## 贝叶斯决策论

以多分类任务为例，假设有N个可能的类别，即 $Y=\{c_1,c_2,...,c_N\}$，$\lambda_{ij}$ 是将一个真实的标记为 $c_j$ 的样本误分为 $c_j$ 所产生的损失。基于后验概率 $P(c_i \mid x)$ 可获得样本x分类为 $c_i$ 所产生的损失。基于后验概率 $P(c_i \mid x)$ 可获得样本x分类为 $c_i$ 所产生的期望损失，也就是在样本x上的条件风险为：

$$R(c_i \mid x)=\sum_{j=1}^{N}\lambda_{ij}P(c_j \mid x)$$

我们做决策肯定是希望总体风险最小化，所以寻求一个判定准则 $h:X\mapsto Y$ 使得总体风险最小：

$$R(h)=E_x(R(h(x) \mid x))$$

如果每个样本x都能最小化条件风险，那么总体的风险也一定会最小化。于是贝叶斯判定准则就出来了：为最小化总体风险，只需在每个样本上选择使得条件风险最小的类别。
根据上文的贝叶斯判定准则，其实就已经足够理解贝叶斯分类器的工作原理了。

## 朴素贝叶斯

朴素贝叶斯的英文名叫Naïve Bayes，说它天真是因为，它的假设确实太天真了，但是不得不说的是，天真的假设却能够带来很好的效果。
这个Naïve的假设全名为：attribute conditional independence assumption，直译就是属性条件独立性假设：对已知类别，假设所有属性相互独立。
先写出用于分类的贝叶斯公式：

$$P(c \mid x)=\frac{P(c)P(x \mid c)}{P(x)}$$

对于这个公式，P(c|x)代表在样本为x的条件下，类别为c的概率，根据上文所述的判定准则，当然倾向于选择概率最大的类别，这个概率为后验概率。先验概率为P(c)与条件概率P(x|c)需要通过学习得到。
利用天真的假设，我们可以推导得到：

$$P(c \mid x)=\frac{P(c)P(x \mid c)}{P(x)}=\frac{P(c)}{P(x)}\prod P(x_i \mid c)$$

而P(x)对所有的类别都是常数，因此只需要计算上半部分即可。根据判定准则可以得到朴素贝叶斯表达式为：

$$h_{nb}(x)=argmax_{c\in Y}P(c)\prod P(x_i \mid c)$$

对于P(c)，这个很容易通过训练集学习得到，令 $D_c$ 为训练集中c类样本的集合，那么先验概率P(c)可以估计为：

$$P(c)=\frac{\mid D_c \mid}{\mid D \mid}$$

对于 $P(x_i \mid c)$，如果属性是离散的，那么可以通过某一类别某一特征的某个取值出现的频率进行估计：

$$P(x_i \mid c)=\frac{D_{c,x_i}}{D_c}$$

对于连续型变量，则需要对 $P(x_i \mid c)$ 进行估计了，一般情况下都是假设服从正态分布的，先估计 $\mu,\theta$，再构成密度函数。

## 算法实现

这里主要实现用于文本分类的朴素贝叶斯分类器，数据来源是2018年服务外包大赛阿里命题的发票分类。
首先介绍一下多项式模型，首先对句子分词，然后重复的词不去合并，直接应用贝叶斯进行分类。
数据是已经分过词的，因此可以直接拿来用，话不多说，直接上代码，代码里有详细的注释。

```
# -*- coding: utf-8 -*-
"""
Created on Fri Jan 11 17:40:31 2019
```

```
@author: zmddzf
"""
import pandas as pd

def readCSV(path):
    """
    用于读取数据
    :param path: 数据文件的路径
    :returns:
        names: 商品名称列表
        labels: 标签列表
    """
    data = pd.read_csv(path)
    names = data["商品名称"].tolist()
    names = [name.split(' ') for name in names]
    labels = data['商品编码'].tolist()
    return names, labels

def computePriori(labels):
    """
    计算先验概率P(c)
    :param labels: 标签列表
    :return:
        priorDict: 字典形式的每一类文档出现的频率
    """
    labelSet = set(labels)
    priorDict = dict()
    for label in labelSet:
        priorDict[label] = labels.count(label) / len(labels)
    return priorDict

def computeConditional(names, labels):
    """
    计算条件概率，这里需要注意，P(w|c)，此处分母应该是c类文档中的词的总数，而不是c类文档数
    :param names: 商品名称列表
    :param labels: 商品标签列表
    :return:
        conditionalDict: 条件概率字典
    """
    conditionalDict = dict()
    cwDict = dict((lab, 0) for lab in set(labels)) #初始化各类文档词的总数字典

    #统计每个词再每一类中出现的频数
    for name, label in zip(names, labels):
        for word in name:
            if word not in conditionalDict.keys():
                conditionalDict[word] = dict([(i, j) for i, j in zip(set(labels), [1 for i in range(len(set(labels)))])])
            else:
                conditionalDict[word][label] += 1

    #统计每一类中的总词数
    for lab in cwDict:
        for i in conditionalDict.values():
            cwDict[lab] += i[lab]

    #计算条件概率
    for item in conditionalDict:
        for priori in set(labels):
            conditionalDict[item][priori] = conditionalDict[item][priori] / cwDict[priori]
    return conditionalDict

def predict(name, priorDict, conditionalDict):
    """
    对商品名称进行分类
    :param name: 一个商品名称的分词列表
    :param priorDict: 先验概率字典
    :param conditionalDict: 条件概率字典
    :return:
        label: 分类结果
    """
    probDict = dict()
    for label in priorDict:
        probDict[label] = priorDict[label] #初始化各类概率字典
        #计算后验概率
        for word in name:
            if word in conditionalDict:
                probDict[label] *= conditionalDict[word][label] #利用贝叶斯公式计算后验概率
            else:
                pass #如果词汇超出了词典范围，则不做处理
    label = max(probDict,key=probDict.get) #求最大概率值的标签
    return label
```

```
if __name__ == "__main__":
    names, labels = readCSV('data.csv')
    priorDict = computePriori(labels)
    conditionalDict = computeConditional(names, labels)

    #对训练集进行分类，观察分类正确率
    count = 0
    for name, label in zip(names, labels):
        predLabel = predict(name, priorDict, conditionalDict)
        if predLabel == label:
            count += 1

    print("Accuracy:", count / len(labels))
```

这么一个天真的模型，都能达到78.72%的accuracy，说明朴素贝叶斯确实一种简单实用的方法。由于代码是图片形式，所以没法直接复制粘贴，对代码有需求的小伙伴可以移步到我的GitHub上下载我的源码。