CMPE561
Natural Language Processing Course
Homework-1 Report

Samet Atdag[1]

[1]*Department of Computer Engineering, Bogazici University, Istanbul*

(Dated: March 18, 2016)

In this homework, a Naive-Bayes classifier is implemented to detect the authorship of column articles. 69 Authors dataset is used for training and test processes. The performance of the author identification system is done using precision, recall and f1-measure values. (The code of the project is in this Github account.)

Contents

I. INTRODUCTION

In most the cases, detection of authorship is a fairly easy task for humans. On the other hand, vastness of the texts and number of columnists makes the process almost impossible for human evaluation. Since machines have fairly faster text process capabilities, we can take advantage of text classification methods for this specific problem. Naive Bayes is a method used for probabilistic classification and reaches impressive results in several cases.

In this study, a Naive Bayes text classifier is implemented to recognize the authors of column articles. The purpose is building and training a text classifier to predict an article's author with a certain performance.

II. DATASET

69 Authors dataset is a set of articles which includes 910 articles from 69 Turkish authors. Each author has 10 to 30 articles.

III. PRE-PROCESSING AND TOKENIZATION

Before training the Naive Bayes classifier, training and test sets are built due to 60% ratio: 60% of files belonging to each author is taken as training set. The rest is used as test set. For example, there are 10 articles that belong to author gulseBirsel. Randomly selected 6 articles out of these 10 articles are placed in the training set. The remaining 4 articles are put into the test set.

Number of files for each author, number of files used for training and test sets are given in Table I.

A. Tokenization

For building a Naive Bayes classifier, a Bag of Words approach is used. To achieve this, a tokenisation method is required. The steps below are applied to the training set files for tokenisation:

1. Punctuation removal: Punctuation characters might be use to recognize the author of the text, but for building a baseline recognizer, I decided to omit them at all. To remove those characters, I cleared the characters which can be find in Python's built-in string.punctuation variable. The characters in the string.punctuation variable can be seen in Fig.I.

2. Then all text is converted to lower case.

3. As the last step, text is splitted into words using Python's regex package via *W+* imperative. *W+* ignores newlines and splits sentences into words.

FIG. 1: Fig.I. Punctuation characters in the string.punctuation built-in.

! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { | } ~

Author	# of files	# of training files	# of test files
abbasGuclu	15	9	6
abdullahAymaz	10	6	4
ahmetAltan	10	6	4
ahmetHakan	15	9	6
aliBulac	10	6	4
atillaDorsay	15	9	6
ayseArman	15	9	6
balcicekPamir	30	18	12
bekirCoskun	10	6	4
bulentKorucu	10	6	4
canDundar	10	6	4
cemilErtem	10	6	4
cemSuer	10	6	4
cengizCandar	10	6	4
cetinAltan	30	18	12
deryaSazak	10	6	4
doganHizlan	15	9	6
eceTemelkuran	10	6	4
ekremDumanli	30	18	12
elifSafak	10	6	4
emreAkoz	30	18	12
emreKongar	15	9	6
ergunBabahan	10	6	4
ertugrulOzkok	10	6	4
fatihAltayli	30	18	12
fehmiKoru	10	6	4
fikretBila	10	6	4
fundaOzkan	10	6	4
gulseBirsel	10	6	4
guneriCivaoglu	10	6	4
hakkiDevrim	15	9	6
hasanCemal	10	6	4
hasanPulur	10	6	4
hasmetBabaoglu	10	6	4
hekimogluIsmail	10	6	4
hincalUluc	10	6	4
huseyinGulerce	10	6	4
ismetBerkan	10	6	4
mahfiEgilmez	15	9	6
mehmetaliBirand	10	6	4
mehmetBarlas	30	18	12
mehmetOz	15	9	6
mehmetTezkan	10	6	4
melihAsik	10	6	4
mumtazerTurkone	10	6	4
muratBardakci	10	6	4
muratBelge	10	6	4
muratYetkin	10	6	4
nazliIlicak	10	6	4
nihalKaraca	10	6	4
nurayMert	10	6	4
omerUrundul	10	6	4
oralCalislar	10	6	4
raufTamer	10	6	4
rehaMuhtar	10	6	4
ridvanDilmen	15	9	6
ruhatMengi	15	9	6
samikohen	30	18	12
savasAy	10	6	4
serpilYilmaz	10	6	4
tahaAkyol	30	18	12
tamerKorkmaz	10	6	4
tarhanErdem	10	6	4
umurTalu	10	6	4
yaseminCongar	10	6	4
yigitBulut	10	6	4
yilmazOzdil	10	6	4
yukselAytug	15	9	6
zekiCol	10	6	4

TABLE I: Properties of 69 Authors dataset, training set and test sets.

IV. AUTHOR IDENTIFICATION SYSTEM

To recognize the author of a new document, a Naive Bayes classifier is implemented. All the implementation is done from scratch; no ready libraries are used.

A. Training Model

For the model, Bag of Words approach and word counts are used. Word counts are used as features. For each class in the training set:

- Read all files belonging to the class
- Tokenize files
- Count the words
- Add word counts to global vocabulary
- Add word counts to trained model-class

B. Testing a Document

When training phase is finished, testing a new file is possible. For a given new document;

1. Read the document
2. Tokenize the text
3. Count the words
4. For each word, calculate Bayesian Probability of belonging to class C_i.
5. Multiply probabilities for each word
6. Multiply the result with the prior of C_i
7. Find the largest $P_C i$ as the prediction.

There are two notable properties: First, in testing, Laplace Smoothing is used. Second; instead of multiplying the probabilities, they are moved into log scale and log values are summed to prevent underflow errors.

C. Extended Model

To improve the performance of the classifier, external complexity features can be used. In this study, Function Words (FWs) are used. 620 Turkish Function words are taken from the text2arff.zip file which lives on this link.

For this extension, counts of the FWs are added to the training model as features. This process basically enriches the Bag of Words model with external features. When testing FWs are used as they are an internal part of the training set.

V. PERFORMANCE EVALUATION

In this section, the performance of the classifier is given. Precision, recall and F1 scores are used for performance evaluation.

A. Basic Model Results

Micro and Macro Averaged Scores:

	Precision	Recall	F1-Score
Micro Averaged	1.4492%	21.9780%	0.0271%
Macro Averaged	1.4492%	12.6811%	0.0256%

TABLE II: Bow Model - Micro and Macro Averaged Scores

B. Extended (BoW + FWs) Model Results

Micro and Macro Averaged Scores:

	Precision	Recall	F1-Score
Micro Averaged	1.4492%	21.9780%	0.0271%
Macro Averaged	1.4492%	12.6811%	0.0256%

TABLE III: Extended (BoW + FWs) Model - Micro and Macro Averaged Scores

VI. COMMENTS ON THE RESULTS

There are two remarkable observations in this experiment. First of all, the overall performance is particularly weak for such a task. This may take us to the decision of creating a larger training set, for example more articles per author; or changing the classifier. For datasets including small number of instances, there are better classification tools such as Support Vector Machines.

Secondly, adding FWs to extend the method does not affect the performance. After investigation, I observed that number of FWs (620) are not enough for such kind of task It would be better to use a large number of Function Words.

VII. CONCLUSION

To conclude, in this study, I learned how to implement a Naive Bayes classifier for author detection. Also I used Laplace Smoothing and additionally, I extend the method via using Function Words. I evaluated the results using precision, recall and f1 scores.