山东大学 计算机科学与技术 学院

机器学习（双语） 课程实验报告

学号：201705130113	姓名：黄瑞哲	班级：计科 17.3
实验题目：朴素贝叶斯		
实验学时：2	实验日期： 2019.10.30	

实验目的：

1. 了解朴素贝叶斯模型，包括假设、求解等。
2. 深化学习并进一步掌握的朴素贝叶斯的原理以及实现方式
3. 根据训练数据建立朴素贝叶斯模型，完成对入学建议的推荐。

硬件环境：

Intel Core i5-8300H @ 2.3GHz

软件环境：

Windows10 Pro 1903

Python 3.7

Visual Studio Code 1.38.

实验步骤与内容：

朴素贝叶斯模型的核心是贝叶斯公式，因此建立模型前有属性独立性假设，即各个特征之间是独立同分布的。

根据条件概率贝叶斯公式可以得到

$$P(y|x) = \frac{P(y) \cdot P(x|y)}{P(x)} = \frac{P(x,y)}{P(x)}$$

因此朴素贝叶斯就是要找到合适的 y 使得在给定的特征情况下，取当前标签的概率最大，即求得

$$\underset{y}{\operatorname{argmax}} P(y|x)$$

从计算上来看，我们需要知道$P(x,y), P(x)$，其中$P(x)$我们无从得知，但是由于每一个式子中都带有$P(x)$，在我们最大化$P(y|x)$时不需要考虑它，因此我们只需要$P(x,y)$ 最大即可。这里的一个核心假设在于各个特征之间时相互独立的，因此我们可以使用链式法则

$$P(Y=y, X_1=x_1, X_2=x_2, \cdots, X_n=x_n)$$

$$= P(X_1=x_1, X_2=x_2, \cdots, X_n=x_n|Y=y)P(Y=y)$$

$$= P(Y=y)\prod_{j=1}^{n} P\left(X_j=x_j\middle|Y=y\right)$$

$$= p(y)\prod_{j=1}^{n} p_j(x_j|y)$$

由此可以推导出对数-似然函数

$$
\begin{aligned}
\ell(\Omega) &= \log \prod_{i=1}^{m} p(x^{(i)}, y^{(i)}) \\
&= \sum_{i=1}^{m} \log p(x^{(i)}, y^{(i)}) \\
&= \sum_{i=1}^{m} \log \left(p(y^{(i)}) \prod_{j=1}^{n} p_j(x_j^{(i)} \mid y^{(i)}) \right) \\
&= \sum_{i=1}^{m} \log p(y^{(i)}) + \sum_{i=1}^{m} \sum_{j=1}^{n} \log p_j(x_j^{(i)} \mid y^{(i)})
\end{aligned}
$$

可以使用拉格朗日乘子法求得最大值，因为我们有约束条件

$$\sum_{\mathrm{y}} p(y) = 1$$

$$\sum_{\mathrm{x}} p_j(x \mid y) = 1, \ \forall y, j$$

可以解得

$$p(y) = \frac{count(y)}{m} = \frac{\sum_{i=1}^{m} \mathbf{1}(y^{(i)} = y)}{m}, \ \forall y$$

$$p_j(x \mid y) = \frac{count_j(x \mid y)}{count(y)} = \frac{\sum_{i=1}^{m} \mathbf{1}(y^{(i)} = y \wedge x_j^{(i)} = x)}{\sum_{i=1}^{m} \mathbf{1}(y^{(i)} = y)}, \ \forall x, y, j$$

根据这个公式便可以建立朴素贝叶斯模型，但是考虑到训练数据中会有一些 y 没有出现，会导致分母为 0，可以加上拉普拉斯平滑解决（认为缺省标签出现的概率相同）。

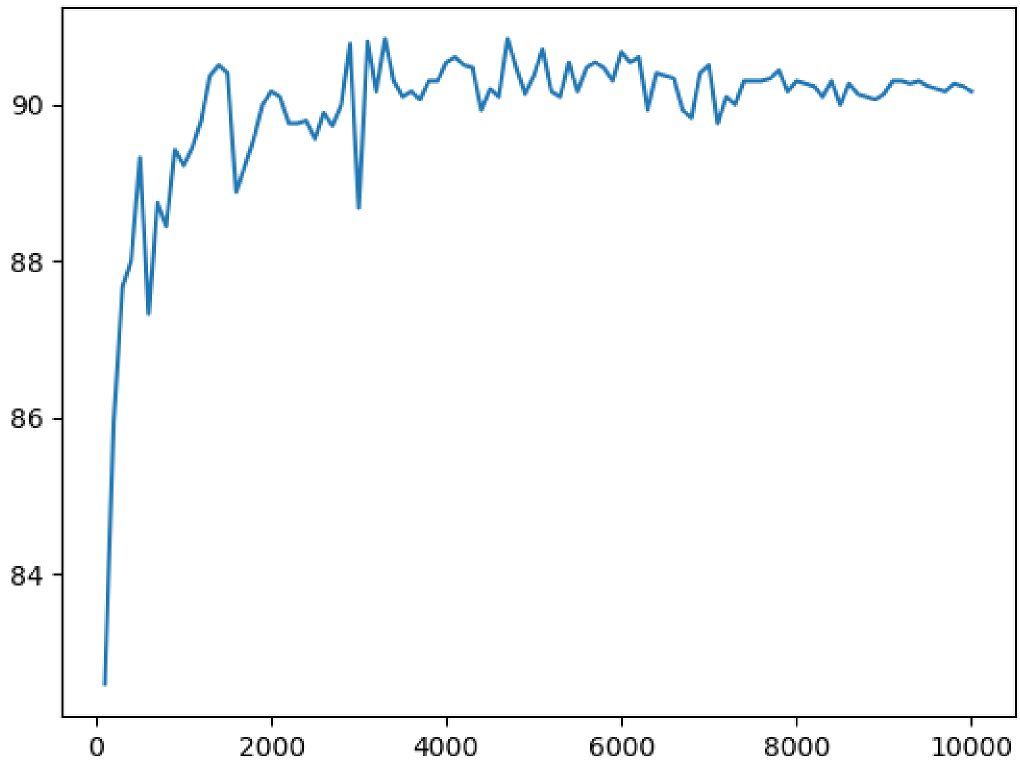
结论分析与体会：

对于任务 1：

我们根据训练数据计算得到每一个$p_j(x|y)$，在预测得时候枚举每一种可能的 y 并计算出其对应得概率，我们取概率最高得标签 y 作为该数据的标签。经过测试数据的测试，准确率达 90.34%。

```
accuracy=90.34%
```

对于任务 2：
我们进行 100 次取样，每次取样多选择 100 个数据，在取样的数据基础上训练朴素贝叶斯模型，并根据这个模型来测试其准确性。



由上图可以看出，随着样本不断增大，模型的准确性在不断提升，最终在 90%左右震荡，与任务一的结果相吻合。由此可以看出，训练数据规模越大，模型的准确性越高，因此为了提高贝叶斯分类器的准确性，我们提供的训练数据越多越好。但是较多的训练数据也会使得训练分类器的时间变长。

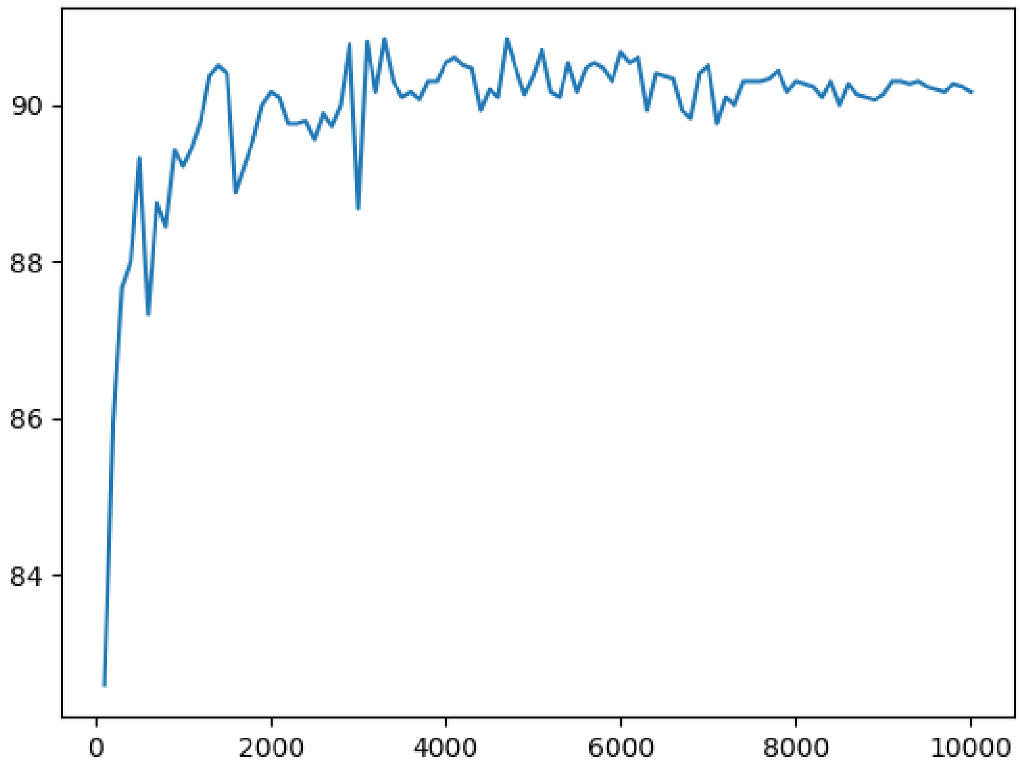
附录：程序源代码

```
# exp4_1.py
import numpy as np


def load_data(file):
    feature = []
    label = []
    with open(file) as f:
        for each_line in f.readlines():
            data = list(map(int, each_line.strip().split()))
            feature.append(data[0:-1])
```

```
        label.append([data[-1]])
    return np.mat(feature), np.mat(label)


def fit(feature, label):
    n, m = np.shape(feature)
    max_x = np.max(feature)
    max_y = np.max(label)
    p = [[[0 for i in range(max_y + 1)] for i in range(max_x + 1)] for i in
range(m)]
    y = [0 for i in range(max_y + 1)]
    for j in range(n):
        y[label[j, 0]] += 1
        for i in range(m):
            p[i][feature[j, i]][label[j, 0]] += 1
    for i in range(m):
        for j in range(max_x + 1):
            for k in range(max_y + 1):
                p[i][j][k] /= y[k]
    for i in range(max_y + 1):
        y[i] /= n
    return p, y


def classify(feature, prob, y):
    ans = 0
    max_p = 0
    m = np.shape(feature)[1]
    for i in range(len(y)):
        p = y[i]
        for j in range(m):
            p *= prob[j][feature[0, j]][i]
        if p > max_p:
            max_p = p
            ans = i
    return ans


if __name__ == "__main__":
    feature, label = load_data("exp4/data/training_data.txt")
    feature_test, label_test = load_data("exp4/data/test_data.txt")
    p, y = fit(feature, label)
    correct = 0
    for (f_test, l_test) in zip(feature_test, label_test):
        if classify(f_test, p, y) == l_test:
```

```
        correct += 1
    print("accuracy=%.2f%%" % (correct / len(feature_test) * 100))


# exp4_2.py
import numpy as np
import matplotlib.pyplot as plt
laplace = [3, 5, 4, 4, 3, 2, 3, 3]

def load_data(file):
    feature = []
    label = []
    with open(file) as f:
        for each_line in f.readlines():
            data = list(map(int, each_line.strip().split()))
            feature.append(data[0:-1])
            label.append([data[-1]])
    return np.mat(feature), np.mat(label)


def fit(feature, label):
    n, m = np.shape(feature)
    max_x = np.max(feature)
    max_y = np.max(label)
    p = [[[0 for i in range(max_y + 1)] for i in range(max_x + 1)] for i in 
range(m)]
    y = [0 for i in range(max_y + 1)]
    for j in range(n):
        y[label[j, 0]] += 1
        for i in range(m):
            p[i][feature[j, i]][label[j, 0]] += 1
    for i in range(m):
        for j in range(max_x + 1):
            for k in range(max_y + 1):
                p[i][j][k] /= (y[k] + laplace[i])
    for i in range(max_y + 1):
        y[i] /= n
    return p, y


def classify(feature, prob, y):
    label = 0
    max_p = 0
    m = np.shape(feature)[1]
    for i in range(5):
```

```
            p = y[i]
            for j in range(m):
                p *= prob[j][feature[0, j]][i]
            if p > max_p:
                max_p = p
                label = i
        return label

def random_test(feature, label, num):
    import random
    vis = dict()
    n = np.shape(feature)[0]
    feature_select = []
    label_select = []
    for _ in range(num):
        pos = random.randint(0, n - 1)
        while pos in vis:
            pos = random.randint(0, n - 1)
        vis[pos] = 1
        feature_select.append(feature[pos].tolist()[0])
        label_select.append(label[pos].tolist()[0])
    return np.mat(feature_select), np.mat(label_select)
```



```
if __name__ == "__main__":
    feature, label = load_data("exp4/data/training_data.txt")
    feature_test, label_test = load_data("exp4/data/test_data.txt")

    tot = [ i * 100 for i in range(1, 101)]
    rate = []
    for cnt in tot:
        f, l = random_test(feature, label, cnt)
        p, y = fit(f, l)
        correct = 0
        for (f_test, l_test) in zip(feature_test, label_test):
            if classify(f_test, p, y) == l_test:
                correct += 1
        rate.append(correct / len(feature_test) * 100)
    plt.plot(tot, rate)
    plt.show()
```