_matrix[predictions[index]][y[index]] += 1
return confusion_matrix

# Calculates macro-average precision, recall and f1 scores.
def calc_macro_metrics(predictions, y):
confusion_matrix = create_confusion_matrix(predictions, y)
number_of_classes = len(set(y))
macro_metrics = {}
precision, recall = 0, 0
```

```
for i in range(number_of_classes):
sum_precision = 0
sum_recall = 0
for j in range(number_of_classes):
sum_precision += confusion_matrix[i][j]
sum_recall += confusion_matrix[j][i]
if sum_precision != 0:
precision += confusion_matrix[i][i]/sum_precision
else:
precision = 0
if sum_recall != 0:
recall += confusion_matrix[i][i]/sum_recall
else:
recall = 0

precision = precision/3
recall = recall/3
macro_metrics['precision'] = precision
macro_metrics['recall'] = recall
if precision+recall != 0:
macro_metrics['f1'] = 2*recall*precision/(precision+recall)
else:
macro_metrics['f1'] = 0

return macro_metrics

# Calculates micro-average precision, recall and f1 scores.
def calc_micro_metrics(predictions, y):

confusion_matrix = create_confusion_matrix(predictions, y)
number_of_classes = len(set(y))
micro_metrics = {}
precision, recall = 0, 0
sum_precision = 0
sum_recall = 0
for i in range(number_of_classes):
for j in range(number_of_classes):
sum_precision += confusion_matrix[i][j]
sum_recall += confusion_matrix[j][i]
precision += confusion_matrix[i][i]
recall += confusion_matrix[i][i]

if sum_precision != 0:
precision = precision/sum_precision
else:
precision = 0
if sum_recall != 0:
recall = recall/sum_recall
else:
recall = 0
```

```
micro_metrics['precision'] = precision
micro_metrics['recall'] = recall
if (precision+recall) != 0:
micro_metrics['f1'] = 2*precision*recall/(precision+recall)
else:
micro_metrics['f1'] = 0

return micro_metrics

x_train,y_train,_ = read_file(TRAINING_PATH, use_stopwords=USE_STOPWORDS)
word_counts, total_word_count, document_count = train(x_train, y_train,
threshold=THRESHOLD, weighting=WEIGHTING)

if WRITE_TO_FILE:
file_word_count = open('word_counts.txt', 'w')
file_word_count.write(str(word_counts))
file_total_word_count = open('total_word_counts.txt', 'w')
file_total_word_count.write(str(total_word_count))
file_doc_count = open('doc_counts.txt', 'w')
file_doc_count.write(str(document_count))

x_test,y_test,_ = read_file(TEST_PATH, use_stopwords=USE_STOPWORDS)
predictions, unseen_words = test(x_test, y_test, word_counts, total_word_count,
document_count)

micro = calc_micro_metrics(predictions, y_test)
macro = calc_macro_metrics(predictions, y_test)

print('__________\n')
print('PARAMETERS')
print('__________\n')
print('Weighting: '+ WEIGHTING)
print('Threshold: ' + str(THRESHOLD))
print('Clean stopwords: ' + str(USE_STOPWORDS))
print('Number of features:' + str(len(total_word_count.keys())))
print('Number of unseen words: '+ str(len(unseen_words)))
#print('Unseen words: '+ str(unseen_words))
print('__________\n')
print('RESULTS')
print('__________\n')
print('True labels: '+str(y_test))
print('Predicted labels: ' + str(predictions))
print('__________\n')
print('EVAULATION')
print('__________')
print('micro-average scores: ' + str(micro))
print('macro-average scores: ' + str(macro))
```