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

E10 Naive Bayes (C++/Python)

16337188 Ouyang Runyu

November 9, 2018

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 μ_k be the mean of the values in x associated with class y_k, and let σ_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 μ_k and σ_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 `E10_YourNumber.pdf`, and send it to `ai_2018@foxmail.com`

4 Codes and Results

You can see the code in NB.py

NB.py

```
import numpy as np
from math import pi, exp, sqrt
from scipy import stats
from decimal import Decimal

labelDict = {'<=50K': 0, '>50K': 1}
labelIndex = ['<=50K', '>50K']

# save the probability of y
labelProbability = [0, 0]
# save the mean and var of continous variable
Continuos = [[[0, 0], [0, 0]] for _ in range(14)]
# save the cpt of discrete variable
Discrete = [{} for _ in range(14)]

trainData = [[] for _ in range(32561)]
trainLabel = [0 for _ in range(32561)]
testData = [[] for _ in range(16281)]
testLabel = [[] for _ in range(16281)]

def isContinuous(i):
    return i in [0, 2, 4, 10, 11, 12]

# read data
def readData():
    global trainData
    global testData
    global trainLabel
    global testLabel
```

```
# read training data
with open("adult.data", "r", encoding = 'utf8') as f:
    j = 0
    for line in f.readlines():
        try:
            nowData = str(line)[:-1].replace(' ', '').split(',')
            if nowData[-1] == '':
                continue
            trainLabel[j] = labelDict[nowData[-1]]
            for i, data in enumerate(nowData[:-1]):
                if isContinuous(i):
                    trainData[j].append(int(data))
                else:
                    trainData[j].append(data)
            j += 1
        except Exception as e:
            print("Error occured in line %d"%j)
            print(e)

# read testing data
with open("adult.test", "r", encoding = 'utf8') as f:
    j = 0
    flag = 0
    for line in f.readlines():
        try:
            if flag == 0:
                flag = 1
                continue
            nowData = str(line)[:-2].replace(' ', '').split(',')
            if nowData[-1] == '':
                continue
            testLabel[j] = labelDict[nowData[-1]]
```

```
                for i, data in enumerate(nowData[:-1]):
                    if isContinuous(i):
                        testData[j].append(int(data))
                    else:
                        testData[j].append(data)
                j += 1
            except Exception as e:
                print("Error occured in line %d"%j)
                print(e)

def naiveBayesTrain():
    global trainData
    global trainLabel
    global Continuos
    global Discrete
    global labelProbability

    # compute the probabilty of y
    labelProbability[1] = sum(trainLabel) / 32561
    labelProbability[0] = 1 - labelProbability[1]

    for i in range(14):
        nowData0 = [trainData[j][i] for j in range(32561) if trainLabel[
            j] == 0]
        nowData1 = [trainData[j][i] for j in range(32561) if trainLabel[
            j] == 1]
        # compute the mean and var of continuous variable
        if isContinuous(i):
            nowData0 = np.array(nowData0).reshape((-1, 1))
            nowData1 = np.array(nowData1).reshape((-1, 1))

            mean0 = nowData0.mean()
            mean1 = nowData1.mean()
```

```
            var0 = nowData0.var() * nowData0.shape[0] / (nowData0.shape
                [0] - 1)
            var1 = nowData1.var() * nowData1.shape[0] / (nowData1.shape
                [0] - 1)

            Continuos[i][0][0] = mean0
            Continuos[i][0][1] = var0
            Continuos[i][1][0] = mean1
            Continuos[i][1][1] = var1
        # compute the cpt of discrete variable
        else:
            addProbability = Decimal(1 / len(nowData0))
            for x in nowData0:
                nowKey = x + '_0'
                if nowKey not in Discrete[i]:
                    Discrete[i][nowKey] = addProbability
                else:
                    Discrete[i][nowKey] += addProbability

            addProbability = Decimal(1 / len(nowData1))
            for x in nowData1:
                nowKey = x + '_1'
                if nowKey not in Discrete[i]:
                    Discrete[i][nowKey] = addProbability
                else:
                    Discrete[i][nowKey] += addProbability

# get the probability of a P(Xi = a | y = yj)
# featureIndex:i, featureValue:a, Label:yj
def getProbability(featureIndex, featureValue, Label):
    global Continuos
    global Discrete
```

```
    if isContinuous(featureIndex):
        mean = Continuos[featureIndex][Label][0]
        var = Continuos[featureIndex][Label][1]
        return Decimal(stats.norm.pdf(featureValue, mean, sqrt(var)))

    nowKey = featureValue + '_' + str(Label)
    if nowKey not in Discrete[featureIndex]:
        return 0

    return Discrete[featureIndex][nowKey]

# predict
def predict():
    global testData
    global labelProbability
    global labelIndex
    global testLabel

    P0 = Decimal(labelProbability[0])
    P1 = Decimal(labelProbability[1])

    accuracy = 0
    for j, dataLine in enumerate(testData):
        nowP0, nowP1 = P0, P1
        trueLabel = testLabel[j]
        # compute the probability
        for i, Value in enumerate(dataLine):
            # ignore the 9th and 11th feature
            if i == 11 or i == 9:
                continue
            nowValue = Decimal(getProbability(i, Value, 0))
            nowP0 *= nowValue
```

```
            nowValue = Decimal(getProbability(i, Value, 1))
            nowP1 *= nowValue
        ans = 0 if nowP0 >= nowP1 else 1

        print("Test %d Pred: %s, Label: %s "%(j, labelIndex[ans],
            labelIndex[trueLabel]), end = "")
        if ans == trueLabel:
            print("True")
            accuracy += 1
        else:
            print("False")

    print("Accuracy: " + "%.3f"%(accuracy / 16281 * 100) + "%")

if __name__ == '__main__':
    readData()
    naiveBayesTrain()
    predict()
```

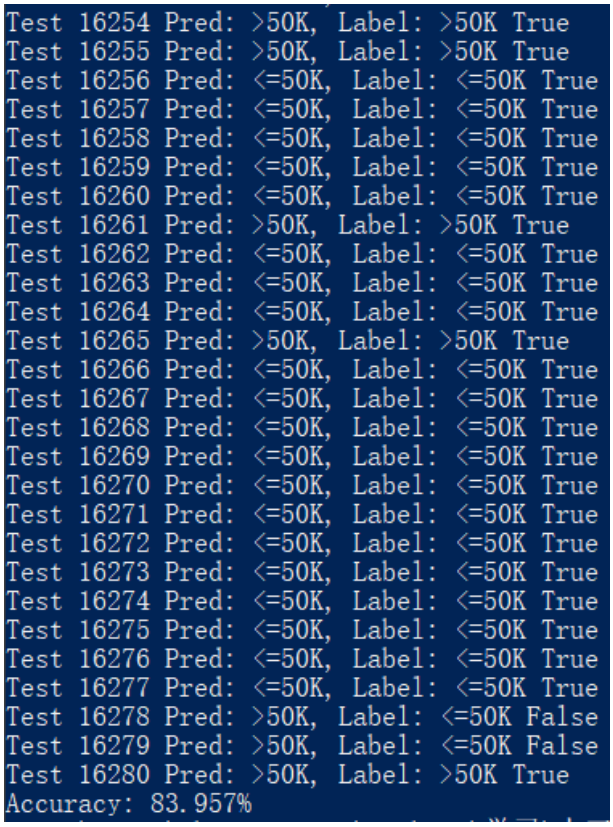

Figure 1: result.png