《数据仓库与数据挖掘》实验报告

姓名：	朱志儒	学号：	SA20225085	日期：	2020/12/21
上机题目：	朴素贝叶斯分类器				

操作环境：
OS：Window 10
CPU：AMD Ryzen 5 3600X 6-Core Processor 4.25GHz
GPU：GeForce RTX 2070 super

一、基础知识：

Naïve Bayes，即朴素贝叶斯分类器，有坚实的理论基础——贝叶斯定理。贝叶斯定理基于条件概率，条件概率 P(A|B)表示在事件 B 已经发生的前提下，事件 A 发生的概率，即$P(A|B)=\frac{P(AB)}{P(B)}$，贝叶斯定理通过 P(A|B)来求 P(B|A)：$P(B|A)=\frac{P(A|B)P(B)}{P(A)}$，其中 P(A)由全概率公式可分解为：$P(A)=\sum_{i=1}^{n}P(B_i)P(A|B_i)$。

假设给定训练数据集(X, Y)，其中每个样本 x 都包括 n 维特征，即$x=(x_1,x_2,\dots,x_n)$，类标记集合含有 k 中类别，即$y=(y_1,y_2,\dots,y_n)$。对于测试集样本 x，为判断其类别，从概率的角度来看，就是 x 属于 k 个类别中哪个概率最大，问题就变成找出$P(y_1|x),P(y_2|x),\dots,P(y_k|x)$中最大的项，即求出后验概率最大的输出：$\arg\max_{y_k}P(y_k|x)$。由贝叶斯定理可知：$P(y_k|x)=\frac{P(x|y_k)P(y_k)}{\sum_{k=1}^{n}P(x|y_k)P(y_k)}$。

分子中的$P(y_k)$是先验概率，可直接根据训练集数据计算得出，而条件概率$P(x|y_k)$有指数级数量的参数，假设第 j 维特征x_j可取值有S_j个，$j=1,2,3,\dots,n$，y 可取值有 K 个，那么参数个数为$K\prod_{j=1}^{n}S_j$。

朴素贝叶斯对条件概率作了条件独立性假设，即各个维度的特征$x_1,x_2,\dots,x_n$相互独立，在这个假设下，条件概率：$P(x|y_k)=P(x_1,x_2,\dots,x_n|y_k)=\prod_{i=1}^{n}P(x_i|y_k)$，如此，参数规模降为$\sum_{i=1}^{n}S_iK$，那么$P(y_k|x)=\frac{P(y_k)\prod_{i=1}^{n}P(x_i|y_k)}{\sum_k P(y_k)\prod_{i=1}^{n}P(x_i|y_k)}$，于是朴素贝叶斯分类器可表示为

$$y=f(x)=\arg\max_{y_k}P(y_k|x)=arg\max_{y_k}\frac{P(y_k)\prod_{i=1}^{n}P(x_i|y_k)}{\sum_k P(y_k)\prod_{i=1}^{n}P(x_i|y_k)}$$

在计算先验概率和条件概率时，需要做平滑处理：

$$P(y_k) = \frac{N_{y_k} + a}{N + ka}$$

$$P(x_i|y_k) = \frac{N_{y_k,x_i} + a}{N_{y_k} + na}$$

其中，N 为总样本个数，k 为总类别个数，N_{y_k}是类别为y_k的样本个数，a 为平滑值，n 为特征的维数，N_{y_k,x_i}是类别为y_k的样本中，第 i 维特征的值是x_i的样本个数。

在实际实现的过程中，考虑到$P(y_k|x)$中分母都为$P(x)$，所以在比较时可以忽略分母而只考虑分子。考虑到大量的概率浮点数乘法运算，为避免 floating-point underflow 问题，将乘法转化为取 log 再相加的运算：

$$y = f(x) = arg\max_{y_k} P(y_k|x) = arg\max_{y_k}(\log P(y_k) + \sum_{i=1}^{n} \log P(x_i|y_i))$$

二、实验过程：

根据原理对训练集数据进行统计，计算先验概率和条件概率，考虑到$P(y_k|x)$中分母都为$P(x)$，所以在比较时可以忽略分母而只考虑分子。考虑到大量的概率浮点数乘法运算，为避免 floating-point underflow 问题，将乘法转化为取 log 再相加的运算。

三、结果分析：

朴素贝叶斯分类器的效果如下：

```
准确率: 0.6196172248803827
```

附录

算法源代码（C/C++/JAVA 描述）：

```
1.  def read_data_set():
2.      '''''处理数据，提取特征'''
3.      trainData = pd.read_csv("train.csv")
4.      testData = pd.read_csv("test.csv")
5.      # 将训练集和测试集整合
6.      data = pd.concat([trainData, testData], axis=0).reset_i
    ndex(drop=True)
7.      # male: 0, female: 1
8.      data['Sex'].replace(['male', 'female'], [0, 1], inplace
    =True)
9.      # S: 0, C: 1, Q: 2
10.     data['Embarked'].replace(['S', 'C', 'Q'], [0, 1, 2], in
    place=True)
11.
12.     # print(data[data['Fare'].isnull()])
13.     # Pclass: 3, Embarked: 0, Sex: 0
```

```
    # 填补 Fare 为 NaN 的数据
    data['Fare'] = data['Fare'].fillna(
        np.mean(data[((data['Pclass'] == 3) & (data['Embarked'] == 0) & (data['Sex'] == 0))]['Fare']))
    # print(data[data['Fare'].isnull()])
    # Empty DataFrame

    # data['FareLimit'] = pd.qcut(data['Fare'], 4)
    # print(data.groupby(['FareLimit'])['Survived'].mean())

    # 使用 FareLimit 替代 Fare
    data['FareLimit'] = 0
    data.loc[data['Fare'] <= 8.662, 'FareLimit'] = 0
    data.loc[(data['Fare'] > 8.662) & (data['Fare'] <= 14.454), 'FareLimit'] = 1
    data.loc[(data['Fare'] > 14.454) & (data['Fare'] <= 53.1), 'FareLimit'] = 2
    data.loc[data['Fare'] > 53.1, 'FareLimit'] = 3

    # print(data[data['Embarked'].isnull()])
    # Pclass: 1, Sex: 1
    # 填补 Embarked 为 NaN 的数据
    data['Embarked'] = data['Embarked'].fillna(
        stats.mode(data[((data['Pclass'] == 1) & (data['Sex'] == 1))]['Embarked'])[0][0])
    # print(data[data['Embarked'].isnull()])
    # Empty DataFrame

    # 添加新特征：家人 Family
    data['Family'] = data['SibSp'] + data['Parch']

    # 填补 Age 为 NaN 的数据
    dataAgeNanIndex = data[data['Age'].isnull()].index
    for i in dataAgeNanIndex:
        # 取 Pclass、Family 相同的数据的平均值
        meanAge = data['Age'][
            (data['Pclass'] == data.iloc[i]['Pclass']) & (data['Family'] == data.iloc[i]['Family'])].mean()
        data['Age'].iloc[i] = meanAge

    # data['AgeLimit'] = pd.cut(data['Age'], 5)
    # print(data.groupby(['AgeLimit'])['Survived'].mean())

    # 使用 AgeLimit 替代 Age
```

```
51.     data['AgeLimit'] = 0
52.     data.loc[data['Age'] <= 16, 'AgeLimit'] = 0
53.     data.loc[(data['Age'] > 16) & (data['Age'] <= 32), 'Age
    Limit'] = 1
54.     data.loc[(data['Age'] > 32) & (data['Age'] <= 48), 'Age
    Limit'] = 2
55.     data.loc[(data['Age'] > 48) & (data['Age'] <= 60), 'Age
    Limit'] = 3
56.     data.loc[data['Age'] > 60, 'AgeLimit'] = 4
57.
58.     # 删除无用列
59.     data.drop(labels=["Age", "Fare", "Ticket", "Cabin", "Na
    me", "PassengerId", 'Family'], axis=1, inplace=True)
60.
61.     data = data[['Pclass', 'Sex', 'SibSp', 'Parch', 'Embark
    ed', 'FareLimit', 'AgeLimit', 'Survived']]
62.
63.     trainY = data[:len(trainData)]['Survived'].values.tolis
    t()
64.     trainX = data[:len(trainData)].drop(labels='Survived',
    axis=1)
65.     testY = data[len(trainData):]['Survived'].values.tolist
    ()
66.     testX = data[len(trainData):].drop(labels='Survived', a
    xis=1)
67.
68.     return trainX, trainY, testX, testY
69.
70. class NaiveBayes:
71.     def __init__(self, x, y):
72.         self.xlabel = ['Pclass', 'Sex', 'SibSp', 'Parch', '
    Embarked', 'FareLimit', 'AgeLimit']
73.         ycounts = [0, 0]
74.         self.pos_dict = {
75.             'Pclass': {},
76.             'Sex': {},
77.             'SibSp': {},
78.             'Parch': {},
79.             'Embarked': {},
80.             'FareLimit': {},
81.             'AgeLimit': {}
82.         }
83.         self.neg_dict = {
84.             'Pclass': {},
```

```
            'Sex': {},
            'SibSp': {},
            'Parch': {},
            'Embarked': {},
            'FareLimit': {},
            'AgeLimit': {}
        }
        for i in range(len(y)):
            ycounts[y[i]] += 1
            for index in self.xlabel:
                item = x.iloc[i][index]
                if y[i]:
                    self.pos_dict[index][item] = self.pos_dict[index].get(item, 0) + 1
                else:
                    self.neg_dict[index][item] = self.neg_dict[index].get(item, 0) + 1
        for index in self.xlabel:
            for key in list(self.pos_dict[index].keys()):

                self.pos_dict[index][key] = (self.pos_dict[index].get(key, 0) + 1) / (ycounts[1] + len(self.pos_dict[index].keys()))
            for key in list(self.neg_dict[index].keys()):

                self.neg_dict[index][index] = (self.pos_dict[index].get(key, 0) + 1) / (ycounts[0] + len(self.neg_dict[index].keys()))
        self.pos_prob = ycounts[1] / len(y)
        self.neg_prob = ycounts[0] / len(y)

    def classify(self, testX):
        result = []
        for i in range(len(testX)):
            positive = math.log(self.pos_prob)
            negative = math.log(self.neg_prob)
            for index in self.xlabel:
                item = testX.iloc[i][index]
                positive += math.log(self.pos_dict[index].get(item, 0.000000001))
                negative += math.log(self.neg_dict[index].get(item, 0.000000001))
            result.append(1 if positive > negative else 0)
```

```
        return result

def validation(testY, result):
    count = 0
    for i in range(len(result)):
        if int(result[i]) == int(testY[i]):
            count += 1
    print('准确率:', count / len(result))

if __name__ == "__main__":
    trainX, trainY, testX, testY = read_data_set()
    navieBayes = NaiveBayes(trainX, trainY)
    result = navieBayes.classify(testX)
    validation(testY, result)
```