实验三：朴素贝叶斯分类器

杨扬　计算机科学与技术　1611132

摘　要

本次实验是朴素贝叶斯分类器的学习及实现，所使用的数据集是 Most Popular Data Set 中的 wine 数据集。本文档针对于此问题，分析数据集本身的特性，介绍了朴素贝叶斯分类器及各种模型评估方法的原理及特性，结合于实验数据集本身设计了算法，并基于源程序对实验的设计进行了解析，最后对实验的结果进行了分析。简单来说，本文档在对实验所有要求均做出实现的同时，也成功地进行了多方面的拓展，以极高的质量完成了此次实验。

关键词：朴素贝叶斯、分类、模型评估

目录

1 问题描述

在机器学习中，朴素贝叶斯分类器是一系列以假设特征之间强（朴素）独立下运用贝叶斯定理为基础的简单概率分类器。

朴素贝叶斯自 20 世纪 50 年代已广泛研究。在 20 世纪 60 年代初就以另外一个名称引入到文本信息检索界中，并仍然是文本分类的一种热门（基准）方法，文本分类是以词频为特征判断文件所属类别或其他（如垃圾邮件、合法性、体育或政治等等）的问题。通过适当的预处理，它可以与这个领域更先进的方法（包括支持向量机）相竞争。它在自动医疗诊断中也有应用。

朴素贝叶斯分类器是高度可扩展的，因此需要数量与学习问题中的变量（特征/预测器）成线性关系的参数。最大似然训练可以通过评估一个封闭形式的表达式来完成，只需花费线性时间，而不需要其他很多类型的分类器所使用的费时的迭代逼近。

本次实验的数据集文件是 *wine.data*。具体来说，此数据集是对意大利同一地区生产的三种不同品种的酒做大量分析所得出的数据，数据的存储形式如图 1 所示，文件中包含 178 行、14 列的数字，列与列之间以逗号分割。其中第 1 列为标签，其余 13 列为特征，各维特征均为连续值，其实际意义如下所示：

1. Alcohol
2. Malic acid
3. Ash
4. Alcalinity of ash
5. Magnesium
6. Total phenols
7. Flavanoids
8. Nonflavanoid phenols
9. Proanthocyanins
10. Color intensity
11. Hue
12. OD280/OD315 of diluted wines

13. Proline

178 行数据中根据标签的不同分为 3 组，每组分别为 59 行、71 行和 48 行。此次实验需要我们基于后 13 列的特征构造朴素贝叶斯分类器，然后根据测试数据的特征计算出预测的标签，与第 1 列的实际标签进行比较，计算出混淆矩阵、精确率、召回率和 F1 值，绘制出 ROC 曲线，并基于此计算 AUC 值。

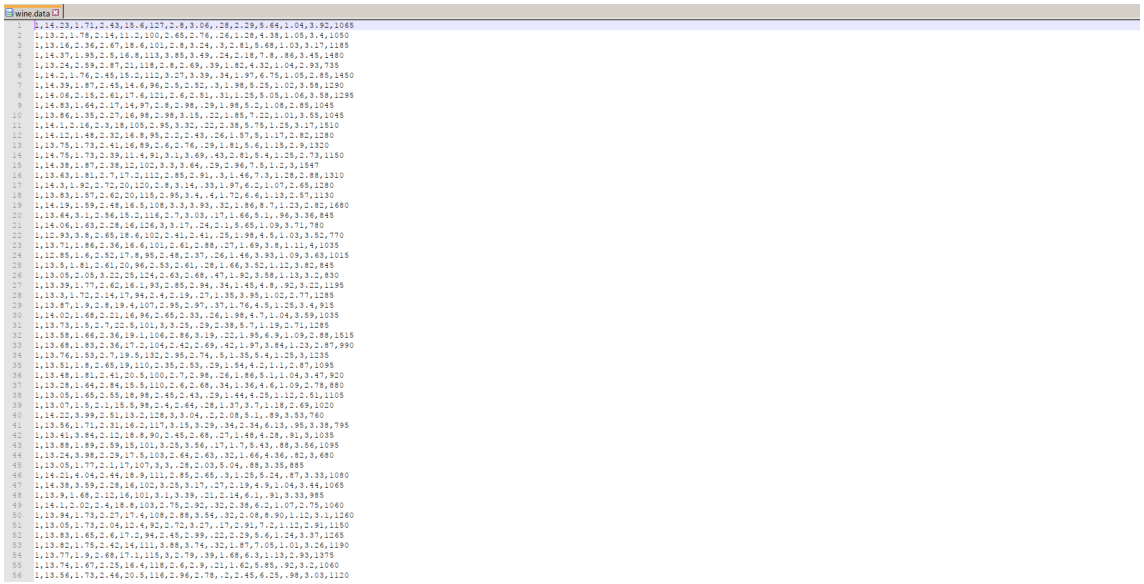

图 1: 数据存储形式

2 贝叶斯定理

本次实验我们需要设计出一个朴素贝叶斯分类器，而其理论基础正是源于概率论中的贝叶斯定理。简单来说，贝叶斯定理是关于随机事件 A 和 B 的条件概率的一则定理，其常见的表示公式如下：

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

公式中各项在贝叶斯定理中的含义为：

- P(A|B) 是已知 B 发生后 A 的条件概率，也由于得自 B 的取值而被称作 A 的后验概率。
- P(A) 是 A 的先验概率（或边缘概率）。之所以称为” 先验” 是因为它不考虑任何 B 方面的因素。

- P(B|A) 是已知 A 发生后 B 的条件概率，也由于得自 A 的取值而被称作 B 的后验概率。
- P(B) 是 B 的先验概率或边缘概率。

按照以上定义，贝叶斯定理即可被理解为后验概率与先验概率和相似度的乘积成正比。

3 朴素贝叶斯分类器

朴素贝叶斯分类器是基于贝叶斯定理提出的一种想法十分简单的分类算法。简单来说，在进行朴素贝叶斯分类时，对于给出的待分类项，求解在此项出现的条件下各个类别出现的概率，哪个最大，就认为此待分类项属于哪个类别。

朴素贝叶斯分类器的具体内容可分解为如下：

首先，设 $X = \{F_1, F_2, \ldots, F_m\}$ 为一个待分类项，其中 F_i 代表 X 的一维特征。然后定义标签类别集合为 $C = \{y_1, y_2, \ldots, y_n\}$。

此时，朴素贝叶斯概率模型是一个条件概率模型：

$$p(C|x) = p\left(C|F_1, \ldots, F_n\right)$$

独立的类别变量 C 有若干类别 $y_1, y_2, \ldots, y_n$，条件依赖于若干特征变量 $F_1, \ldots, F_n$。但问题在于如果特征数量 n 较大或者每个特征能取大量值时，基于概率模型列出概率表变得不现实。所以我们修改这个模型使之变得可行。贝叶斯定理有以下式子：

$$p\left(C|F_1, \ldots, F_n\right) = \frac{p(C)p\left(F_1, \ldots, F_n|C\right)}{p\left(F_1, \ldots, F_n\right)}$$

从上式可以看出，实际上我们只关心分式中的分子部分，因为分母不依赖于 C 而且特征 F_i 的值是给定的，于是分母可以认为是一个常数。这样分子就等价于联合分布模型：

$$p\left(C, F_1, \ldots, F_n\right)$$

重复使用链式法则，可将该式写成条件概率的形式，如下所示：

$$
\begin{aligned}
& p\left(C, F_1, \ldots, F_n\right) \\
& \propto p(C) p\left(F_1, \ldots, F_n \mid C\right) \\
& \propto p(C) p\left(F_1 \mid C\right) p\left(F_2, \ldots, F_n \mid C, F_1\right) \\
& \propto p(C) p\left(F_1 \mid C\right) p\left(F_2 \mid C, F_1\right) p\left(F_3, \ldots, F_n \mid C, F_1, F_2\right) \\
& \propto p(C) p\left(F_1 \mid C\right) p\left(F_2 \mid C, F_1\right) p\left(F_3 \mid C, F_1, F_2\right) p\left(F_4, \ldots, F_n \mid C, F_1, F_2, F_3\right) \\
& \propto p(C) p\left(F_1 \mid C\right) p\left(F_2 \mid C, F_1\right) p\left(F_3 \mid C, F_1, F_2\right) \ldots p\left(F_n \mid C, F_1, F_2, F_3, \ldots, F_{n-1}\right)
\end{aligned}
$$

推导到此步，则需应用到朴素贝叶斯分类器中的“朴素”二字，简单来说，其思想即是假定样本的每个特征与其他特征都不相关，即每个特征 F_i 对于其他特征 $F_j, j \neq i$ 是条件独立的，这也就等同于：

$$
p\left(F_i \mid C, F_j\right)=p\left(F_i \mid C\right)
$$

所以，对于 $i \neq j$，联合分布模型可以表达为：

$$
\begin{aligned}
p\left(C \mid F_1, \ldots, F_n\right) & \propto p\left(C, F_1, \ldots, F_n\right) \\
& \propto p(C) p\left(F_1 \mid C\right) p\left(F_2 \mid C\right) p\left(F_3 \mid C\right) \cdots p\left(F_n \mid C\right) \\
& \propto p(C) \prod_{i=1}^{n} p\left(F_i \mid C\right)
\end{aligned}
$$

基于以上推导，便可得到标签类别变量 C 的条件概率分布即为：

$$
p\left(C \mid F_1, \ldots, F_n\right)=\frac{p(C) \prod_{i=1}^{n} p\left(F_i \mid C\right)}{p\left(F_1, \ldots, F_n\right)}
$$

前文已经提到，上式的分母与不同样本中 $F_1, \ldots, F_n$ 的取值无关，可认定为一个常数。因此，我们便可构造朴素贝叶斯分类器模型。

对于每一条待分类样本 $x=\{f_1, f_2, \ldots, f_n\}$，有：

$$
\text{classify}(x)=\text{classify}\left(f_1, \ldots, f_n\right)=\underset{y}{\text{argmax}} p(C=y) \prod_{i=1}^{n} p\left(F_i=f_i \mid C=y\right)
$$

至此，我们便完成了对朴素贝叶斯分类器的推导。

3.1 高斯朴素贝叶斯

前文中推导出的公式中的参数需要根据实际的样本数据进行估计，当样本某一特征的类型为离散值时，我们可以简单地通过统计训练样本中各个划分在每个类别中出现的频率，即可用频率值来代替概率值。

但当特征的类型为连续值时，我们则需要假定该特征符合高斯分布，即：

$$P(x=v|y)=\frac{1}{\sqrt{2\pi\sigma_y^2}}e^{-\frac{(v-\mu_y)^2}{2\sigma_y^2}}$$

此概率密度函数中的 μ_y、σ_y 等值可通过训练样本进行估计计算得出，当有测试数据输入时，将对应维度的特征输入上式计算即可。

至此，我们便完成了朴素贝叶斯分类器的理论推导部分。

4 模型评估方法及 ROC 曲线

为对模型进行多方面的评估，本次实验还要求我们计算分类器的精确率、召回率和 F1 值。简单来说，这些度量指标通常针对二分类问题提出，在本次实验中，我们需要将某一类当作正类，其余两类当作负类来进行计算。

具体来说，精确率和召回率度量指标依赖于混淆矩阵来计算，混淆矩阵的定义如下图所示：

	Predicted	
	Positive	Negative
Actual True	TP	FN
Actual False	FP	TN

图 2: 混淆矩阵

图中各项的含义为：

- True Positive(真正，TP)：将正类预测为正类数。
- True Negative(真负，TN)：将负类预测为负类数。
- False Positive(假正，FP)：将负类预测为正类数误报。
- False Negative(假负，FN)：将正类预测为负类数 → 漏报。

而精确率则定义为：

$$Precision = \frac{TP}{TP + FP}$$

召回率定义为：

$$Recall = \frac{TP}{TP + FN}$$

基于精确率和召回率，便可计算分类器的 F1 值：

$$F = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

在计算以上的度量指标后，实验的提高要求是 ROC 曲线的绘制。简单来说，ROC 分析的是二元分类模型，也就是输出结果只有两种类别的模型，例如：（阳性 / 阴性）、（有病 / 没病）等。

ROC 空间将伪阳性率（FPR）定义为 X 轴，真阳性率（TPR）定义为 Y 轴：

- TPR：在所有实际为阳性的样本中，被正确地判断为阳性之比率。$TPR = TP/(TP + FN)$
- FPR：在所有实际为阴性的样本中，被错误地判断为阳性之比率。$FPR = FP/(FP + TN)$

在本次实验中，我们可以以第 1 类作为二分类基准，将阈值设置为测试样本在第 1 类上的概率值，即可绘制出 ROC 曲线。

最后，将模型的 ROC 曲线画出来后，我们可以计算曲线下面积做为模型优劣的指标，即 AUC（Area under the Curve of ROC）值。AUC 的估算公式如下所示：

$$AUC = \sum_{i=1}^{n-1} (x_i - x_{i-1})(y_i + y_{i-1})$$

至此，模型度量方法和 ROC 曲线的原理分析部分结束。

5 实验及源程序分析

5.1 实验环境

本次实验在 Ubuntu 18.04 下进行，实验所使用的编程语言为 Python 3.6，IDE 为 Pycharm 2018.3.2，主要使用了 numpy、random、matplotlib 等 Python 包。

5.2 朴素贝叶斯分类器源程序分析

由于此次实验的内容较少，实验数据规模也不算太大，所以我们在一个 py 文件里即可完成本次实验，具体来说，程序的步骤如下所示：

首先我们在 Python 中导入所需要用到的 Python 包：

```
import numpy as np
import random
import matplotlib.pyplot as plt
```

然后我们需要导入相应的实验数据，*wine.data* 文件中数据的分割方式为逗号，我们采用 numpy 中的 loadtxt() 方法导入即可，再调用 random 中的 shuffle 方法将数据行序打乱，以降低训练过程中的偏倚，最后根据第 1 列的标签将数据分为 3 类：

```
def read_data(path):
    data = list(np.loadtxt(path, delimiter=','))
    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)
```

读入数据后，由于本次实验没有提供专门的测试集，所以我们需要进行训练集与验证集的划分，与前几次实验一样，我们仍旧采用多折交叉验证的方式：

```
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
```

然后我们便可开始朴素贝叶斯分类器的核心内容了，此次实验数据集中各维特征均为连续值，需要我们使用高斯分布来进行参数估计，因此我们定义一个参数估计函数如下：

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

得到高斯分布中的各参数后，我们开始构造概率密度函数公式，根据前文中推导得出的分类器函数，将所有条件概率进行连乘，公式中的 $P(C=y)$ 采用训练样本中各类的比例进行估计，得到最后的概率值：

```
def cal_prob(test_vec, train_sample, total):
    mean, var = cal_para(train_sample)
    prob = (1 / np.sqrt(2*np.pi*var)) * np.exp(-np.square(test_vec -
        mean) / (2*var))
    prior = train_sample.shape[0] / total
    likelihood = 1
    for each in prob:
        likelihood *= each
    return prior*likelihood
```

在得到测试集数据在 3 类标签上的概率值后，将概率值最大的 1 类作为测试样本的预测标签，为后续实验的 ROC 曲线绘制需要，这里我们将测试样本在第 1 类上的概率大小也一并返回：

```
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)
        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)
```

为方便对模型的调用，我们定义一个交叉验证的接口函数，在此函数中完成交叉验证的内容，并计算出多折交叉验证情况下各算法的平均准确率：

```
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)
```

至此，我们便完成了朴素贝叶斯分类器的搭建，即实验要求中的基础部分。

5.3 模型评估源程序分析

在上一小节的程序中，我们简单地计算了一下分类器在 5 折交叉验证情况下的平均准确率，但在机器学习中，对一个模型性能的度量，往往不能只计算准确率，而需要考虑多种不同情形下模型的表现。根据前文介绍的各种模型度量方法，我们定义一个结果分析函数，计算出在 4:1 的训练测试划分比的情况下模型的混淆矩阵，及分别针对于 3 类的精确率、召回率和 F1 值：

```
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))
```

```
    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]
    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
```

5.4 ROC 曲线绘制源程序分析

对于 ROC 曲线的绘制，我们针对于第 1 类进行，通过分别调整阈值至测试样本在第 1 类上的概率值，计算出阳性率（FPR）和真阳性率（TPR），再调用 matplotlib 中的方法进行曲线绘制，最后根据 AUC 值计算公式计算出 AUC 值即可：

```
def draw_roc(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)
    fpr = []
    tpr = []
    for i in np.argsort(pred_score)[::-1]:
        threshold = pred_score[i]
        if threshold == np.max(pred_score):
            fpr.append(0)
            tpr.append(0)
            continue
        elif threshold == np.min(pred_score):
            fpr.append(1)
            tpr.append(1)
            continue
        tp = 0
        fp = 0
        fn = 0
        tn = 0
        for j in np.argsort(pred_score)[::-1]:
            if pred_score[j] >= threshold:
                if test_label[j] == 1:
                    tp += 1
                else:
                    fp += 1
            else:
                if test_label[j] == 1:
                    fn += 1
                else:
                    tn += 1
        fpr.append(fp / (fp + tn))
        tpr.append(tp / (tp + fn))
    auc = 0
    for i in range(1, len(fpr)):
        auc += 0.5 * (fpr[i] - fpr[i - 1]) * (tpr[i] + tpr[i - 1])
    plt.title('ROC')
    plt.plot(fpr, tpr, color='green', label='ROC')
    plt.xticks([0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
```

```
    plt.yticks([0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
    plt.legend()
    plt.xlabel('False positive rate')
    plt.ylabel('True positive rate')
    plt.show()
    return auc
```

最后，本次实验的主函数体为如下：

```
if __name__ == '__main__':
    data_path = 'wine.data'
    fold = 5
    sample1, sample2, sample3 = read_data(data_path)
    acc = cross_valid(sample1, sample2, sample3, fold)
    print(fold, 'fold cross validation mean acc:', acc)
    cm, precision1, precision2, precision3, recall1, recall2, recall3,
        f1, f2, f3 = result_analysis(sample1, sample2, sample3)
    print('Result analysis of the second fold cross validation')
    print('Confusion matrix:', cm)
    print('precision for 1:', precision1)
    print('precision for 2:', precision2)
    print('precision for 3:', precision3)
    print('recall for 1:', recall1)
    print('recall for 2:', recall2)
    print('recall for 3:', recall3)
    print('F1-score for 1:', f1)
    print('F1-score for 2:', f2)
    print('F1-score for 3:', f3)
    auc = draw_roc(sample1, sample2, sample3)
    print('AUC:', auc)
```

至此，我们便完成了本次实验的源程序解析部分。

6 实验结果分析

实验中验证方式我设置为 5 折交叉验证，得到分类器的运行结果为如下：

```
0 acc: 1.0
1 acc: 0.9444444444444444
2 acc: 0.9722222222222222
3 acc: 0.9428571428571428
4 acc: 0.9705882352941176
5 fold cross validation mean acc: 0.9660224089635854
Result analysis of the second fold cross validation
Confusion matrix:
[[11.  1.  0.]
 [ 1. 13.  0.]
 [ 0.  0. 10.]]
precision for 1: 0.9166666666666666
precision for 2: 0.9285714285714286
precision for 3: 1.0
recall for 1: 0.9166666666666666
recall for 2: 0.9285714285714286
recall for 3: 1.0
F1-score for 1: 0.9166666666666666
F1-score for 2: 0.9285714285714286
F1-score for 3: 1.0
AUC: 0.9930555555555555
```

图 3: 分类器性能分析

从图 3 可以看出，我所实现的朴素贝叶斯分类器取得了非常好的效果，从朴素贝叶斯原理的角度进行分析，我们可以判断本次实验数据集的 13 维特征间的确具有较低的相关性，从而我们在假设其概率分布彼此独立的条件下并没有收到多大的影响。另一方面，从最后得到的混淆矩阵来看，分类器无论在精确率还是召回率方面都达到了较好的效果，证明此分类器在绝大多数场景下都具有较强的普适性。

此外，我以第 1 类作为基准，绘制了分类器的 ROC 曲线如下：

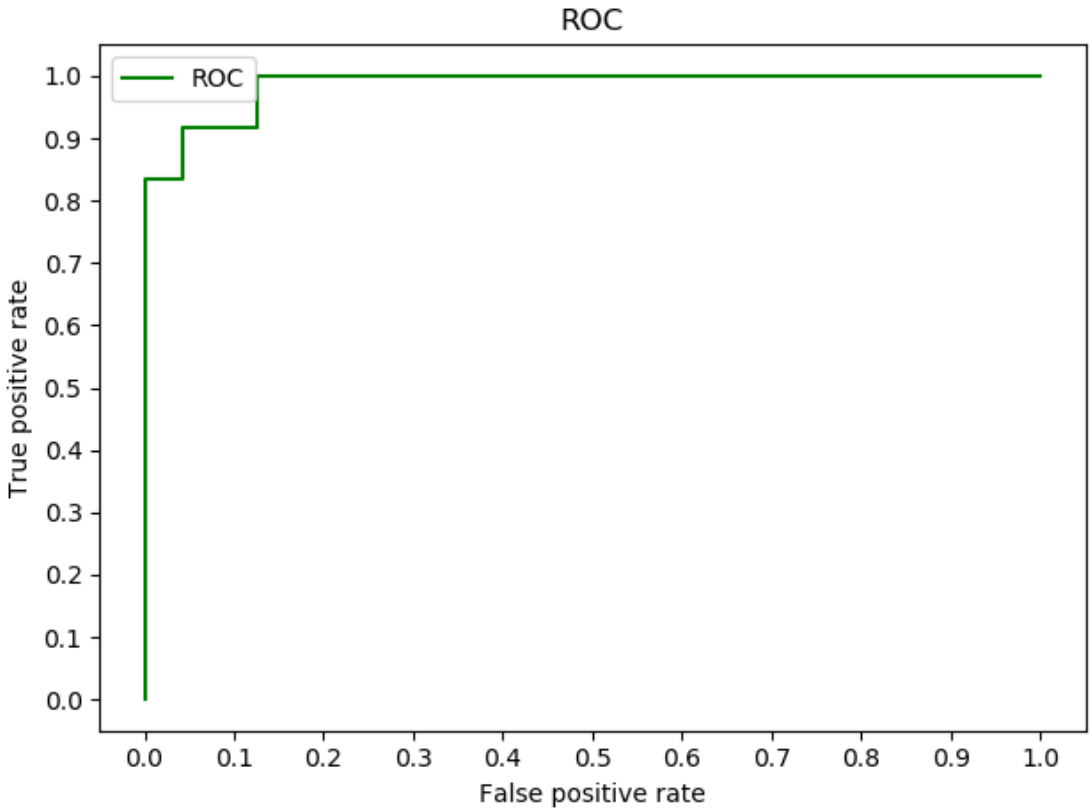

图 4: 以第 1 类为二分类基准的 ROC 曲线

由图 4 可以看出，分类器已经极为接近理想情况，此外，曲线的 AUC 值也达到了非常高的程度，接近一个完美的预测。

7 总结

朴素贝叶斯分类器是机器学习中一种极其优美的方法，在其推导和代码实现中我也对概率论的基础知识和课上所学进行了巩固。虽然感觉实验最后要求绘制的 ROC 曲线不太适用于分析本次实验，有点为出题而出题的感觉，不过总的来说此次实验我收获颇丰，对机器学习也有了更深层次的理解。

参考文献

[1] Thomas M. Mitchell. 1997. Machine Learning (1 ed.). McGraw-Hill, Inc., New York, NY, USA.

[2] https://zh.wikipedia.org/wiki/Wikipedia