南開大學
Nankai University

南　开　大　学

计 算 机 学 院

机器学习实验报告

实验四 朴素贝叶斯分类器

姓名：王泳鑫

学号：1911479

年级：2019级

专业：计算机科学与技术

指导教师：卫金茂

2021 年 11 月 5 日

摘要

关键字：朴素贝叶斯，**Machine Learning ， Deep Learning**

目录

一、 实验描述

（一） 实验内容

Most Popular Data Set中的wine数据集（对意大利同一地区声场的三种不同品种的酒做大量分析所得出的数据）

（二） 实验要求

基本要求：

- 采用分层采样的方式将数据集划分为训练集和测试集。
- 给定编写一个朴素贝叶斯分类器，对测试集进行预测，计算分类准确率。

中级要求：

- 使用测试集评估模型，得到混淆矩阵，精度，召回率，F值。

二、 代码实现

首先时读取数据的部分，通过numpy库的loadtxt读取数据，并且遍历数据集，将不同类别的数据放入三个list中，返回这三个列表。

```
# 读取数据
def read_data(path):
    data = list(np.loadtxt(path, delimiter=','))
    # 打乱数据，为方便训练集和测试集划分
    random.shuffle(data)
    data = np.array(data)
    sample1 = []
    sample2 = []
    sample3 = []
    for each in data:
        if each[0] == 1:
            sample1.append(each)
        elif each[0] == 2:
            sample2.append(each)
        else:
            sample3.append(each)
    return np.array(sample1), np.array(sample2), np.array(sample3)
```

然后我们要对数据集进行一个划分，基本要求中写到，需要采用分层抽样的的方法划分训练集和测试集, 先将所有个体样本按照某种特征划分为几个类别，这一步我们在读取数据的函数中已经实现了，然后从每个类别中使用随机抽样或等距抽样的方法选择个体组成样本。

```
def train_test_split(sample1, sample2, sample3, fold, idx):
 train_sample1 = []
 train_sample2 = []
 train_sample3 = []
 test_sample = []
 for i in range(sample1.shape[0]):
     if i % fold != idx:
         train_sample1.append(sample1[i])
```

```
        else:
            test_sample.append(sample1[i])
    for i in range(sample2.shape[0]):
        if i % fold != idx:
            train_sample2.append(sample2[i])
        else:
            test_sample.append(sample2[i])
    for i in range(sample3.shape[0]):
        if i % fold != idx:
            train_sample3.append(sample3[i])
        else:
            test_sample.append(sample3[i])
    train_sample1 = np.array(train_sample1)
    train_sample2 = np.array(train_sample2)
    train_sample3 = np.array(train_sample3)
    train_label1 = train_sample1[:, 0]
    train_label2 = train_sample2[:, 0]
    train_label3 = train_sample3[:, 0]
    train_sample1 = train_sample1[:, 1:]
    train_sample2 = train_sample2[:, 1:]
    train_sample3 = train_sample3[:, 1:]
    test_sample = np.array(test_sample)
    test_label = test_sample[:, 0]
    test_sample = test_sample[:, 1:]
    return train_sample1, train_label1, train_sample2, train_label2, \
           train_sample3, train_label3, test_sample, test_label
```

接下来我们需要计算极大后验概率，公式如下：

$$c_{MAP} = \operatorname{argmax}_{c_k \in C} P(c_k \mid x) = \operatorname{argmax}_{c_k \in C} \frac{P(x \mid c_k) P(c_k)}{P(x)} = \operatorname{argmax}_{c_k \in C} P(x \mid c_k) P(c_k) \tag{1}$$

```
   def cal_para(sample):
    mean = np.mean(sample, 0)
    var = np.var(sample, 0)
    return mean, var


def cal_prob(test_vec, train_sample, total):
    mean, var = cal_para(train_sample)
    # 根据高斯函数计算
    prob = (1 / np.sqrt(2*np.pi*var)) * np.exp(-np.square(test_vec - mean) / (2*var))
    # 先验概率
    prior = train_sample.shape[0] / total
    likelihood = 1
    for each in prob:
        # 连乘条件概率
        likelihood *= each
    return prior*likelihood
```

然后是预测标签的函数,通过上面的计算概率函数，选择概率最大的作为预测标签，代码如下：

```
    def decide_label(test_sample, train_sample1, train_sample2, train_sample3):
    total = train_sample1.shape[0] + train_sample2.shape[0] + train_sample3.shape[0]
```

```
    label = []
    score = []
    for each in test_sample:
        prob1 = cal_prob(each, train_sample1, total)
        prob2 = cal_prob(each, train_sample2, total)
        prob3 = cal_prob(each, train_sample3, total)
        # 选择概率最大的作为预测标签
        if max([prob1, prob2, prob3]) == prob1:
            label.append(1)
        elif max([prob1, prob2, prob3]) == prob2:
            label.append(2)
        else:
            label.append(3)
        score.append(prob1)
    return np.array(label), np.array(score)
```

下面则是本次实验交叉验证的函数，参数sample分别是对应数据集中的三个类别，fold代表折数

```
def cross_valid(sample1, sample2, sample3, fold):
    acc = []
    for i in range(fold):
        train_sample1, train_label1, train_sample2, train_label2, \
        train_sample3, train_label3, \
        test_sample, test_label = train_test_split(sample1, sample2, sample3, fold, i)
        pred_label, pred_score = decide_label(test_sample, train_sample1,
                                              train_sample2, train_sample3)
        correct = 0
        for j in range(test_label.shape[0]):
            if pred_label[j] == test_label[j]:
                correct += 1
        print(i, 'acc:', correct / test_label.shape[0])
        acc.append(correct / test_label.shape[0])
    return np.mean(acc)
```

后面，是提高要求的部分，我们可以通过下面结果分析的函数来构造混淆矩阵，精度，召回率和F值,这一部分只要根据公式就可以顺利地写出来：

```
    def result_analysis(sample1, sample2, sample3):
    train_sample1, train_label1, train_sample2, train_label2, \
    train_sample3, train_label3, \
    test_sample, test_label = train_test_split(sample1, sample2, sample3, fold, 1)
    pred_label, pred_score = decide_label(test_sample, train_sample1, train_sample2,
                                          train_sample3)
    cm = np.zeros((3, 3))
    # 构造混淆矩阵
    for i in range(test_label.shape[0]):
        cm[int(test_label[i]) - 1, pred_label[i] - 1] += 1
    tp1 = cm[0, 0]
    tp2 = cm[1, 1]
    tp3 = cm[2, 2]
    fn1 = cm[0, 1] + cm[0, 2]
    fn2 = cm[1, 0] + cm[1, 2]
    fn3 = cm[2, 0] + cm[2, 1]
    fp1 = cm[1, 0] + cm[2, 0]
    fp2 = cm[0, 1] + cm[2, 1]
```

```
    fp3 = cm[0, 2] + cm[1, 2]
    tn1 = cm[1, 1] + cm[1, 2] + cm[2, 1] + cm[2, 2]
    tn2 = cm[0, 0] + cm[0, 2] + cm[2, 0] + cm[2, 2]
    tn3 = cm[0, 0] + cm[0, 1] + cm[1, 0] + cm[1, 1]
    # 根据公式计算、和precisionrecallF1
    precision1 = tp1 / (tp1 + fp1)
    precision2 = tp2 / (tp2 + fp2)
    precision3 = tp3 / (tp3 + fp3)
    recall1 = tp1 / (tp1 + fn1)
    recall2 = tp2 / (tp2 + fn2)
    recall3 = tp3 / (tp3 + fn3)
    f1 = (2*precision1*recall1) / (precision1 + recall1)
    f2 = (2*precision2*recall2) / (precision2 + recall2)
    f3 = (2*precision3*recall3) / (precision3 + recall3)
    return cm, precision1, precision2, precision3, recall1, recall2, recall3, f1, f2,
                                                f3
```

三、 实验结果展示与分析

对于交叉验证，我们得到的结果如图5所示：

```
0 acc: 1.0
1 acc: 0.9444444444444444
2 acc: 0.9722222222222222
3 acc: 0.9428571428571428
4 acc: 1.0
5 fold cross validation mean acc: 0.9719047619047618
Result analysis of the second fold cross validation
```

图 1: 交叉验证结果

我们可以看到5折交叉检验的平均准确率达到了0.9719，可以看到该分类器得到的分类结果可信度还是比较高。

根据公式我们得到的混淆矩阵如图5所示：

```
Confusion matrix:
[[11.  1.  0.]
 [ 0. 13.  1.]
 [ 0.  0. 10.]]
```

图 2: 混淆矩阵

经过result_analysis函数的计算，我们得到的精度如下图5所示：

```
precision for 1: 1.0
precision for 2: 0.9285714285714286
precision for 3: 0.9090909090909091
```

图 3: 精度

召回率如下图5所示：

```
recall for 1: 0.9166666666666666
recall for 2: 0.9285714285714286
recall for 3: 1.0
```

图 4: 召回率

F如下图5所示：

```
F1-score for 1: 0.9565217391304348
F1-score for 2: 0.9285714285714286
F1-score for 3: 0.9523809523809523
```

图 5: F值

NKU