

2016~2017春季学期机器学习概论

朴素贝叶斯分类器实验

实验报告

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

计43

唐玉涵

2014011328

2016~2017春季学期机器学习概论

朴素贝叶斯分类器实验

实验报告

1、任务说明

在本实验中，我们需要完成一个朴素贝叶斯分类器，并且在一组真实的数据集上测试它的分类性能。具体包括以下几个部分：

1. 在真实数据集上实现一个机器学习的具体算法（特指朴素贝叶斯算法），并且改善它的性能；
2. 为其性能给出合理的评价方式；
3. 分析实验结果。

2、实验设计

我采用自己比较熟悉的java编程语言进行实验，与python相比可能运行时间较快一些。共编写两个类：Sample.java 和 Test.java。Sample是为训练数据和测试数据编写的类，Test为利用朴素贝叶斯方法实现的分类器实现与测试。以下分别对其进行介绍：

Sample类

针对数据集的14个特征属性和1个结果属性，我采用了以下的结构对其进行存储：

```
public class Sample {
    int age;
    String workclass;
    int fnlwgt;
    String education;
    int education_num;
    String marital_status;
    String occupation;
    String relationship;
    String race;
    String sex;
    int capital_gain;
    int capital_loss;
    int hours_per_week;
    String native_country;
    String result;
    String my_result;
    public Sample(){}
    public Sample(int age, String workclass, int fnlwgt, String education,
        int education_num, String marital_status, String occupation,
        String relationship, String race, String sex, int capital_gain, int capital_loss,
        int hours_per_week, String native_country, String result){
        this.age=age;
        this.fnlwgt = fnlwgt;
        this.workclass=workclass;
        this.education=education;
        this.education_num=education_num;
        this.marital_status=marital_status;
        this.occupation=occupation;
        this.relationship=relationship;
        this.race=race;
        this.sex=sex;
        this.capital_gain=capital_gain;
        this.capital_loss=capital_loss;
        this.hours_per_week=hours_per_week;
        this.native_country=native_country;
        this.result=result;
    }
}
```

其中Sample的前15个属性与数据集中属性相对应，最后一个属性my_result为为测试集所准备，用来存放对测试数据的预测结果，与result属性进行比较即可对性能作出评价。

需要说明的是，我针对每一个属性都编写了set函数和get函数，这样不直接对Sample类进行取值和修改，符合类的开闭原则，见下图：

```
public int getAge() { return age; }
public void setAge(int age) { this.age = age; }
public String getWorkclass() { return workclass; }
public void setWorkclass(String workclass) { this.workclass = workclass; }
public int getFnlwgt() { return fnlwgt; }
public void setFnlwgt(int fnlwgt) { this.fnlwgt = fnlwgt; }
public String getEducation() { return education; }
public void setEducation(String education) { this.education = education; }
public int getEducation_num() { return education_num; }
public void setEducation_num(int education_num) { this.education_num = education_num; }

public String getMarital_status() { return marital_status; }
public void setMarital_status(String marital_status) { this.marital_status = marital_status; }
public String getOccupation() { return occupation; }
public void setOccupation(String occupation) { this.occupation = occupation; }

public String getRelationship() { return relationship; }
public void setRelationship(String relationship) { this.relationship = relationship; }
public String getRace() { return race; }
public void setRace(String race) { this.race = race; }
public String getSex() { return sex; }
public void setSex(String sex) { this.sex = sex; }
public int getCapital_gain() { return capital_gain; }
public void setCapital_gain(int capital_gain) { this.capital_gain = capital_gain; }
public int getCapital_loss() { return capital_loss; }
public void setCapital_loss(int capital_loss) { this.capital_loss = capital_loss; }
public int getHours_per_week() { return hours_per_week; }
public void setHours_per_week(int hours_per_week) { this.hours_per_week = hours_per_week; }
public String getNative_country() { return native_country; }
public void setNative_country(String native_country) { this.native_country = native_country; }
public String getResult() { return result; }
public void setResult(String result) { this.result = result; }
public String getMy_result() { return my_result; }
public void setMy_result(String my_result) { this.my_result = my_result; }
```

Test类

对训练集和测试集分别采用ArrayList<Sample>的形式进行储存，命名为list和list1。读入训练集数据文件后，逐行扫描并对每行按照逗号分割开存入一个Sample对象中（list中）。

针对测试集，采用同样的方式将数据存入list1中。针对list1中的每一个测试对象，逐个扫描list中的训练对象，首先判断训练对象结果属于“>50K.”还是“<=50K.”类别，然后在这两类下分别逐个扫描训练对象的14个特征属性，统计在该类别下与测试对象各个属性相同的对象个数。同时记录训练集中“>50K.”和“<=50K.”这两类的个数。

使用上面统计的数目，利用贝叶斯公式计算该测试对象分属“>50K.”和“≤50K.”的概率（相对值，其和并不为1），相对大的那个结果即记为其预测结果。所依据的原理如下：

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

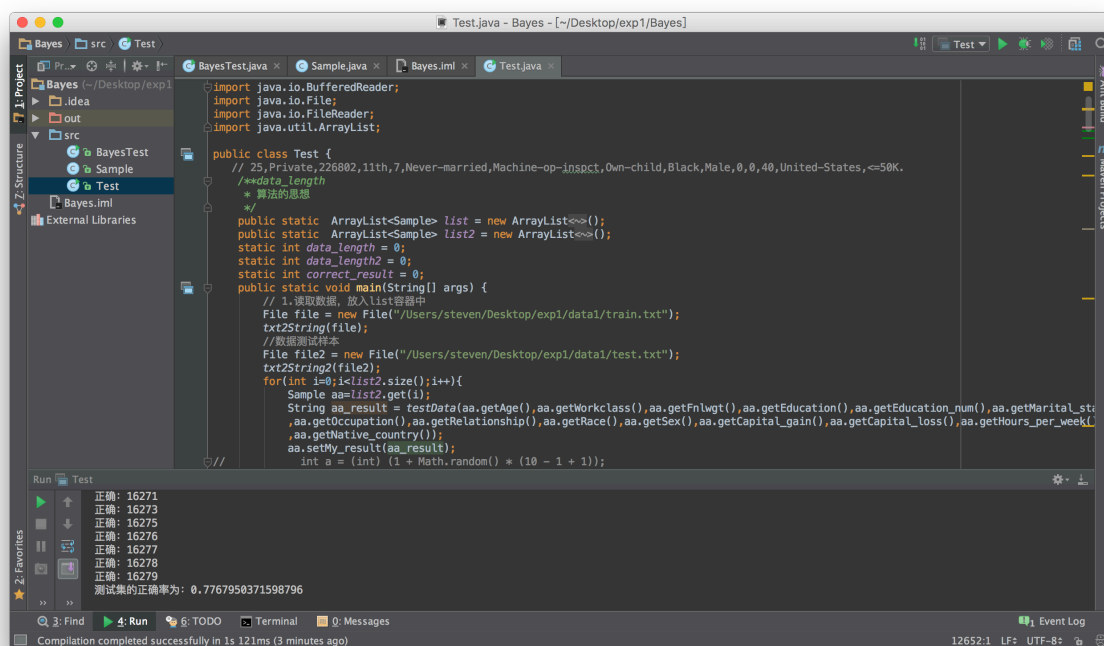
为了对分类的性能做出评价，我才用了准确率作为衡量指标，即

$$\bullet \text{ Accuracy} = \frac{\text{number of correctly classified records}}{\text{number test records}}$$

统计针对全体测试集数据的正确预测个数，除以测试集大小，即可得到该算法的准确率。

3、实验结果

如上述方法实现的朴素贝叶斯分类器（未经过优化），得到的准确率约为77.67%，见下图：



The screenshot shows an IDE window titled "Test.java - Bayes - [~/Desktop/exp1/Bayes]". The code defines a `Test` class with a `main` method. It reads training data from `data1/train.txt` and test data from `data1/test.txt`. The `main` method iterates over the test data, uses the `testData` method to predict the class for each sample, and counts the number of correct predictions. The output shows 8 correct predictions out of 10 test records, resulting in an accuracy of 0.7767950371598796.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.ArrayList;

public class Test {
    // 25,Private,226802,11th,7,Never-married,Machine-op-inspct,Own-child,Black,Male,0,0,40,United-States,<=50K.
    /**data_length
     * 算法的思想
     */
    public static ArrayList<Sample> list = new ArrayList<>();
    public static ArrayList<Sample> list2 = new ArrayList<>();
    static int data_length = 0;
    static int data_length2 = 0;
    static int correct_result = 0;
    public static void main(String[] args) {
        // 1.读取数据，放入list容器中
        File file = new File("/Users/steven/Desktop/exp1/data1/train.txt");
        txt2String(file);
        //数据测试样本
        File file2 = new File("/Users/steven/Desktop/exp1/data1/test.txt");
        txt2String2(file2);
        for(int i=0;i<list2.size();i++){
            Sample aa=list2.get(i);
            String aa_result = testData(aa.getAge(),aa.getWorkclass(),aa.getFnlwgt(),aa.getEducation(),aa.getEducation_num(),aa.getMarital_status(),aa.getOccupation(),aa.getRelationship(),aa.getRace(),aa.getSex(),aa.getCapital_gain(),aa.getCapital_loss(),aa.getHours_per_week(),aa.getNative_country());
            aa.setMy_result(aa_result);
            int a = (int) (1 + Math.random() * (10 - 1 + 1));
        }
    }
}
```

Run Test

正确: 16271
正确: 16273
正确: 16275
正确: 16276
正确: 16277
正确: 16278
正确: 16279
测试集的正确率为: 0.7767950371598796

可见数据的分类性能比较普通。由于训练集中“ $\leq 50K$.”的数据比例占到了76.07%，与之相对应的若直接预测测试集数据全部为“ $\leq 50K$.”也可得到76%左右的准确率，提升空间比较大。

同时，我将过程中每一个测试对象分属“ $> 50K$.”和“ $\leq 50K$.”的概率打印出来，惊奇地发现其中有很多两个概率均为0。这时我的算法将其判断为“ $\leq 50K$.”，即类似于上段所述直接预测得结果的方法，准确率也比较相近。研究其原因，发现是“零概率”问题造成的，针对其以及其他部分的优化详见下一部分。

4、实验优化与分析

1. 训练集大小的影响

实验所给出的全部训练集合大小为32561条数据，集合数目较大，下对训练集的大小对分类性能的影响进行研究：

分别使用5%，50%和100%的训练集数据来进行训练，5%和50%的数据均为从训练集中随机产生，并各测试5组，求其中最大、最小和平均准确率。见下表：

	5%	50%
1	81.47%	81.75%
2	82.20%	81.24%
3	80.83%	81.61%
4	82.59%	81.60%
5	81.71%	81.65%
最大	82.59%	81.75%
最小	80.83%	81.24%
平均	81.76%	81.57%

作为对比，100%的训练数据集合准确率为81.68%（加入了下面几个分点所述优化，故比76.07%提高了不少）。

分析：

从准确率的结果上看，5%的数据集合优于100%的数据集合优于50%的数据集合，最好的准确率甚至达到了82.59%，但是结果的波动较大，不很稳定。

可能的解释是朴素贝叶斯并不因更多的数据而执行的更好。朴素贝叶斯算法需要足够的数据来了解每个属性的概率关系，这些属性独立于输出变量。而模型忽略给定属性间的交互关系，我们不需要这些有交互关系的样本，因此通常情况下比其他算

法需要更少的数据。更进一步，它不会因小规模样本而产生过拟合的数据。所以如果没有太多的训练数据，朴素贝叶斯算法也许是比较好的选择。

2. 零概率问题

上一部分已发现算法中存在大量的“零概率”问题，这种问题出现的原因是：很多测试集中的数据的某一属性在训练集中可能并没有（一个重要的例子是，一个分类属性具有数值，然而却没有出现在训练集中。这种情况下，模型会赋0概率并且不能够进行预测），即有下式：

$$\hat{P}(y = c | x_1, \dots, x_i = k, \dots, x_n) = 0$$

这显然并不合理，将直接导致测试对象概率为0，无法给出比较合理的预测结果。

优化方法：采用拉普拉斯平滑方法对数据进行处理：

$$\hat{P}(x_i = k | y = c) = \frac{\#\{y=c, x_i=k\} + \alpha}{\#\{y=c\} + M\alpha}$$

上式子中 α 取1，即为拉普拉斯平滑，其原理为假定训练样本很大时，每个分量 x 的计数加1造成的估计概率变化可以忽略不计，但可以方便有效的避免零概率问题。

这样对概率计算进行平滑处理后，实验结果的准确率得到了很大的提升，由76.07%提升为81.07%。

其他平滑处理的方式限于时间原因，未进行尝试，但拉普拉斯算法在理论上可靠，对原计算改动不大，实际效果也比较好，推荐作为处理时的首选。对于上式中 α 的取值，可以不一定为1，选取其他参数也许会有更好的结果，本次实验暂未进行讨论。

3. 连续属性和缺失属性的处理

(1) 连续属性：

针对6个连续的数值型特征属性，我才用了分类离散法和高斯分布估值法两种方法进行处理。以下详细说明：

分类法：

对于训练集中的连续属性，我先扫描统计了其最大和最小值，得到下表：

属性	最小值	最大值	均分类数
age	17	90	5
fnlwgt	20534	1455435	8
education_num	1	16	9
capital_gain	0	99999	5
capital_loss	0	2415	5
hours_per_week	1	99	5

这样，连续的数值被分为离散的有限的类，方便下一步的计算，也能有效地避免“零概率”问题的出现。

实验结果的准确率提升为81.68%

分析：

这样直接对连续属性进行等分，在实际操作中较为简便，但也可能存在着数据集中在某一类中的问题（如某一属性有很多值为0）。可改进的方法有：统计（进一步可画出）连续属性的分布（图），按照分布的不同等分为间隔不等的类别，使得分到每一类中的属性数目均等，这样更符合统计规律，也许会进一步提高实验结果的准确率；另一种分类方法是对比连续属性每一个数值下面最可能的分类结果，然后根据相邻数值结果相反的地方划分边界，这样也更加利用到了数据集合本身的一些特征，可能会有更好的结果，限于时间原因，可留待后期研究。

高斯分布估值法：

原理见下公式：

$$P(x_i | y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

即针对连续属性，不采用常规的方式计算 $P(x_i | y)$

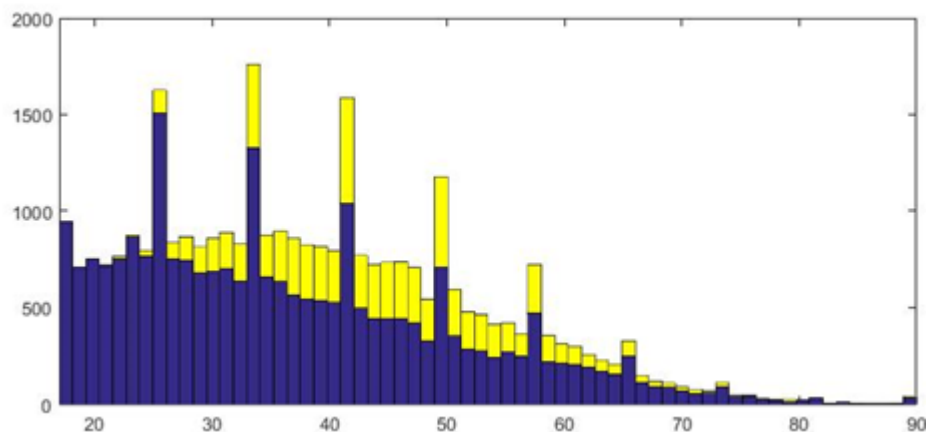
而是先算出每一个连续属性在“>50K.”和“<=50K.”两种结果下的平均值和方差，然后利用高斯分布进行概率估计，再代入之后的计算之中。

我本来对此方法比较有信心，然而实验结果反而准确率有所下降，只有77.87%（100%数据集，5%数据集时平均准确率约为80%）。

分析：

也许由于数据的几个连续属性并不是按照类似高斯分布的形式进行分布的，所以强行采用高斯分布进行概率估计所得的结果可能偏差会比较大，这也导致了实验结果准确率的下降。

我也把第一个属性age的分布情况利用matlab进行了绘图，结果见下（蓝色部分为“ $\leq 50K$ ”，黄色部分为“ $> 50K$ ”，可见其分布确实与高斯分布相差甚远，不适合利用此公式进行估计，可以考虑利用指数函数或者其他函数进行拟合，用以估计其概率值。



(2) 缺失属性：

针对缺失属性，我采用了三种处理方法，直接去掉训练集中带有缺失属性的数据、将其单独分为一类、用该特征出现最多的属性值代替它。以下详细进行说明：

去除法：

遇到带有“?”的属性，直接将该条数据弃用，这样保证了全部的训练数据都是完整的，但问题是测试集中也存在缺失属性，反而会带来“零概率”问题，故这种方法效果预期不是很好。

实际测试准确率为81.09%，相比原来有所下降。

单独归类法：

即将缺少的数据“?”单独作为一类属性值，也即不作处理时默认这样做，好处是可以挖掘缺失数据之间可能存在的某种联系，用以对结果进行预测。

实际测试准确率为81.68%。

常见值替代法：

此方法为赋予缺失属性训练集中该属性的最常见值，即“众数”，该方法看起来较为合理，利用到了数据集本身的特性。

实际准确率为81.62%，不如单独归类法。

分析：

这三种处理的方法比较起来，单独归类法效果最好，直接去除法效果最差，其实还是有一些出乎意料的，原本认为常见值替代法应该更好一些（计算量也较大一些）。我个人分析，原因可能是数据集中缺失的属性本身存在一定的规律，或者可能隐藏着一些隐式的信息，如某类属性涉嫌隐私等。将其分为一类正好可利用其内在联系。

另一种可能的处理方法为赋给缺失数据被分到的结果类中该属性的最常见值，但缺点是测试集中的缺失属性并没有已知结果，仍存在引入“零概率”问题的可能，在本次实验中由于时间有限，未做详细研究。

4. 选定属性研究

通过查阅相关资料，朴素贝叶斯方法有时只选用给出的某一些属性，反而会得到更好的分类效果。本次实验，我也对相关问题进行了简单的研究。

作为对照，我选用50%训练数据，拉普拉斯平滑处理，连续属性分类法，缺失属性单独归类法，实验结果准确率为81.46%

分别去掉14个属性，研究去掉之后的准确率的变化，见下表：

属性	准确率 (%)	对比	属性	准确率 (%)	对比
age	81.40	-	relationship	81.77	+
workclass	81.60	+	race	81.72	+
fnlwgt	81.56	+	sex	81.93	+
education	80.30	-	capital-gain	81.29	-
education-num	80.31	-	capital-loss	81.23	-
marital-status	81.29	-	hours-per-week	81.48	+
occupation	80.68	-	native-country	81.58	+

其中去掉后准确率提升的属性有7个，下降的属性也有7个。

其中 relationship, race 和 sex 的准确率提升很明显，故进一步进行尝试：

- (1) 去掉 race 和 sex，准确率81.96%；
- (2) 去掉 relationship, race 和 sex，准确率81.80%；
- (3) 去掉 sex，采用100%的训练集数据，准确率82.09%；
- (4) 去掉 race 和 sex，采用100%的训练集数据，准确率82.00%。

可见，去除掉某些属性，确实可以提高实验结果的准确率。这这里，去掉 sex，并采用100%的训练集数据效果最佳，可达到82.09%。

分析：

选择那些能更好地描述预测变量的数据属性，可能确实会有更好的结果。在朴素贝叶斯算法中，每一种属性的概率计算都是独立于训练集的，故可以使用搜索算法去探索不同属性组合在一起的概率，并评估它们预测的性能。由于时间有限，我只做了简单的常识，之后可以做更多的搜索来探索更好的属性组合。

5. 其他

(1) 使用对数概率

概率经常是非常小的数值。为了计算联合概率，需要将所有概率连乘在一起。而当一个小数乘以另一个小数时，将得到一个更小的数。它可能遇到浮点数精度难题，例如下溢。为了避免这种情况，使用对数概率空间（对概率进行对数运算，对数具有很好的保序特性）能够很好地工作，因为朴素的贝叶斯预测只需要知道哪一类拥有较大的概率，而不需要知道具体的概率值是多少。

我在实验中确实使用了对数概率进行计算，效果较好。

(2) 冗余属性的去除

我在实验中发现数据的 education 属性和 education-num 属性关联性非常强。从原理上看，朴素贝叶斯的性能可能因数据包含高度相关的属性而下降。这是因为高度相关的属性在模型中引入两次，这样增加了这一属性的重要性。但根据上页表中的数据来看，去掉这二者中的任意一个都导致了准确性的下降。这或许与我实验的训练样本具有一定随机性（50%训练集数据）有关，但也有可能教育的因素对于最后的分类结果确实非常重要，所以被重复强调后反而结果更好。

这也启示我们，不能盲目相信理论，而要从实际数据出发，通过实际实验来验证我们的想法。

5、实验小结

本次实验中，我第一次在真实数据集上实现了一个机器学习的具体算法（朴素贝叶斯算法），并且通过各种尝试来改善它的性能，并对各种实验结果进行了自己的分析。

实验优化部分结束后，我还对不同方法的搭配进行了一些小尝试，最高的一次实验结果准确率达到了82.97%（5%随机训练集，连续数据采取高斯分布），这个结果令我感到十分意外，因为高斯分布的处理结果在我之前的分析中使得结果准确率大幅下降。同样条件我又进行了5次实验，平均的准确率为82.03%，远高于利用全体训练集时的高斯分布处理结果。其中的原因还有待进一步的研究，可能是高斯分布对于较小数据集时效果更好，也可能是随机性的结果偏差。

在实验中我也发现了一些数据集的小规律，比如说给定的训练集和测试集中连续属性值均无缺失，这为我们的计算带来了很大的便利（平均数、方差，甚至是读入数据等均有便利）。另外限于时间原因，还有很多可以继续研究探讨的地方，比如平滑系数的选取，特定属性的选取等。

最后，这次实验令我收获很大，不仅自己动手实现了一个具有实际意义的分类器，更让我意识到算法写好之后的优化和调整往往也是十分重要，更重要的是去思考每一种方法对实验结果产生相应影响的原因。同时，十分感谢张敏老师和张帆助教对我的帮助。