# Lab3.Naïve-Bayes

September 19, 2022

#

LAB3 tutorial for Machine Learning Naïve Bayes Classifier Algorithm

Written by JIa Yanhong in 2022. Sept. 16th

## 0.1 Objective

- Understand one of the most popular and simple machine learning classification algorithms, the Naive Bayes algorithm
- Learn how to implement the Naive Bayes Classifier in Python
- Complete the LAB assignment and submit it to BB.

## 0.2 1 The Naïve Bayes algorithm

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- **Naïve**: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- **Bayes**: It is called Bayes because it depends on the principle of Bayes' Theorem.

### 0.2.1 1.1 Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

**Where,**

**P(A|B) is Posterior probability**: Probability of hypothesis A on the observed event B.

**P(B|A) is Likelihood probability**: Probability of the evidence given that the probability of a hypothesis is true.

**P(A) is Prior Probability**: Probability of hypothesis before observing the evidence.

**P(B) is Marginal Probability**: Probability of Evidence.

$$posterior = \frac{likelihood \times prior}{evidence}$$

In practice, there is interest only in the numerator of that fraction, because the denominator does not depend on C and the values of the features $x_i$ are given, so that the denominator is effectively constant. The numerator is equivalent to the joint probability model

### 0.2.2 1.2 Working of Naïve Bayes' Classifier:

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

   **Problem**: If the weather is sunny, then the Player should play or not?

   **Solution**: To solve this, first consider the below dataset:

| Day | Outlook | Play |
|---|---|---|
| **0** | Rainy | Yes |
| **1** | Sunny | Yes |
| **2** | Overcast | Yes |
| **3** | Overcast | Yes |
| **4** | Sunny | No |
| **5** | Rainy | Yes |
| **6** | Sunny | Yes |
| **7** | Overcast | Yes |
| **8** | Rainy | No |
| **9** | Sunny | No |
| **10** | Sunny | Yes |
| **11** | Rainy | No |
| **12** | Overcast | Yes |
| **13** | Overcast | Yes |

**Frequency table for the Weather Conditions:**

| Weather | Yes | No |
|---|---|---|
| Overcast | 5 | 0 |
| Rainy | 2 | 2 |
| Sunny | 3 | 2 |

| Weather | Yes | No |
|---|---|---|
| Total | 10 | 5 |

**Likelihood table weather condition:**

| Weather | No | Yes | |
|---|---|---|---|
| Overcast | 0/4 | 5/10 | 5/14= 0.35 |
| Rainy | 2/4 | 2/10 | 4/14=0.29 |
| Sunny | 2/4 | 3/10 | 5/14=0.35 |
| All | 4/14=0.29 | 10/14=0.71 | |

**Applying Bayes'theorem:**

**P(Yes|Sunny)= P(Sunny|Yes)*P(Yes)/P(Sunny)**

P(Sunny|Yes)= 3/10= 0.3

P(Sunny)= 0.35

P(Yes)=0.71

So P(Yes|Sunny) = 0.3*0.71/0.35= **0.60**

**P(No|Sunny)= P(Sunny|No)*P(No)/P(Sunny)**

P(Sunny|NO)= 2/4=0.5

P(No)= 0.29

P(Sunny)= 0.35

So > P(No|Sunny)= 0.5*0.29/0.35 = **0.41**

So as we can see from the above calculation that **P(Yes|Sunny)>P(No|Sunny)**

**Hence on a Sunny day, Player can play the game.**

### 0.2.3 1.3 naive assumption

To start with, let us consider a dataset.

Consider a fictional dataset that describes the weather conditions for playing a game of golf. Given the weather conditions, each tuple classifies the conditions as fit("Yes") or unfit("No") for plaing golf.

Here is a tabular representation of our dataset.

| Day | OUTLOOK | TEMPERATURE | HUMIDITY | WINDY | PLAY GOLF |
|---|---|---|---|---|---|
| 0 | Rainy | Hot | High | False | No |
| 1 | Rainy | Hot | High | True | No |
| 2 | Overcast | Hot | High | False | Yes |
| 3 | Sunny | Mild | High | False | Yes |

| Day | OUTLOOK | TEMPERATURE | HUMIDITY | WINDY | PLAY GOLF |
|---|---|---|---|---|---|
| 4 | Sunny | Cool | Normal | False | Yes |
| 5 | Sunny | Cool | Normal | True | No |
| 6 | Overcast | Cool | Normal | True | Yes |
| 7 | Rainy | Mild | High | False | No |
| 8 | Rainy | Cool | Normal | False | Yes |
| 9 | Sunny | Mild | Normal | False | Yes |
| 10 | Rainy | Mild | Normal | True | Yes |
| 11 | Overcast | Mild | High | True | Yes |
| 12 | Overcast | Hot | Normal | False | Yes |
| 13 | Sunny | Mild | High | True | No |

The dataset is divided into two parts, namely, **feature matrix** and the **response vector**.

- Feature matrix contains all the vectors(rows) of dataset in which each vector consists of the value of **dependent features**. In above dataset, features are 'Outlook', 'Temperature', 'Humidity' and 'Windy'.
- Response vector contains the value of **class variable**(prediction or output) for each row of feature matrix. In above dataset, the class variable name is 'Play golf'.

Now, its time to put a naive assumption to the Bayes' theorem, which is, **independence** among the features. So now, we split **evidence** into the independent parts.

Now, if any two events A and B are independent, then,

$$P(A, B) = P(A)P(B)$$

Hence, we reach to the result:

$$P(y|x_1, ..., x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

which can be expressed as:

$$P(y|x_1, ..., x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1)P(x_2)...P(x_n)}$$

Now, as the denominator remains constant for a given input, we can remove that term:

$$P(y|x_1, ..., x_n) \propto P(y)\prod_{i=1}^{n} P(x_i|y)$$

Now, we need to create a classifier model. For this, we find the probability of given set of inputs for all possible values of the class variable $y$ and pick up the output with maximum probability. This can be expressed mathematically as:

$$y = argmax_y P(y)\prod_{i=1}^{n} P(x_i|y)$$

So, finally, we are left with the task of calculating P(y) and $P(x_i|y)$.

> Please note that P(y) is also called **class probability** and $P(x_i|y)$ is called **conditional probability**.

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i|y)$.

Let us try to apply the above formula manually on our weather dataset. For this, we need to do some precomputations on our dataset.

We need to find $P(x_i|y_i)$ for each $x_i$ in X and $y_i$ in y. All these calculations have been demonstrated in the tables below:

So, in the figure above, we have calculated $P(x_i|y_i)$ for each xi in X and $y_i$ in y manually in the tables 1-4. For example, probability of playing golf given that the temperature is cool, i.e `P(temp. = cool | play golf = Yes) = 3/9`.

Also, we need to find class probabilities (P(y)) which has been calculated in the table 5. For example, P(play golf = Yes) = 9/14.

So now, we are done with our pre-computations and the classifier is ready!

Let us test it on a new set of features (let us call it today):

$$today = (Sunny, Hot, Normal, False)$$

So, probability of playing golf is given by:

$$P(Yes|today) = \frac{P(Sunny\ Outlook|Yes)P(Hot\ Temperature|Yes)P(Normal\ Humidity|Yes)P(No\ Wind|Yes)P}{P(today)}$$

and probability to not play golf is given by:

$$P(No|today) = \frac{P(Sunny\ Outlook|No)P(Hot\ Temperature|No)P(Normal\ Humidity|No)P(No\ Wind|No)P(N}{P(today)}$$

Since, P(today) is common in both probabilities, we can ignore P(today) and find proportional probabilities as:

$$P(Yes|today) \propto \frac{2}{9}.\frac{2}{9}.\frac{6}{9}.\frac{6}{9}.\frac{9}{14} \approx 0.0141$$

and

$$P(No|today) \propto \frac{3}{5}.\frac{2}{5}.\frac{1}{5}.\frac{2}{5}.\frac{5}{14} \approx 0.0068$$

Now, since

$$P(Yes|today) + P(No|today) = 1$$

These numbers can be converted into a probability by making the sum equal to 1 (normalization):

$$P(Yes|today) = \frac{0.0141}{0.0141 + 0.0068} = 0.67$$

and

$$P(No|today) = \frac{0.0068}{0.0141 + 0.0068} = 0.33$$

Since

$$P(Yes|today) > P(No|today)$$

So, prediction that golf would be played is 'Yes'.

The method that we discussed above is applicable for discrete data. In case of continuous data, we need to make some assumptions regarding the distribution of values of each feature. The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of P(xi | y).

Now, we discuss one of such classifiers here.

### 0.2.4 1.4 Gaussian Naive Bayes classifier

In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a **Gaussian distribution**. A Gaussian distribution is also called Normal distribution. When plotted, it gives a bell shaped curve which is symmetric about the mean of the feature values as shown below:

The likelihood of the features is assumed to be Gaussian, hence, conditional probability is given by:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Example: Continuous-valued Features

- Temperature is naturally of continuous value > Yes: 25.2, 19.3, 18.5, 21.7, 20.1, 24.3, 22.8, 23.1, 19.8 > > No: 27.3, 30.1, 17.4, 29.5, 15.1
- Estimate mean and variance for each class

$$\mu = \frac{1}{N}\sum_{1}^{N} x_n \ , \ \sigma^2 = \frac{1}{N}\sum_{1}^{N}(x_n - \mu)^2$$

  $ *{Yes}=21.64* , {Yes}=2.35$

  $ *{No}=23.88* , {No}=7.09 $

- Learning Phase: output two Gaussian models > $\hat{P}(x|Yes) = \frac{1}{2.35\sqrt{2\pi}} exp\left(-\frac{(x-21.64)^2}{2\times 2.35^2}\right)$ > >$\hat{P}(x|No) = \frac{1}{7.09\sqrt{2\pi}} exp\left(-\frac{(x-23.88)^2}{2\times 7.09^2}\right)$

### 0.2.5 1.5 Zero Conditional Probability

Sometimes, we face a zero conditional probability problem during test.

$$\hat{P}(x_1|c_i)\cdots\hat{P}(a_{jk}|c_i)\cdots\hat{P}(x_n|c_i) = 0 \; for \; x_i = a_{jk}, \; \hat{P}(a_{jk}|c_i) = 0$$

Let's test another example:

$$today = (Overcast, Hot, Normal, False)$$

and probability to not play golf is given by:

$$\begin{aligned} P(No|today) &\propto P(Overcast\ Outlook|No)P(Hot\ Temperature|No)P(Normal\ Humidity|No)P(No\ Wind|No)P( \\ &\propto \frac{0}{5}.\frac{2}{5}.\frac{1}{5}.\frac{2}{5}.\frac{5}{14} = 0 \end{aligned}$$

In this case, the model becomes ineffective!!!

For a remedy, class conditional probabilities reestimated with:

$$\hat{P}(a_{jk}|c_i) = \frac{n_c + mp}{n + m}$$

$n_c$: number of training examples for which $x_j = a_{jk}$ and $c = c_i$

n : number of training examples for which $c = c_i$

p : prior estimate (usually,$p = \frac{1}{t}$ for t possible values of $x_{j} $)

m : weight to prior (number of " virtual" examples, $m \geqslant 1$)+ Temperature is naturally of continuous value

- Example: P(outlook=overcast|no)=0 in the play-golf dataset
  - Adding m `virtual` examples
    - In this dataset, training examples for the `no` class is 5.
    - We can only add m=1 `virtual` example in our m-esitmate remedy.
  - The `outlook` feature can takes only 3 values. So p=1/3.
  - Re-estimate P(outlook|no) with the m-estimate

$p(overcast|no) = \frac{0+1*\frac{1}{3}}{5+1} = \frac{1}{18}$

$p(sunny|no) = \frac{3+1*\frac{1}{3}}{5+1} = \frac{5}{9}$

$p(rain|no) = \frac{2+1*\frac{1}{3}}{5+1} = \frac{7}{18}$

## 0.3 2 LAB Assignment

Text mining (deriving information from text) is a wide field which has gained popularity with the huge text data being generated. Automation of a number of applications like sentiment analysis, document classification, topic classification, text summarization, machine translation, etc., has been done using machine learning models.

### 0.3.1 Exercise1 Spam filtering

In this Exercise, you are required to write your spam filter by using naïve Bayes method. This time you should not use 3rd party libraries including scikit-learn.

Spam filtering is a beginner's example of the document classification task which involves classifying an email as spam or non-spam (a.k.a. ham) mail. An email dataset will be provided. We will use the following steps to build this application: 1) Preparing the text data 2) Creating a word dictionary 3) Feature extraction 4) Implementation of the Naive Bayes algorithm 5) Training the classifier 6) Checking the results on the test set ### 1) Preparing the text data The data-set used here, is split into a training set and a test set containing 702 mails and 260 mails respectively, divided equally between spam and ham mails. You will easily recognize spam mails as it contains `spmsg` in its filename.

In any text mining problem, text cleaning is the first step where we remove those words from the document which may not contribute to the information we want to extract. Emails may contain a lot of undesirable characters like punctuation marks, stop words, digits, etc which may not be helpful in detecting the spam email. The emails in Ling-spam corpus have been already preprocessed in the following ways:

1. **Removal of stop words** – Stop words like "and", "the", "of", etc are very common in all English sentences and are not very meaningful in deciding spam or legitimate status, so these words have been removed from the emails.
2. **Lemmatization** – It is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item. For example, "include", "includes," and "included" would all be represented as "include". The context of the sentence is also preserved in lemmatization as opposed to stemming (another buzz word in text mining which does not consider meaning of the sentence)

We still need to remove the non-words like punctuation `marks` or `special characters` from the mail documents. There are several ways to do it. Here, we will remove such words after creating a dictionary, which is a very convenient method to do so since when you have a dictionary; you need to remove every such word only once. ### 2) Creating word dictionary We will only perform text analytics on the content to detect the spam mails. As the first step, we need to create a dictionary of words and their frequency. For this task, a training set of 700 mails is utilized. This python function will create the dictionary for you.

```
from collections import Counter
def make_Dictionary(train_dir):
    emails = [os.path.join(train_dir,f) for f in os.listdir (train_dir)]
    all_words = []
    for mail in emails:
        with open (mail) as m:
            for i,line in enumerate (m) :
                if i == 2:
                    words = line.split()
                    all_words += words
    dictionary = Counter(all_words)
    # Write code for non-word removal here
```

```
    return dictionary
```

Once the dictionary is created we can add just a few lines of code written below to the above function to remove non-words. Absurd single characters in the dictionary which are irrelevant here are also removed. Do not forget to insert the below code in the function of make_Dictionary:

```
list_to_remove = list(dictionary.keys())
for item in list_to_remove:
    if item.isalpha() == False:
        del dictionary[item]
    elif len(item) == 1:
        del dictionary[item]
dictionary = dictionary.most_common(3000)
```

The dictionary can be seen by the command "print dictionary". You may find some absurd word counts to be high but don't worry, it's just a dictionary and you always have a chance to improve it later. If you use the provided dataset, make sure your dictionary has some of the entries given below as most frequent words. Here 3000 most frequently used words are chosen in the dictionary.

[ ]:
```
# To show the most frequent words in train-mails
dictionary = make_Dictionary('ling-spam/train-mails')
dictionary
```

### 0.3.2 3) Feature Extraction Process

Once the dictionary is ready, we can extract word count vector (our feature here) of 3000 dimensions for each email of the training set. Each **word count vector** contains the frequency of 3000 words in the training file. Of course you might have guessed by now that most of them will be zero. Let us take an example. Suppose we have 500 words in our dictionary. Each word count vector contains the frequency of 500 dictionary words in the training file. Suppose the text in the training file is "Get the work done, work done", then it will be encoded as

$$[0, 0, 0, 0, 0, .....0, 0, 2, 0, 0, 0, ....., 0, 0, 1, 0, 0, ...0, 0, 1, 0, 0, .....2, 0, 0, 0, 0, 0]$$

Here, all the word counts are placed at the 296th, 359th, 415th, 495th elements of the word count vector in the length of 500 and the rest are zero.

The below python code will generate a feature vector matrix whose rows denote 700 files of the training set and columns denote 3000 words of the dictionary. The value at index $ij$ will be the number of occurrences of the $j^{th}$ word of the dictionary in the $i^{th}$ file

[ ]:
```
import os
import numpy as np
def extract_features(mail_dir):
    files = [os.path.join(mail_dir,fi) for fi in os.listdir(mail_dir)]
    features_matrix = np.zeros((len(files),3000))
    docID = 0
    _i = 0
    print(len(files))
    for fil in files:
        _i+=1
```

```
        with open(fil) as fi:
            for i,line in enumerate(fi):
                if i == 2:
                    words = line.split()
                    for word in words:
                        wordID = 0
                        for i,d in enumerate(dictionary):
                            if d[0] == word:
                                wordID = i
                                features_matrix[docID,wordID]+=1
            docID = docID + 1
        print('\r','done {} files'.format(_i),flush=True,end='')
    return features_matrix
```

[ ]:
```
# Create a dictionary of words with its frequency
features_matrix = extract_features("ling-spam/train-mails")
features_matrix
```

### 0.3.3 4) Training the Classifiers

Here you should write your Naïve Bayes classifiers after fully understanding its principle.

[ ]:
```
########### Write Your Code Here ###########
# Prepare feature vectors per training mail and its labels

X_train = [[]]

y_train = []

##############################################
```

[ ]:
```
########### Write Your Code Here ###########
# Prepare feature vectors per testing mail and its labels
X_test = [[]]
y_test = []



##############################################
```

### 0.3.4 5) Implementation of the Naive Bayes algorithm

Complete the code for naive Bayes algorithm in `predict` function.

[ ]:
```
########### Write Your Code Here ###########
class NaiveBayes():

```

```
    def fit(self, X, y):
        y = y.astype(int)
        self.X = X
        self.y = y
        self.classes = np.unique(y)
        self.parameters = {}
        for i, c in enumerate(self.classes):
            # Calculate the mean, variance, prior probability of each class
            X_Index_c = X[np.where(y == c)]
            X_index_c_mean = np.mean(X_Index_c, axis=0, keepdims=True)
            X_index_c_var = np.var(X_Index_c, axis=0, keepdims=True)
            parameters = {"mean": X_index_c_mean, "var": X_index_c_var, "prior":
 ↪ X_Index_c.shape[0] / X.shape[0]}
            self.parameters["class" + str(c)] = parameters



    def predict(self, X):
        # return class with highest probability
        prediction = 0

        # Complete code for naive Bayes algorithm




        return prediction



##############################################
```

[ ]:
```
from sklearn.metrics import confusion_matrix

#Call the Naive Bayes algorithm, which we wrote ourselves
model = NaiveBayes()
model.fit(X_train,y_train)
result = model.predict(X_test)
print (confusion_matrix(y_test, result))
```

### 0.3.5 5) Checking the results on test set

The test set contains 130 spam emails and 130 non-spam emails. Please compute accuracy, recall, F-1 score to evaluate the performance of your spam filter.

[ ]:
```
############ Write Your Code Here ############
from sklearn.metrics import accuracy_score, recall_score, f1_score
```

```
##############################################
```

### 0.3.6 Exercise 2 Compare your Naïve Bayes algorithm with GassianNB from Sklearn, which one does better? Where is the gap?

[ ]:
```
########### Write Your Code Here ###########
from sklearn.naive_bayes import GaussianNB

##############################################
```

### 0.3.7 Exercise 3 Questions

1. Describe another real-world application where the naïve Bayes method can be applied
2. What are the strengths of the naïve Bayes method; when does it perform well?
3. What are the weaknesses of the naïve Bayes method; when does it perform poorly?
4. What makes the naïve Bayes method a good candidate for the classification problem, if you have enough knowledge about the data?