# Applying a probabilistic algorithm to spam filtering

Olga V. Okhlupina, Dmitry S. Murashko

***Abstract*— Among the common methods of combating spam, a special place is occupied by a probabilistic machine learning algorithm, which is based on the well-known Bayes theorem. The so-called "naive" Bayesian classifier establishes the class of the document by determining the a posteriori maximum. With the development of machine learning methods, the Bayesian algorithm has not lost its relevance and continues to be very popular for solving a large number of tasks, including spam detection. The main advantages of this classifier are simplicity, fast learning, fairly high accuracy, reliability. The paper considers the solution of the problem of determining spam messages using a probabilistic machine learning algorithm. The mathematical justification and implementation of the Bayesian algorithm on a concrete example using program code in the Python programming language is given.**

***Keywords*— Spam, filtering, probabilistic algorithm, Bayes formula, a posteriori probability, conditional probability, machine learning, class, classifier, training.**

## I. INTRODUCTION

The use of approaches and methods implemented in artificial intelligence systems makes it possible to significantly improve the efficiency of solutions to most practical problems compared to traditional approaches. The human experience of spam detection plays an important role in the fight against such mailings due to the unconventionality and high efficiency. However, optimization of the filtration process and improvement of its mechanisms is becoming increasingly relevant.

There are a number of spam detection methods that have their advantages and disadvantages in terms of meeting the necessary criteria of simplicity, trainability and reliability, minimizing false conclusions.

In the first part of the paper, the mathematical basis of the probabilistic Bayesian algorithm is given. The second part is devoted to the implementation of an algorithm for detecting and filtering spam messages with a demonstration of program code in the Python programming language.

## II. BAYESIAN ALGORITHM AND FILTERING

The probable Bayesian algorithm is based on the use of the well-known Bayes theorem [1], which is closely related to conditional probabilities. This algorithm refers to machine learning algorithms.

We introduce the following notation. Let $X$ - be a set of objects, $Y$ - is a set of classes (finite). The probability space $X \times Y$ has a density of $p(x,y) = P(y)\,p(x|y)$, $P(y)$ - a priori probabilities of the appearance of objects of each class, $p_y(x) = p(x|y)$ - class distribution densities (likelihood functions). $\alpha : X \to Y$ - is the filtering algorithm. $\beta_{yk}$ - losses when assigning a class object to a class $k$ ($\beta_{yy} = 0$, $\beta_{yk} > 0$, $y \neq k$).

When detecting spam: $y = 1$ - spam, $y = 0$ - not spam, $\beta_{01} > \beta_{10}$ (that is, the loss when passing spam is less of a loss than false detection).

If we assume that the losses are determined only by the true classification of the object, and not by the class to which it was mistakenly assigned, then $\beta_{yk} \equiv \beta_y$, $\forall y,k \in Y$.

With a priori probabilities $P(y)$ and likelihood functions $p_y(x)$, $\beta_{yk} \equiv \beta_y$, $\forall y,k \in Y$, $\beta_{yy} = 0$, the average risk is minimized thanks to the algorithm $\alpha(x) = \arg\max_{y \in Y} \beta_y P_y p_y(x)$.

According to the definition of conditional probability, we have: $p(x,y) = p_y(x) P_y = P(y|x)\,p(x)$. The conditional probability $P(y|x)$ is the a posteriori probability of a class $y$ for an object $x$. To calculate it, we apply the Bayes formula: $P(y|x) = \dfrac{p(x,y)}{p(x)} = \dfrac{p_y(x) P_y}{\sum\limits_{k \in Y} p_k(x) P_k}$.

With the help of a posteriori probability , the algorithm will take the form: $\alpha(x) = \arg\max_{y \in Y} \beta_y P(y|x)$.

Under the condition of equivalence of classes $\beta_y \equiv 1$ we are talking about the maximum $P(y|x)$. If the classes are equally probable, then the object $x$ belongs to the class with the highest distribution density, $\alpha(x) = \arg\max_{y \in Y} p_y(x)$.

For example, a mail system that has been trained on a certain number of incoming emails (belonging to two classes: spam and non-spam) needs to attribute the following message to one of the classes considered during training.

It is believed that the words in the letter do not depend on each other. The use of the so-called "naive" Bayesian algorithm is associated with the assumption of independence

Manuscript received February 19, 2022.

Olga V. Okhlupina, Ph.D., Associate Professor, Bryansk State University of Engineering and Technology, Bryansk, Russia (e-mail: helga131081@yandex.ru )

Dmitry S. Murashko, student, Bryansk State University of Engineering and Technology, Bryansk, Russia (e-mail: murashko100500@gmail.com)

and equal possibility of all the parameters under consideration. It should be noted that these assumptions, which are not entirely correct in practice, justify themselves in practical application. Hence the "naivety" of the algorithm.

$P(y|x)$ calculated by the Bayes formula for each class (by creating frequency tables for all objects (relative to the desired result), from which likelihood tables are created). The class with the highest $P(y|x)$ and is the desired one.

## III. Algorithm implementation

Here is an example of the implementation of the Bayesian algorithm.

Let the system be offered the following messages as a training sample (see Table 1):

Table 1

| Spam | Not spam |
|---|---|
| Laptops at a bargain price | There will be a conference tomorrow |
| Sale! Order a bike and get headphones as a gift | Order skates and a bike |

As a spam message to be checked, we will select the following message: "Skates are presented on the website. Order one pair and a bike."

We will perform mathematical calculations and carry out verification using the program code.

To calculate the probabilities , we use the formula $\frac{\beta+n}{\beta V+N}$, $\beta$ - smoothing parameter (let's put it equal to 1), $n$ - the number of hits of a word in a class document, $N$ - the number of words of the class document, $V$ - the size of the training sample.

Let's enter the data in Table 2:

Table 2

| | Words | Getting into the "Spam" class | Getting into the "Not spam" class | The probability of getting into "Spam" | The probability of getting into "Not spam" |
|---|---|---|---|---|---|
| The words of the training sample | laptops | 1 | 0 | | |
| | bargain | 1 | 0 | | |
| | price | 1 | 0 | | |
| | sale | 1 | 0 | | |
| | order | 1 | 1 | (1+1)/(13+9) | (1+1)/(13+6) |
| | bike | 1 | 1 | (1+1)/(13+9) | (1+1)/(13+6) |
| | get | 1 | 0 | | |
| | headphones | 1 | 0 | | |
| | gift | 1 | 0 | | |
| | tomorrow | 0 | 1 | | |
| | will be | 0 | 1 | | |
| | conference | 0 | 1 | | |
| | skates | 0 | 1 | (1+0)/(13+9) | (1+1)/(13+6) |
| | pair | 0 | 0 | (1+0)/(13+9) | (1+0)/(13+6) |
| | website | 0 | 0 | (1+0)/(13+9) | (1+0)/(13+6) |
| | are presented | 0 | 0 | (1+0)/(13+9) | (1+0)/(13+6) |
| | one | 0 | 0 | (1+0)/(13+9) | (1+0)/(13+6) |

We get the following result for the "Spam" class:

$$\frac{2}{4}\cdot\frac{1}{22}\cdot\frac{1}{22}\cdot\frac{1}{22}\cdot\frac{2}{22}\cdot\frac{1}{22}\cdot\frac{1}{22}\cdot\frac{2}{22}=\frac{1}{1247178944}=8,01809560E-10$$

For "Not Spam":

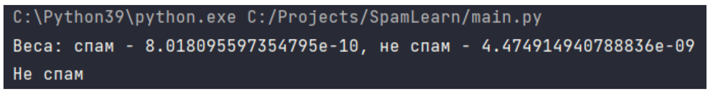
$$\frac{2}{4}\cdot\frac{1}{19}\cdot\frac{1}{19}\cdot\frac{2}{19}\cdot\frac{2}{19}\cdot\frac{1}{19}\cdot\frac{1}{19}\cdot\frac{2}{19}=\frac{16}{3575486956}=4,47491494E-9\,.$$

We are implementing the task in the Python programming language.

As a result of the program, we get weights equal to (see Table 3, Fig. 1):

Table 3

| Spam | Not spam |
|---|---|
| 8.01E-10 | 4.47E-09 |



```
C:\Python39\python.exe C:/Projects/SpamLearn/main.py
Веса: спам - 8.018095597354795e-10, не спам - 4.474914940788836e-09
Не спам
```

Fig. 1. The result of the program

Since the weights of the "Spam" class are less than the weights of the "Not spam" class, we can conclude that the message is not spam, which is confirmed by the program.

Program listing:

```
# the library from which the list of
punctuation marks is taken
from string import punctuation
# the library from which the list of
"stop words" is taken
from stop_words import get_stop_words

spam_line = ['Laptops at a bargain price
',
             'Sale! Order a bike and get
headphones as a gift']
# a list with "non-spam messages"
not_spam_line = ['There will be a
conference tomorrow',
                 'Order skates and a
bike']

# spam verification message
search_spam_line = 'The site presents
skates. Order one pair and a bike'

# a function for formatting a message
(accepts a string, returns a tuple)
def clear_line(line_clearing: str) ->
tuple:
    # the whole string is converted to
lowercase
    line_clearing =
line_clearing.lower()
    # we go through the list of
punctuation marks
    for i in punctuation:
        # replacing the punctuation mark
with "emptiness"
        line_clearing =
line_clearing.replace(i, '')
    # splitting the string into a list
of words
    list_words = line_clearing.split()
    # let's go through the list of "stop
words"
    for i in get_stop_words('ru'):
        # if a word is found in the list
of ready-made words
        if i in list_words:
            # removing a word from the
list
            list_words.remove(i)
    # returning the tuple of the
finished list
    return tuple(list_words)

# spam check function (accepts string
and dictionary, returns boolean value)
def check_spam(line_check: str,
table_info: dict) -> bool:
    # weights "spam" and "not spam"
    result = [1, 1]
    # the number of words from the
training sample,
    # the number of words included in
"spam" and "not spam"
    count = [0, 0, 0]
    ### filling in count ###
    # passage through all the "spam"
elements
    for i in table_info['spam']:
        # adding the number of
"meetings" to the total counter
        count[0] +=
table_info['spam'][i]
        # adding the number of
"interruptions" to the local "spam"
counter
        count[1] +=
table_info['spam'][i]
    # passing through the "not spam"
elements"
    for i in table_info['not_spam']:
        # if the item is not in "spam"
        if i not in table_info['spam']:
            # adding the number of
"meetings" to the total counter
            count[0] +=
table_info['not_spam'][i]
        # adding the number of
"meetings" to the local "not spam"
counter"
        count[2] +=
table_info['not_spam'][i]
    # passing through all the final
words of the test string after
formatting
    for i in clear_line(line_check):
        # if the word is not in "spam"
        if i not in table_info['spam']:
            # adding a word with a
meaning 0
            table_info['spam'][i] = 0
        # if the word is not in "not
spam"
        if i not in
table_info['not_spam']:
            # adding a word with a
meaning 0
            table_info['not_spam'][i] =
0
        # smoothing parameter
        a = 1
        # we change the weights
according to the formula
        result[0] *= (a +
table_info['spam'][i]) / (a * count[0] +
count[1])
        result[1] *= (a +
table_info['not_spam'][i]) / (a *
count[0] + count[2])
    result[0] *=
table_info['count_in_spam'] /
(table_info['count_in_spam'] +
table_info['count_in_not_spam'])
    result[1] *=
table_info['count_in_not_spam'] /
(table_info['count_in_spam'] +
table_info['count_in_not_spam'])
    # debugging information about the
balance status
    print('Weight: spam - %s, not spam -
%s' % (result[0], result[1]))
    # if the weights "spam" are greater
than the weights "not spam"
    if result[0] > result[1]:
        # return true (spam)
        return True
    # otherwise, if "not spam" is more
"spam"
    else:
        # return false (not spam)
        return False

# a function for "learning" (accepts a
list of "spam" and a list of "not spam",
returns a dictionary)
def learn_spam(spam: list, not_spam:
list) -> dict:
    # creating a "blank sheet" of the
dictionary
    dict_words = {'spam': {},
'not_spam': {}, 'count_in_spam': 0,
'count_in_not_spam': 0}
    # buffer lists for words
    spam_words, not_spam_words = [], []
    # going through the "spam" list"
    for i in spam:
        # combining the formatting
result into a single list
        spam_words.extend(clear_line(i))
    # passing through the "spam" buffer
list
    for i in spam_words:
        # adding to the effective
dictionary in the dictionary "spam"
        # the word and as a value - the
number of repetitions
        dict_words['spam'][i] =
spam_words.count(i)
    # going through the "not spam" list
    for i in not_spam:
        # combining the formatting
result into a single list

not_spam_words.extend(clear_line(i))
    # passing through the buffer list
"not spam"
    for i in not_spam_words:
        # adding to the effective
dictionary in the dictionary "not spam"
        # the word and as a value - the
number of repetitions
        dict_words['not_spam'][i] =
not_spam_words.count(i)
    # adding training list lengths to
the result list
    dict_words['count_in_spam'],
dict_words['count_in_not_spam'] =
len(spam), len(not_spam)
    # returning the "trained" dictionary
return dict_words

# we call the spam check function,
passing the checked string to it and
# "trained" by the lists of "spam" and
"not spam" dictionary
if check_spam(search_spam_line,
learn_spam(spam_line, not_spam_line)):
    # if the function returned "true"
    print('Spam')
else:
    # if the function returned "false"
    print('Not spam')
```

## IV. Advantages and disadvantages of the algorithm

The classifier in question has greater performance compared to other simple algorithms on a smaller amount of training data.

The naive Bayesian algorithm characterizes the simplicity and speed of determining the class of the proposed data set.

It is effective in working with categorical features.

When we encounter a variable category in the test set that is not represented in the training set, we will encounter zero frequency, which will require a smoothing technique to solve.

In reality, it is extremely rare to talk about the independence of signs. However, the point is not to consider independent parameters, but that we should not assume any dependence. This allows you to speed up the learning process and predict using any data sets.

## References

[1] V. E. Gmurman, *Teoriya veroyatnostej i matematicheskaya statistika: uchebnoe posobie dlya vuzov*. 11 izd. M.: Vysshaya shkola, 2005. 479 p. (In Russian)

[2] Vysokourovnevyj yazyk programmirovaniya Python [Online]. Available: https://www.python.org/

[3] D. Barber, *Bayesian reasoning and machine learning*. Cambridge University Press, 2012. 642 p.

[4] O.V. Ohlupina, A.A. Prokopenko, A.O. Zgonnikova, *O yomkosti modeli klassifikacii* // Uchyonye zapiski Bryanskogo gosudarstvennogo universiteta. Bryansk: BGU, 2021 (4). pp. 22-27. (In Russian)

**Olga V. Okhlupina,** Candidate Sc. (Phys. and Math.), associate Professor, Bryansk state engineering-technological University,
Prospekt Stanke Dimitrova, 3, Bryansk 241037, Russia.

Dmitry S. Murashko, student, Bryansk state engineering-technological University, Prospekt Stanke Dimitrova, 3, Bryansk 241037, Russia.