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

E12 Naive Bayes (C++/Python)

Yixin Zhang

November 29, 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 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 `E12_YourNumber.pdf`, and send it to `ai_201901@foxmail.com`

4 Codes and Results

The implementation of Naive Bayes classifier is much easier than that of decision tree algorithm. I can achieve an accuracy of 83.269%.

Please refer to the following pages, which are exported from Jupyter Notebook. For a better experience, I put a demo on my website, please visit https://down.jeddd.com/temp/NaiveBayes.html.

1 朴素贝叶斯

[1]:
```
import numpy as np
import pandas as pd
from scipy import stats
```

1.1 正确地读取数据

注意原始数据文件的格式，对其进行正确地处理后读入两个 DataFrame：adult_data_df 是训练集，adult_test_df 是测试集。DataFrame 中名为 "50K" 的列为标签（即分类）。

读取数据的方法与上个实验（决策树算法）完全相同。

[2]:
```
col_names = ['age', 'workclass', 'fnlwgt', 'education', 'education-num',␣
 ↪'marital-status', 'occupation', 'relationship', 'race', 'sex',␣
 ↪'capital-gain', 'capital-loss', 'hours-per-week', 'native-country', '50K']
adult_data_df = pd.read_csv('dataset/adult.data', index_col=False, header=None,␣
 ↪names=col_names, sep=', ', engine='python')

adult_test_df = pd.read_csv('dataset/adult.test', skiprows=[0],␣
 ↪index_col=False, header=None, names=col_names, sep=', ', engine='python')
adult_test_df['50K'] = adult_test_df['50K'].map(lambda x: x[:-1])  # 去除行末的
点
```

1.2 补充缺失值

通过对数据的基本观察得知，缺失值所在的列均为离散属性，因此只需要对离散缺失值进行补全即可，本例数据集上无需考虑连续型数据的补全。我采用的方法是使用该列出现次数最多的值（即众数）代替缺失值。

补充缺失值的方法与上个实验（决策树算法）完全相同。

[3]:
```
# 补充缺失值，
print('[adult.data]')
mode_df = adult_data_df.mode()  # 众数
for col in adult_data_df:
    if '?' in adult_data_df[col].tolist():
        missing_count = adult_data_df[col].value_counts()['?']   # 缺失值的个数
        adult_data_df[col] = adult_data_df[col].replace('?', mode_df[col][0])
        print('{}: {} missing values are replaced with "{}"'.format(col,␣
 ↪missing_count, mode_df[col][0]))

print('--------------------------------')
print('[adult.test]')
mode_df = adult_test_df.mode()  # 众数
for col in adult_test_df:
    if '?' in adult_test_df[col].tolist():
        missing_count = adult_test_df[col].value_counts()['?']   # 缺失值的个数
        adult_test_df[col] = adult_test_df[col].replace('?', mode_df[col][0])
```

```
        print('{}: {} missing values are replaced with "{}"'.format(col,␣
↪missing_count, mode_df[col][0]))
```

```
[adult.data]
workclass: 1836 missing values are replaced with "Private"
occupation: 1843 missing values are replaced with "Prof-specialty"
native-country: 583 missing values are replaced with "United-States"
-------------------------------
[adult.test]
workclass: 963 missing values are replaced with "Private"
occupation: 966 missing values are replaced with "Prof-specialty"
native-country: 274 missing values are replaced with "United-States"
```

1.3 预测和测试

对于测试集中的每个样本，使用朴素贝叶斯方法进行预测，然后与标签比对，并统计准确率。

[4]:
```
# 连续型属性
continuous_attrs = {'age', 'fnlwgt', 'education-num', 'capital-gain',␣
↪'capital-loss', 'hours-per-week'}

# 计算概率
def probability(df, attr, value):
    """
    计算数据集中某属性为某值的概率。
    Params:
        df: 数据集。
        attr_: 属性名。
        value: 属性值。
    Return:
        对于离散型属性，返回给定属性中值等于给定值的比例；
        对于连续型属性，返回对应高斯分布的概率密度函数值。
    """
    attr_series = df[attr]
    if attr in continuous_attrs:  # 连续型属性
        mean = attr_series.mean()  # 期望
        var = attr_series.var()    # 方差
        return stats.norm.pdf(value, loc=mean, scale=np.sqrt(var))  # 高斯分布的
概率密度
    else:  # 离散型属性
        return list(attr_series).count(value) / len(df)
```

[5]:
```
def predict(sample):
    """
    对一个样本进行预测。
    Params:
        sample: 待测样本。
    Returns:
```

```
        预测分类结果。
    """
    class_list = ['<=50K', '>50K']  # 所有类别
    max_prob = 0
    max_class = ''

    # 遍历所有可能的分类（本例中只有两种分类）
    for class_ in class_list:
        class_df = adult_data_df[adult_data_df['50K']==class_]  # 按类划分数据集
        prob = adult_data_df['50K'].value_counts().get('<=50K', 0) /␣
↪len(adult_data_df)  # 初始化为类的先验概率

        for attr in sample.index:
            if attr == '50K':  # 标签列不是属性，要跳过
                continue
            prob *= probability(class_df, attr, sample[attr])  # 累乘每个属性在数
据集中出现的概率

        if prob >= max_prob:
            max_prob = prob
            max_class = class_

    return max_class  # 返回概率最大的类作为预测结果
```

[6]:
```
correct_count = 0
for i in range(len(adult_test_df)):
    sample = adult_test_df.iloc[i]
    if predict(sample) == sample['50K']:
        correct_count += 1
```

[7]:
```
print('准确率：{:.3%}'.format(correct_count / len(adult_test_df)))
```

准确率：83.269%