School of Data and Computer Science, Sun Yat-sen University

# E12 Naive Bayes

17341015 Hongzheng Chen

November 28, 2019

## Contents

# 1 Datasets

The UCI dataset (`http://archive.ics.uci.edu/ml/index.php`) is the most widely used dataset for machine learning. If you are interested in other datasets in other areas, you can refer to `https://www.zhihu.com/question/63383992/answer/222718972`.

Today's experiment is conducted with the **Adult Data Set** which can be found in `http://archive.ics.uci.edu/ml/datasets/Adult`.

| | | | | | |
|---|---|---|---|---|---|
| **Data Set Characteristics:** | Multivariate | **Number of Instances:** | 48842 | **Area:** | Social |
| **Attribute Characteristics:** | Categorical, Integer | **Number of Attributes:** | 14 | **Date Donated** | 1996-05-01 |
| **Associated Tasks:** | Classification | **Missing Values?** | Yes | **Number of Web Hits:** | 1305515 |

You can also find 3 related files in the current folder, `adult.name` is the description of **Adult Data Set**, `adult.data` is the training set, and `adult.test` is the testing set. There are 14 attributes in this dataset:

```
>50K, <=50K.

1. age: continuous.
2. workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov,
State-gov, Without-pay, Never-worked.
3. fnlwgt: continuous.
4. education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm,
Assoc-voc, 9th, 7th-8th, 12th, Masters, 5. 1st-4th, 10th, Doctorate, 5th-6th,
Preschool.
5. education-num: continuous.
6. marital-status: Married-civ-spouse, Divorced, Never-married, Separated,
Widowed, Married-spouse-absent, Married-AF-spouse.
7. occupation: Tech-support, Craft-repair, Other-service, Sales,
Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct,
Adm-clerical,Farming-fishing,Transport-moving,Priv-house-serv,Protective-serv,
Armed-Forces.
8. relationship: Wife,Own-child,Husband,Not-in-family,Other-relative,Unmarried.
9. race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
10. sex: Female, Male.
11. capital-gain: continuous.
12. capital-loss: continuous.
13. hours-per-week: continuous.
14. native-country: United-States, Cambodia,England,Puerto-Rico,Canada,Germany,
Outlying-US(Guam-USVI-etc),India,Japan,Greece, South,China,Cuba,Iran,Honduras,
Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France,
Dominican-Republic,Laos,Ecuador,Taiwan, Haiti, Columbia,Hungary,Guatemala,
Nicaragua,Scotland,Thailand,Yugoslavia,El-Salvador, Trinadad&Tobago,Peru,Hong,
Holand-Netherlands.
```

**Prediction task is to determine whether a person makes over 50K a year.**

# 2 Naive Bayes

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. It is not a single algorithm for training such classifiers, but a family of algorithms

based on a common principle: all naive Bayes classifiers assume that **the value of a particular feature is independent of the value of any other feature**, given the class variable.

For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features.

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. Bayes theorem states the following relationship, given class variable $y$ and dependent feature vector $x_1$ through $x_n$:

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

Using the naive conditional independence assumption that

$$P(x_i \mid y, x_1, ..., x_{i-1}, x_{x+1}, ..., x_n) = P(x_i \mid y)\,,$$

for all $i$, this relationship is simplified to

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

Since $P(x_1, ..., x_n)$ is constant given the input, we can use the following classification rule:

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

$$\hat{y} = \arg\max_y P(y)\prod_{i=1}^{n} P(x_i \mid y),$$

and we can use Maximum A Posteriori (MAP) estimation to estimate $P(y)$ and $P(x_i \mid y)$, the former is then the relative frequency of class $y$ in the training set.

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

- When attribute values are discrete, $P(x_i \mid y)$ can be easily computed according to the training set.
- When attribute values are continuous, an assumption is made that the values associated with each class are distributed according to Gaussian i.e., Normal Distribution. For example, suppose the training data contains a continuous attribute $x$. We first segment the data by the class, and then compute the mean and variance of $x$ in each class. Let $\mu_k$ be the mean of the values in $x$ associated with class $y_k$, and let $\sigma_k^2$ be the variance of the values in $x$ associated with class $y_k$. Suppose we have collected some observation value $x_i$. Then, the probability distribution of $x_i$ given a class $y_k$, $P(x_i \mid y_k)$ can be computed by plugging $x_i$ into the equation for a Normal distribution parameterized by $\mu_k$ and $\sigma_k^2$. That is,

$$P(x = x_i \mid y = y_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(x_i - \mu_k)^2}{2\sigma_k^2}}$$

## 3 Task

- Given the training dataset `adult.data` and the testing dataset `adult.test`, please accomplish the prediction task to determine whether a person makes over 50K a year in `adult.test` by using Naive Bayes algorithm (C++ or Python), and compute the accuracy.
- Note: keep an eye on the discrete and continuous attributes.
- Please finish the experimental report named `E12_YourNumber.pdf`, and send it to `ai_201901@foxmail.com`

## 4 Codes and Results

The following code is generated by jupyter notebook, please refer to `bayesian.ipynb` and `nb.py`.

```
# To add a new cell, type '# %%'
# To add a new markdown cell, type '# %% [markdown]'

# %%
import os, sys, time
import pandas as pd
import numpy as np

attr_dict = {"age":0, "workclass":1, "fnlwgt":0, "education":1, "education-num":0, "marital
    ↪ -status":1, "occupation":1, "relationship":1, "race":1, "sex":1, "capital-gain":0, "
    ↪ capital-loss":0, "hours-per-week":0, "native-country":1, "salary":0} # 0: continuous
    ↪ , 1: discrete

train_data = pd.read_csv("adult.data",names=attr_dict.keys(),index_col=False)
test_data = pd.read_csv("adult.test",names=attr_dict.keys(),index_col=False,header=0)

def preprocessing(data):
    """
    Select some useful attributes
    """
    # attributes = ["workclass","education","marital-status","occupation","relationship","
        ↪ race","sex","native-country","salary"] # discrete
    attributes = list(attr_dict.keys())
    attributes.remove("fnlwgt")
    attributes.remove("capital-gain")
    attributes.remove("capital-loss")
    return data[attributes]

def fill_data(data,flag=1):
    """
    Fill in missing data (?)
    """
    if flag == 0: # directly remove missing data
        for a in data.columns.values:
            if attr_dict[a]: # discrete
                data = data[data[a] != " ?"] # remove unknown
        return data
    else: # fill data with the most value
        for a in data.columns.values:
            if attr_dict[a]: # discrete
                data.loc[data[a] == " ?",a] = data[a].value_counts().argmax() # view or copy
                    ↪ ? Use loc!
```

```
            else: # continuous
                pass
        return data

# Data cleaning
train_data = preprocessing(train_data)
test_data = preprocessing(test_data)
train_data = fill_data(train_data,1)
test_data = fill_data(test_data,1)


# %%
class NaiveBayesClassifier():
    """
    A Naive Bayes Classifier
    """
    def __init__(self,train_data,attr_dict):
        """
        Initialize and calculate the aprior probability
        """
        self.train_data = train_data
        self.attr_dict = attr_dict

        # calculate the aprior probability P(x_i|y)
        self.prob = {}
        self.prob[" >50K"] = train_data["salary"].value_counts(normalize=True)[" >50K"]
        self.prob[" <=50K"] = 1 - self.prob[" >50K"]
        self.attributes = train_data.columns.values[train_data.columns.values != "salary"]
        less_than_50k = train_data[train_data["salary"] == " <=50K"]
        greater_than_50k = train_data[train_data["salary"] == " >50K"]
        for a in self.attributes:
            if self.attr_dict[a]: # discrete
                count_a_less_than_50k = less_than_50k[a].value_counts()
                count_a_greater_than_50k = greater_than_50k[a].value_counts()
                V = len(train_data[a].unique())
                for xi in train_data[a].unique():
                    # laplacian smoothing
                    self.prob[(xi," <=50K")] = (count_a_less_than_50k.get(xi,0) + 1) / (len(
                        ↪ less_than_50k) + V)
                    self.prob[(xi," >50K")] = (count_a_greater_than_50k.get(xi,0) + 1) / (len
                        ↪ (greater_than_50k) + V)
            else: # continuous
                # use Gaussian aprior
                mu_less_than_50k = np.mean(less_than_50k[a])
                sigma_less_than_50k = np.var(less_than_50k[a])
                self.prob[(a," <=50K")] = lambda x: 1 / np.sqrt(2*np.pi*sigma_less_than_50k)
                    ↪  * np.exp(-(x-mu_less_than_50k)**2/(2*sigma_less_than_50k)) # use
                    ↪ anonymous function
                mu_greater_than_50k = np.mean(greater_than_50k[a])
                sigma_greater_than_50k = np.var(greater_than_50k[a])
                self.prob[(a," >50K")] = lambda x: 1 / np.sqrt(2*np.pi*
                    ↪ sigma_greater_than_50k) * np.exp(-(x-mu_greater_than_50k)**2/(2*
                    ↪ sigma_greater_than_50k))

    def predict(self,test_data):
        """
```

```
        Predict the salary of test data
        """
        acc = 0
        for i, row in test_data.iterrows():
            # calculate P(y|x_1,...,x_n)
            prod = np.array([self.prob[" <=50K"],self.prob[" >50K"]])
            for a in self.attributes:
                xi = row[a]
                if self.attr_dict[a]: # discrete
                    prod[0] *= self.prob[(xi," <=50K")]
                    prod[1] *= self.prob[(xi," >50K")]
                else: # continuous
                    prod[0] *= self.prob[(a," <=50K")](xi)
                    prod[1] *= self.prob[(a," >50K")](xi)

            # find the catagory with the max probability
            catagory = " <=50K" if prod.argmax() == 0 else " >50K"
            if catagory == row["salary"][:-1]: # be careful of "."
                acc += 1
            if i % 1000 == 0:
                print("Finish {}/{}".format(i,len(test_data)))

        acc /= len(test_data)
        print("Accurary: {:.2f}%".format(acc * 100))
        return acc


# %%
nb = NaiveBayesClassifier(train_data,attr_dict)
nb.predict(test_data)
```

I can at most achieve this accuracy...

```
[152] ▷ M↓
      nb = NaiveBayesClassifier(train_data,attr_dict)
      nb.predict(test_data)

    Finish 0/16281
    Finish 1000/16281
    Finish 2000/16281
    Finish 3000/16281
    Finish 4000/16281
    Finish 5000/16281
    Finish 6000/16281
    Finish 7000/16281
    Finish 8000/16281
    Finish 9000/16281
    Finish 10000/16281
    Finish 11000/16281
    Finish 12000/16281
    Finish 13000/16281
    Finish 14000/16281
    Finish 15000/16281
    Finish 16000/16281
    Accurary: 82.18%

    0.8217554204287206
```