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

E11 Naive Bayes (C++/Python)

18364066 Lu Yanzuo

November 25, 2020

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 `E11_YourNumber.pdf`, and send it to `ai_2020@foxmail.com`

4 Codes and Results

决策树和朴素贝叶斯都是概率图模型的一种，本次实验所要解决的问题和上一次的 E10 决策树实验是相同的，这里不再过多阐述，而相较于决策树模型，朴素贝叶斯的思想更为简单直接，最根本的原理就是贝叶斯公式。

当我们要推断某个测试样本属于哪一类别时，我们利用贝叶斯公式可以将概率转化为三个概率进行计算，分别是在训练样本中某一类别的概率、在某一类别下某属性取特定值的概率以及在所有类别下某属性取特定值的概率。此外，朴素贝叶斯假定属性之间是彼此独立的，因此可以非常轻松地计算出上述提到的三种概率。

缺失值处理方面，在统计训练样本的某个属性时，我的实现是直接忽略属性缺失的样本不计入总数，而在测试样本中，若某个属性值缺失，则根据缺失属性是离散值还是连续值进行随机取值，若是离散值则根据概率随机选取，若是连续值则在对应的正态分布中随机选值。

连续值处理方面，按照 ta 给定的提示，我们可以假定连续值属性都是符合正态分布的，我们可以利用训练样本统计得到的均值和方差，在测试样本时根据公式计算概率即可。

下图是程序运行结果，可以看出效果还是相当不错的，精度达到了 83.06%，甚至要比我在 E10 的决策树实现的精度还要高，这可能是由于决策树模型中实现的缺失值和连续值处理不够完善导致的。

```
(base) PS C:\Users\YanzuoLu\OneDrive\人工智能\实验\E11_2020_NB> python .\src\main.py
Read train data done.
Test Dataset Accuracy: 83.0600085989804%
```

```python
# Author: Lu Yanzuo
# Date: 2020-11-25
# Description: Naive Bayes classifier


import numpy as np
from scipy import stats
```

```
WORKCLASS = ["Private", "Self-emp-not-inc", "Self-emp-inc", "Federal-gov
    ", "Local-gov",
     "State-gov", "Without-pay", "Never-worked"]
EDUCATION = ["Bachelors", "Some-college", "11th", "HS-grad", "Prof-
    school", "Assoc-acdm",
     "Assoc-voc", "9th", "7th-8th", "12th", "Masters", "1st-4th", "10th",
          "Doctorate",
     "5th-6th", "Preschool"]
MARITAL_STATUS = ["Married-civ-spouse", "Divorced", "Never-married", "
    Separated", "Widowed",
     "Married-spouse-absent", "Married-AF-spouse"]
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"]
RELATIONSHIP = ["Wife", "Own-child", "Husband", "Not-in-family", "Other-
    relative",
     "Unmarried"]
RACE = ["White", "Asian-Pac-Islander", "Amer-Indian-Eskimo", "Other", "
    Black"]
SEX = ["Female", "Male"]
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"]
LABEL = ["<=50K", ">50K"]

CONT_ATTR = [0, 2, 4, 10, 11, 12]
DISC_ATTR = [1, 3, 5, 6, 7, 8, 9, 13]
DISC_ATTR_LEN = [len(WORKCLASS), len(EDUCATION), len(MARITAL_STATUS),
    len(OCCUPATION),
    len(RELATIONSHIP), len(RACE), len(SEX), len(NATIVE_COUNTRY)]
ATTR_NAME = ["age", "workclass", "fnlwgt", "education", "education_num",
    "marital_status",
        "occupation", "relationship", "race", "sex", "capital_gain", "
            capital_loss", "hours_per_week",
        "native_country"]


class Item(object):
    def __init__(self, age, workclass, fnlwgt, education, education_num,
        marital_status,
        occupation, relationship, race, sex, capital_gain, capital_loss,
            hours_per_week,
        native_country, label, test=False):
        self.age = -1 if age == "?" else int(age)
        self.workclass = -1 if workclass == "?" else WORKCLASS.index(
            workclass)
        self.fnlwgt = -1 if fnlwgt == "?" else int(fnlwgt)
        self.education = -1 if education == "?" else EDUCATION.index(
            education)
        self.education_num = -1 if education_num == "?" else int(
            education_num)
        self.marital_status = -1 if marital_status == "?" else
```

```
            MARITAL_STATUS.index(marital_status)
        self.occupation = -1 if occupation == "?" else OCCUPATION.index(
            occupation)
        self.relationship = -1 if relationship == "?" else RELATIONSHIP.
            index(relationship)
        self.race = -1 if race == "?" else RACE.index(race)
        self.sex = -1 if sex == "?" else SEX.index(sex)
        self.capital_gain = -1 if capital_gain == "?" else int(
            capital_gain)
        self.capital_loss = -1 if capital_loss == "?" else int(
            capital_loss)
        self.hours_per_week = -1 if hours_per_week == "?" else int(
            hours_per_week)
        self.native_country = -1 if native_country == "?" else
            NATIVE_COUNTRY.index(native_country)

        self.label = LABEL.index(label) if not test else LABEL.index(
            label[:-1])

    def to_list(self):
        return [self.age, self.workclass, self.fnlwgt, self.education,
            self.education_num,
             self.marital_status, self.occupation, self.relationship,
                self.race, self.sex,
             self.capital_gain, self.capital_loss, self.hours_per_week,
                self.native_country]


def cal_normal_prob(x, mean, std):
    exp = np.exp(-(np.power(x-mean, 2))/(2*np.power(std, 2)))
    prob = (1 / (np.sqrt(2*np.pi)*std)) * exp
    return prob
```

```python
class Dataset(object):
    def __init__(self, attr, label):
        self.attr = np.array(attr, dtype=np.int_)
        self.label = np.array(label, dtype=np.int_)

        self.label_prob = self.cal_label_prob()

        self.disc_prob = []
        for disc in DISC_ATTR:
            self.disc_prob.append(self.cal_attr_prob(disc))

        self.cont_mean = []
        self.cont_std = []
        for cont in CONT_ATTR:
            ret = self.cal_attr_prob(cont)
            self.cont_mean.append(ret[0])
            self.cont_std.append(ret[1])

        self.cond_disc_prob = {}
        for label in range(len(LABEL)):
            self.cond_disc_prob[label] = []
            for disc in DISC_ATTR:
                self.cond_disc_prob[label].append(self.
                    cal_cond_attr_prob(disc, label))

        self.cond_cont_mean = {}
        self.cond_cont_std = {}
        for label in range(len(LABEL)):
            self.cond_cont_mean[label] = []
            self.cond_cont_std[label] = []
            for cont in CONT_ATTR:
                ret = self.cal_cond_attr_prob(cont, label)
```

```
            self.cond_cont_mean[label].append(ret[0])
            self.cond_cont_std[label].append(ret[1])

    def cal_label_prob(self):
        result = []
        total_num = len(self.label)
        for label in range(len(LABEL)):
            result.append(np.sum(self.label == label) / total_num)

        return result

    def cal_attr_prob(self, index):
        temp_attr = self.attr[:,index]
        temp_attr = temp_attr[temp_attr != -1]

        if index in DISC_ATTR:
            result = []
            total_num = len(temp_attr)
            for attr in range(DISC_ATTR_LEN[DISC_ATTR.index(index)]):
                result.append(np.sum(temp_attr == attr) / total_num)

            return result

        mean = np.mean(temp_attr)
        std = np.std(temp_attr)
        return mean, std

    def cal_cond_attr_prob(self, attr_index, label):
        temp_index = self.label == label
        temp_attr = self.attr[temp_index, attr_index]
        temp_attr = temp_attr[temp_attr != -1]

        if attr_index in DISC_ATTR:
```

```
            result = []
            total_num = len(temp_attr)
            for attr in range(DISC_ATTR_LEN[DISC_ATTR.index(attr_index)
                ]):
                result.append(np.sum(temp_attr == attr) / total_num)

            return result

        mean = np.mean(temp_attr)
        std = np.std(temp_attr)
        return mean, std

    def test(self, item):
        item_list = item.to_list()

        for disc in DISC_ATTR:
            if item_list[disc] == -1:
                disc_prob = self.disc_prob[DISC_ATTR.index(disc)]
                item_list[disc] = np.random.choice(list(range(len(
                    disc_prob))), p=disc_prob)

        for cont in CONT_ATTR:
            if item_list[cont] == -1:
                cont_mean = self.cont_mean[CONT_ATTR.index(cont)]
                cont_mean = self.cont_std[CONT_ATTR.index(cont)]
                item_list[cont] = np.random.normal(cont_mean, cont_mean)

        max_label = None
        max_label_prob = None
        for label in range(len(LABEL)):
            prob = self.label_prob[label]

            for disc in DISC_ATTR:
```

```
                prob *= self.cond_disc_prob[label][DISC_ATTR.index(disc)
                    ][item_list[disc]]
                prob /= self.disc_prob[DISC_ATTR.index(disc)][item_list[
                    disc]]

            for cont in CONT_ATTR:
                prob *= cal_normal_prob(
                    item_list[cont],
                    self.cond_cont_mean[label][CONT_ATTR.index(cont)],
                    self.cond_cont_std[label][CONT_ATTR.index(cont)]
                )
                prob *= cal_normal_prob(
                    item_list[cont],
                    self.cont_mean[CONT_ATTR.index(cont)],
                    self.cont_std[CONT_ATTR.index(cont)]
                )

            if max_label_prob is None or prob > max_label_prob:
                max_label = label
                max_label_prob = prob

        return item.label == max_label



if __name__ == "__main__":
    train_data_path = "dataSet/adult.data"
    test_data_path = "dataSet/adult.test"

    train_attr = []
    train_label = []

    with open(train_data_path, "r") as train_data:
```

```
    line = train_data.readline()
    while line.strip():
        line_split = line.strip().split(',')
        args = tuple(list(map(lambda x: x.strip(), line_split)))
        item = Item(*args)

        train_attr.append(item.to_list())
        train_label.append(item.label)

        line = train_data.readline()
print("Read train data done.")

dataset = Dataset(train_attr, train_label)
# print("dataset.label_prob:", dataset.label_prob)
# print("dataset.disc_prob:", dataset.disc_prob)
# print("dataset.cont_mean:", dataset.cont_mean)
# print("dataset.cont_std:", dataset.cont_std)
# print("dataset.cond_disc_prob:", dataset.cond_disc_prob)
# print("dataset.cond_cont_mean:", dataset.cond_cont_mean)
# print("dataset.cond_cont_std:", dataset.cond_cont_std)

total_num = 0
correct_prediction = 0
with open(test_data_path, "r") as test_data:
    # 丟掉第一行
    line = test_data.readline()

    line = test_data.readline()
    while line.strip():
        total_num += 1

        line_split = line.strip().split(',')
        args = tuple(list(map(lambda x: x.strip(), line_split)))
```

```
        item = Item(*args, test=True)
        if dataset.test(item):
            correct_prediction += 1

        line = test_data.readline()

print("Test Dataset Accuracy: {}%".format(correct_prediction /
    total_num * 100))
```