VIET NAM NATIONAL UNIVERSITY, HO CHI MINH CITY
HO CHI MINH UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DISCRETE STRUCTURES FOR COMPUTER SCIENCE

Assignment

Naive Bayes Classifier

Instructor:	Lê Hồng Trang
Student:	Trần Hoàng Long - 1852545
	Phạm Hoàng Hải - 1852020
	Phạm Hoàng Tùng - 1852852
	Trần Nguyễn - 1852740

Ho Chi Minh City, Nov 2018

Mục lục

1 Introduction

Classification is a form of data analysis that extracts models describing important data classes. Such models, called classifiers, predict categorical (discrete, unordered) class labels.

One of the major problems encountered by designers of object-oriented software is classification; that is, finding which classes should be grouped together under a shared base class.

When attempting to perform classification on the problem space, several issues must be addressed. For example, the designer must decide which properties should be used to determine commonality. The classification should also be flexible enough to permit the introduction of new objects into the system which appear to belong to neither class nor appear to have properties of several classes.

This is a problem faced by other members of the scientific community as well. For example, biologists have traditionally divided living beings into two classes: plants and animals; every living entity must belong to one, and only one, of these classes.

In this report, we solve the car buying problem using naive Bayes classifier and provide a practical implementation.

2 Theoretical basis

2.1 Classifier

Classification involves supervised learning of a function $f(x)$ whose value is nominal, that is, drawn from a finite set of possible values. The learned function is called a classifier. It is given instances x of one or another class, and it must determine which class each instance belongs to; the value $f(x)$ is the classifier's prediction regarding the class of the instance. For example, an instance might be a particular word in context, and the classification task is to determine its part of speech. The learner is given labeled data consisting of a collection of instances along with the correct answer, that is, the correct class label, for each instance.

The classification algorithm builds the classifier by analyzing or "learning from" a training set made up of database tuples and their associated class labels.

2.2 Bayes' theorem

In probability theory and statistics, Bayes' theorem (alternatively Bayes' law or Bayes' rule) describes the probability of an event, based on prior knowledge of conditions that might be related to the event. For example, if cancer is related to age, then, using Bayes' theorem, a person's age can be used to more accurately assess the probability that they have cancer, compared to the assessment of the probability of cancer made without knowledge of the person's age.

Bayes' theorem is stated mathematically as the following equation:

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

where:

A is an event

B is an event such that $P(B) \neq 0$

$P(A|B)$ is a conditional probability: the likelihood of event A occurring given that B is true

$P(B|A)$ is also a conditional probability: the likelihood of event B occurring given that A is true

$P(A)$ and $P(B)$ are the probabilities of observing A and B independently of each other; this is known as the marginal probability

2.3 Naive Bayes classifier

Bayesian classifiers are statistical classifiers. They can predict class membership probabilities such as the probability that a given tuple belongs to a particular class.

For the purpose of the naive Bayes algorithm, an instance is viewed as a n-tuple of attribute values: $\mathbf{a} = (a_1, a_2, \ldots, a_n)$, which each attribute has a set of values $A_1, A_2, \ldots, A_n$. Then the algorithm assign to this instance $P(v|\mathbf{a})$ for each label v from V.

Using Bayes' theorem, the conditional probability above can be decomposed as

$$P(v|\mathbf{a}) = \frac{P(v)P(\mathbf{a}|v)}{P(a)}$$

Since the denominator is constant across choices of label, we can ignore it. To reduce computation in evaluating $P(\mathbf{a}|v)$, the naive assumption of class-conditional independence is made. This presumes that the attributes' values are conditionally independent of one another, given the class label of the tuple.

Further explaination: The Naive Bayes classifier infers that the probability of a class label given data using a simplifying assumption that the attributes are independent given the label. In other words, attributes cannot affect each others, this property seems to be impossible in real life because true or false information is affect by many unpredicted events and it is hard to collect such a huge data and its attributes.

Thus, given that these attributes are truly independent, we have:

$$P(\mathbf{a}|v) = P(x_1|V_i)P(x_2|V_i)\ldots P(x_n|V_i) = \prod_i P(x_i|v) \tag{1}$$

We can rewrite equation as follow

$$P(v|\mathbf{a}) \propto P(v)\prod_i P(x_i|v)$$

The algorithm estimates $P(v)$ and $P(a_i|v)$ by relative frequency in the training data.

$$P(v) = \frac{N(v)}{N}$$

$$P(a_i|v) = \frac{N(v, a_i)}{N(v)}$$

where:

N is the number of training sample

$N(v)$ is the number of training samples for which the label is v

$N(v, a_i)$ is the number of samples for which the label is v and the attribute is a_i

But the above formula for $P(a_i|v)$ provides a poor estimate if $N(v, a_i)$ is very small. In extreme case, if $N(v, a_i) = 0$, then the whole posterior will be zero. The solution is using the m-estimate of probabilities:

$$P(a_i|v) = \frac{N(v, a_i) + mp(a_i)}{N(v) + m} \tag{2}$$

where:

$p(a_i)$ is prior estimate of the probability

m is equivalent sample size

In the absence of other information, we can assume a uniform prior:

$$p(a_i) = \frac{1}{|A_i|} \tag{3}$$

The discussion so far has derived the independent feature model, that is, the naive Bayes probability model. The naive Bayes classifier combines this model with a decision rule. One common rule is to pick the hypothesis that is most probable; this is known as the maximum a posteriori or MAP decision rule. Hence the prediction of the classifier is a subset V^* of V: [1]

$$V^* = \underset{v \in V}{\operatorname{argmax}}\, P(v) \prod_i P(x_i|v) \tag{4}$$

In practice, the function will only maximize at a single value v^*, that is, V^* is a singleton: $V^* = \{v^*\}$. We can safely say that the predicted label is v^*.

3 Problem

A person (male or female) usually choose to buy a car with different attributes of the car such as color, type, and origin. Following table gives examples of such car buying.

No.	Color	Type	Origin	Buyer
1	red	sport	domestic	male
2	red	sport	domestic	female
3	red	sport	domestic	male
4	yellow	sport	domestic	female
5	yellow	sport	imported	male
6	yellow	SUV	imported	female
7	yellow	SUV	imported	male
8	yellow	SUV	domestic	female
9	red	SUV	imported	female
10	red	sport	imported	male

So, the question is, given a car with any random property, let's say (red, SUV, imported) (as required by the assignment), should the car be chosen by a male or female?

[1] Given a function $f : X \to Y$, the arg max over some subset S of X is defined by

$$\underset{x \in S}{\operatorname{argmax}}\, f(x) = \{x \in S \mid \forall y \in S : f(y) \le f(x)\}$$

3.1 Recognize the problem

Our training data is a set of 10 4-tuples, where each 4-tuple presents 4 attributes to an object. Each element of the 4-tuple can have one of the following values:

- *red* or *yellow* for the *color* attribute
- *SUV* or *sport* for the *type* attribute
- *domestic* or *imported* for the *origin* attribute
- *female* or *male* for the *buyer* label

In this problem, the target attribute, the attribute we want to predict, is *buyer*. The target attribute can be considered as labels for our data.
So each sample in the training data have 3 attributes and a label

- *color* is the first attribute
- *type* is the second attribute
- *origin* is the third attribute
- *buyer* is the label

and each attribute and the label has the following set of values

$$\begin{aligned} A_1 &= \{\text{red}, \text{yellow}\} \\ A_2 &= \{\text{sport}, \text{SUV}\} \\ A_3 &= \{\text{domestic}, \text{imported}\} \\ V &= \{\text{female}, \text{male}\} \end{aligned}$$

The object is the target which is classified by the classifier, specifically in our case: A car with any random properties such as **(Yellow, Sport, Domestic)**
To determine the label of the given tuple $\mathbf{a} = (\text{red}, \text{SUV}, \text{imported})$, we need to calculate the probabilities:

$$P(v = \text{male}) \prod_i P(a_i \mid v = \text{male})$$

$$P(v = \text{female}) \prod_i P(a_i \mid v = \text{female})$$

where $a_1 = \text{red}$, $a_2 = \text{SUV}$, $a_3 = \text{imported}$
From the given formula (4) by the naive Bayes classification, we can see that our needed variables are:

$P(v = \text{female})$	$P(v = \text{male})$
$P(a_1 = \text{red} \mid v = \text{female})$	$P(a_1 = \text{red} \mid v = \text{male})$
$P(a_2 = \text{SUV} \mid v = \text{female})$	$P(a_2 = \text{SUV} \mid v = \text{male})$
$P(a_3 = \text{imported} \mid v = \text{female})$	$P(a_3 = \text{imported} \mid v = \text{male})$

To calculate these varibles using formula (2), we also have to find following varible

$N(v = \text{female})$	$N(v = \text{male})$
$N(a_1 = \text{red} \mid v = \text{female})$	$N(a_1 = \text{red} \mid v = \text{male})$
$N(a_2 = \text{SUV} \mid v = \text{female})$	$N(a_2 = \text{SUV} \mid v = \text{male})$
$N(a_3 = \text{imported} \mid v = \text{female})$	$N(a_3 = \text{imported} \mid v = \text{male})$

After calculating above varibles, we can compare $P(v = \text{male}) \prod_i P(a_i \mid v = \text{male})$ and $P(v = \text{female}) \prod_i P(a_i \mid v = \text{female})$. The label correspond to greater value will be the label of the object.

3.2 Calculation

All the values needed for the formula (2) are computed in the following table:

	$a_1 = \text{red}$	$a_2 = \text{SUV}$	$a_3 = \text{imported}$
$v = \text{female}$			
$N(v)$	5	5	5
$N(v, a_i)$	2	3	2
$p(a_i)$	0.5	0.5	0.5
m	3	3	3
$v = \text{male}$			
$N(v)$	5	5	5
$N(v, a_i)$	3	1	3
$p(a_i)$	0.5	0.5	0.5
m	3	3	3

Now we simply apply equation (2) using the values above

$$\begin{aligned}
P(a_1 = \text{red} \mid v = \text{female}) &= \frac{2 + 3 \cdot 0.5}{5 + 3} = \frac{7}{16} \\
P(a_2 = \text{SUV} \mid v = \text{female}) &= \frac{3 + 3 \cdot 0.5}{5 + 3} = \frac{9}{16} \\
P(a_3 = \text{imported} \mid v = \text{female}) &= \frac{2 + 3 \cdot 0.5}{5 + 3} = \frac{7}{16} \\
P(a_1 = \text{red} \mid v = \text{male}) &= \frac{3 + 3 \cdot 0.5}{5 + 3} = \frac{9}{16} \\
P(a_2 = \text{SUV} \mid v = \text{male}) &= \frac{1 + 3 \cdot 0.5}{5 + 3} = \frac{5}{16} \\
P(a_3 = \text{imported} \mid v = \text{male}) &= \frac{3 + 3 \cdot 0.5}{5 + 3} = \frac{9}{16}
\end{aligned}$$

Since $N(v = \text{female}) = N(v = \text{male}) = 5$ and $N = 10$, we have

$$P(v = \text{female}) = P(v = \text{male}) = \frac{5}{10} = \frac{1}{2}$$

To classify, we caculate:

$$\begin{aligned}
& P(v = \text{female})P(a_1 = \text{red}|v = \text{female})P(a_2 = \text{SUV} \mid v = \text{female})P(a_3 = \text{imported} \mid v = \text{female}) \\
= \quad & \frac{1}{2} \cdot \frac{7}{16} \cdot \frac{9}{16} \cdot \frac{7}{16} \\
= \quad & \frac{441}{8192} \\
\approx \quad & 0.0538
\end{aligned}$$

$$\begin{aligned}
& P(v = \text{male})P(a_1 = \text{red} \mid v = \text{male})P(a_2 = \text{SUV} \mid v = \text{male})P(a_3 = \text{imported} \mid v = \text{male}) \\
= \quad & \frac{1}{2} \cdot \frac{9}{16} \cdot \frac{5}{16} \cdot \frac{9}{16} \\
= \quad & \frac{405}{8192} \\
\approx \quad & 0.0494
\end{aligned}$$

Since $0.0538 > 0.0494$, our example's label is *female*. In another way, the car whose features are (red, SUV, imported) is more likely to be bought by a female customer.

4 Program

4.1 Algorithm

4.1.1 Preprocessing

In this section, we will discuss about our algorithm and apply it to the assignment problem.

We will use matrices and vectors as data structures to transform all of our calculations to matrix operations. To achieve this, all data will be binarized in preprocessing phase. Simply, we will use zero-one vectors to present attributes and labels. A component of this vector can takes the value 0 or 1 to indicate the absence or presence of the corresponding feature in the instance.

To binarize attributes, a function f which maps a n-tuple to a zero-one vector whose dimension equals number of features $n_f = \sum_i |A_i|$ will be applied to each attribute.

$$f : A_1 \times A_2 \times \ldots \times A_n \to \{0, 1\}^{n_f}$$

$$f\big((\text{red}, \text{sport}, \text{domestic})\big) = \begin{array}{cccccc} \text{red} & \text{yellow} & \text{sport} & \text{SUV} & \text{domestic} & \text{imported} \\ \big(1 & 0 & 1 & 0 & 1 & 0 \big) \end{array}$$

To binarize labels, a function g which map a label to a zero-one vector will be applied to labels.

$$g : V \to \{0, 1\}^{|V|}$$

$$g(\{\text{male}\}) = \begin{array}{cc} \text{female} & \text{male} \\ \big(0 & 1 \big) \end{array}$$

Applying f and g along the rows of the training data and test data and stack the result vertically, we will get the training data matrix $\mathbf{T}$, test data matrix $\mathbf{T}'$ and label matrix $\mathbf{L}$. For our problem:

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}, \quad \mathbf{L} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \mathbf{T} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

4.1.2 Training

Here come the interesting part. To know the number of examples for which the label is v and the attribute is a_i, multiply the transpose of label matrix **L** with training matrix **T** and the result will be the matrix **C** containing all the numbers we want.

$$\mathbf{C} = \mathbf{L}^\top \mathbf{T} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 3 & 3 & 2 & 3 & 2 \\ 3 & 2 & 1 & 4 & 2 & 3 \end{pmatrix}$$

Why is this working? According to the definition of matrix multiplication

$$C_{ij} = \sum_{k=1}^{N} L_{ik}^\top T_{kj} = \sum_{k=1}^{N} L_{ki} T_{kj}$$

That is, each element in **C** equals the sum of element-wise product of corresponding column vector from **L** and **T**. Each element in the column vector correspond to a sample in training data. Take the first element in **C** as an example:

$$\begin{aligned} C_{11} &= 0 \cdot 1 + 1 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 + 0 \cdot 0 + 1 \cdot 0 + 1 \cdot 0 + 0 \cdot 0 + 1 \cdot 0 + 1 \cdot 1 + 0 \cdot 1 \\ &= 2 \end{aligned}$$

As we can see, just the product of the element correspond to a sample with color red and label *female* will have value 1, while other product will have value 0. The sum will equal the number of term 1, which is the number of sample with color *red* and label *female*.

From original dataset, we can calculate the vector **p** containing the priori estimate for each feature and **c** containing the numbers of samples for each label

$$\mathbf{c} = \begin{array}{c} \text{female} \quad \text{male} \\ \begin{pmatrix} \;5 \qquad\; 5\; \end{pmatrix} \end{array}$$

$$\begin{array}{cccccccc} & & \text{red} & \text{yellow} & \text{sport} & \text{SUV} & \text{domestic} & \text{imported} \\ \mathbf{p} = & (& 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 &) \end{array}$$

Then matrix $\mathbf{P}$ containing all the values of $P(a_i|v)$ can be calculated as follow

$$P_{ij} = \frac{C_{ij} + mp_j}{c_i + m} \tag{5}$$

$$\mathbf{P} = \begin{pmatrix} 0.4375 & 0.5625 & 0.5625 & 0.4375 & 0.5625 & 0.4375 \\ 0.5625 & 0.4375 & 0.3125 & 0.6875 & 0.4375 & 0.5625 \end{pmatrix}$$

4.1.3 Classifying

Instead of multiplying probalities like in equation (4), we will use natural logarithm to transform multiplication into addition and prevent underflow in our program when multiplying tiny probalities in giant datasets. Because natural logarithm is a monotonically increasing function:

$$\begin{aligned} V^* = \operatorname*{argmax}_{v \in V} P(v) \prod_i P(x_i|v) &= \operatorname*{argmax}_{v \in V} \left(\ln \left(P(v) \prod_i P(x_i|v) \right) \right) \\ &= \operatorname*{argmax}_{v \in V} \left(\ln \frac{N(v)}{N} + \sum_i \ln P(x_i|v) \right) \\ &= \operatorname*{argmax}_{v \in V} \left(\ln N(v) - \ln N + \sum_i \ln P(x_i|v) \right) \end{aligned}$$

As the number of training samples N doesn't depend on the label, the term $\ln N$ can be removed from our equation

$$V^* = \operatorname*{argmax}_{v \in V} \left(\ln N(v) + \sum_i \ln P(x_i|v) \right)$$

To calculate all the value of the expression in parenthesis, add the entrywise natural logarithm of $\mathbf{c}$ to each row of the product of $\mathbf{T}'$ and the transpose of $\mathbf{P}$.

$$P'_{i\cdot} = \ln \mathbf{c} + (\mathbf{T}' \ln \mathbf{P}^\top)_{i\cdot}$$

$$P'_{ij} = \ln c_j + (\mathbf{T}' \ln \mathbf{P}^\top)_{ij} = \ln c_j + \sum_{k=1}^{n_f} T'_{ik} \ln P^\top_{kj} = \ln c_j + \sum_{k=1}^{n_f} T'_{ik} \ln P_{jk}$$

$$\mathbf{P}' = \begin{pmatrix} -0.61928338 & -0.70444119 \end{pmatrix}$$

To find the label that mamixize the probality, we apply a function h which is similar to argmax along the rows of $\mathbf{P}'$ that return the binary vector correspond to the predicted label.

$$\mathbf{L}' = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

Applying function g^{-1} to above vector, we can find the label

$$g^{-1}\left(\begin{pmatrix} 1 & 0 \end{pmatrix}\right) = \{\text{female}\}$$

Our algorithm return the same prediction as previous section.

4.2 Implementation

Our program is written in Python using NumPy library. Python is a general purpose programming language created by Guido van Rossum and first released in 1991. Thanks to its gentle learning curve and large repository of third-party software, Python is very popular in academia and industry. NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

First, we import library that we will use in our program and set equivalent sample size as given by the assignment. Note that we bind the `numpy` to a local name `np`, so in our code when we want to use a function called `numpy.foo`, we just have to write `np.foo`.

```
import numpy as np
import sys

# equivalent sample size
m = 3
```

Then we define function f, g, h and g^{-1}, which we will call `binarize_attr`, `binarize_label`, `argmax` and `find_label`, respectively. The function `binarize_attr` scan through a row in the dataset loop up each feature in `features` list.

```
def binarize_attr(row):
    attrs_b = []
    for i in range(n_attrs):
        for zeroone in np.isin(features[i], row[i]):
            attrs_b.append(zeroone)
    return attrs_b

def binarize_label(label):
    return np.isin(labels, label).astype(np.uint8)

def argmax(row):
    return row == np.amax(row)

def find_label(vector):
    return labels[np.where(vector)]
```

To load data from CSV file, use `numpy.loadtxt`. Because NumPy consider a vector as a 1D array and a matrix as a 2D array, we have to use `numpy.atleast_2d` to guarantee that our test set is always a matrix in case there is just one test sample. The dataset will be splited into training set and labels using `numpy.split`.

```
dataset = np.loadtxt(sys.argv[1], delimiter=",", dtype=np.object_)
testset = np.atleast_2d(np.loadtxt(sys.argv[2], delimiter=",", dtype=np.object_))
trainset, label = np.split(dataset, [-1], axis=1)
```

Number of attributes can be easily determined from the `trainset`'s width using its `shape` attribute. Using `numpy.unique`, we can find labels and the number of sample with each labels in our dataset.

```
n_attrs = trainset.shape[1]
labels, label_count = np.unique(label, return_counts=True)
```

Finding features is harder. First we define the `features` which store all features as a empty list. To make sure that `features` include new features in test set that don't appear in the training set, we have to scan through both `trainset` and `testset` using `numpy.concatenate` and `numpy.transpose` to loop through each columns instead of rows. In the loop, list of unique values in the column are appended to `features`. To find the number of features, simply find the length of each list in `features` and add them together.

```
features = []
for col in np.transpose(np.concatenate((trainset, testset))):
    features.append(np.unique(col))
```

To find priori estimate for each feature's probality, we also define the `priori` which store all priori estimate as a empty list. Then we loop through each list in `features` to calculate a list of priori estimate whose length equal the number of attribute values using formula (3). Finally, we use `numpy.ravel` to flatten our list.

```
priori = []
for attr in features:
    for _ in range(len(attr)):
        priori.append(1/len(attr))
priori = np.array(priori)
```

Then we apply `binarize_attr` and `binarize_label` along the rows of `trainset`, `testset` and `label` to binarize our data.

```
trainset_b = np.apply_along_axis(binarize_attr, 1, trainset)
testset_b = np.apply_along_axis(binarize_attr, 1, testset)
label_b = np.apply_along_axis(binarize_label, 1, label)
```

We multiply the transpose of 2D array `label_b` with `trainset_b` using `@` operator.

```
count = np.dot(label_b.T, trainset_b)
```

Now we will calculate the values of $P(a_i|v)$ using formula (5). But that formula can't be interpreted as any matrix operation, does that mean we have to use loops? Not necessarily, because NumPy has a powerful feature called *broadcating*. *Broadcasting* provides a mean of vectorizing array operations so that looping occur in C instead Python, which makes our code cleaner and faster. When doing arithmetic operation with a scalar and a 1D array, we can think of the scalar being stretched during the arithmetic operation into an array with the same shape as the 1D array and the operation is done element-wisely. It is the same with 1D array and 2D array, but the 1D array always be stretched vertically by default. The division part is trickier. Because we want to divide element-wisely each row of numerator by denominator, we use NumPy advanced indexing to create a column vector from 1D array `label_count`, so NumPy will stretch it horizontally instead of vertically to match the size of numerator, and then we can divide numerator by denominator element by element.

```
prob = (count + m * priori) / (label_count[:,None] + m)
```

To calculate matrix $\mathbf{P}'$, we also use broadcasting to add 1D array `np.log(label_count)` to 2D array `testset_b @ np.log(prob.T)`. Then we apply function `argmax` along the rows of `func` to find binary representation of predicted labels.

```
func = np.log(label_count) + testset_b @ np.log(prob.T)
print(func)
```

Funcion `find_label` is then applied along the rows of `predict` to find the labels and print it to the screen.

```
predict = np.apply_along_axis(find_label, 1, predict_b)
```

Test this program on Ubuntu 18.04 with Python 3.6.7 and NumPy 1.13.3, we have following result:

```
$ python3 bayes.py data.csv test.csv
[['female']]
```

5 Conclusion

In this assignment, we have delved into building and understanding a Naive Bayesian Classifier. As we go through the content,it could be seen that this algorithm is well suited for data that can be asserted to be independent. Being a probablistic model, it works well for classifying data into multiple directions given the underlying score.

This supervised learning method is useful for fraud detection, spam filtering, and any other problem that has these types of features.

Even though, the condition for the Bayes classification method is conditional independence assumption, which makes it an inaccurate data point-class label association probability estimator in many cases. Because, as you can see in real life situations, the attributes of a tuple, such as a person is mostly dependent of each others like status, age, occupation and income!

Tài liệu

[1] Jiawei Han, Micheline Kamber, Jian Pei., *Data Mining: Concepts and Techniques*, 3rd ed. Waltham, Massachusetts: Morgan Kaufmann, 2012

[2] Steven Abney, *Semisupervised Learning for Computational Linguistics*, London: Chapman & Hall/CRC, 2008

[3] Powerpoint presentation about Naive Bayes Classifiers from Frank Keller of Saarland University
http://www2.cs.uh.edu/~arjun/courses/nlp/naive_bayes_keller.pdf