机器学习算法总结--朴素贝叶斯

这次需要总结的是朴素贝叶斯算法，参考文章：

- 《统计学习方法》
- 机器学习常见算法个人总结（面试用）
- 朴素贝叶斯理论推导与三种常见模型
- 朴素贝叶斯的三个常用模型：高斯、多项式、伯努利

简介

> 朴素贝叶斯是基于贝叶斯定理与特征条件独立假设的分类方法。

贝叶斯定理是基于条件概率来计算的，条件概率是在已知事件B发生的前提下，求解事件A发生的概率，即
$P(A|B)=\frac{P(AB)}{P(B)}$
，而贝叶斯定理则可以通过
$P(A|B)$
来求解
$P(B|A)$
：

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

其中分母
$P(A)$
可以根据全概率公式分解为：
$P(A)=\sum_{i=1}^{n}P(B_i)P(A|B_i)$

而**特征条件独立假设**是指假设各个维度的特征
$x_1,x_2,...,x_n$
互相独立，则条件概率可以转化为：

$$P(x|y_k)=P(x_1,x_2,...,x_n|y_k)=\prod_{i=1}^{n}P(x_i|y_k)$$

朴素贝叶斯分类器可表示为：

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

而由于对上述公式中分母的值都是一样的，所以可以忽略分母部分，即可以表示为：

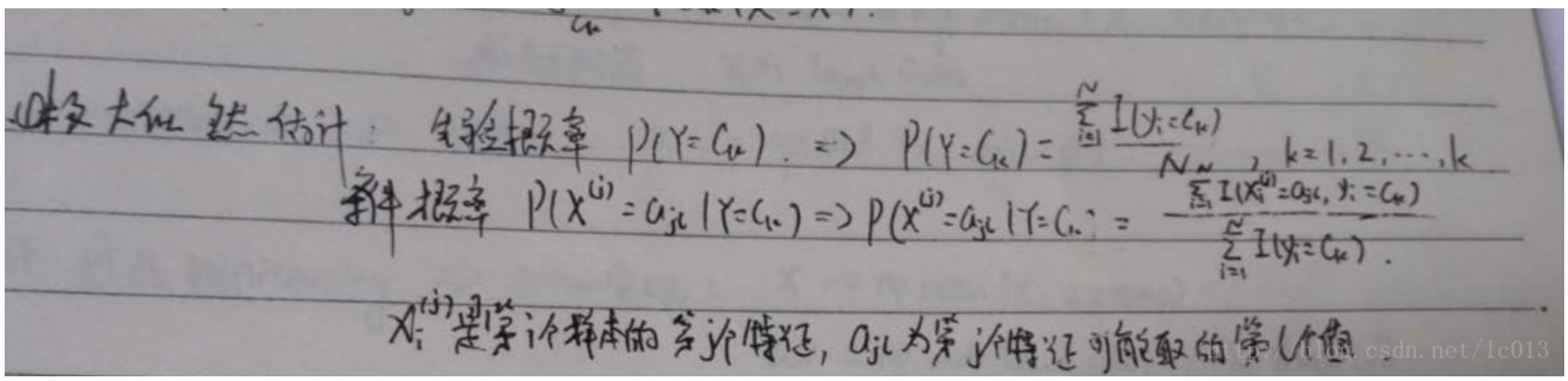
$$f(x) = argmax P(y_k)\prod_{i=1}^{n} P(x_i | y_k)$$

这里
$P(y_k)$
是先验概率，而
$P(y_k | x)$
则是后验概率，朴素贝叶斯的目标就是最大化后验概率，这等价于期望风险最小化。

参数估计

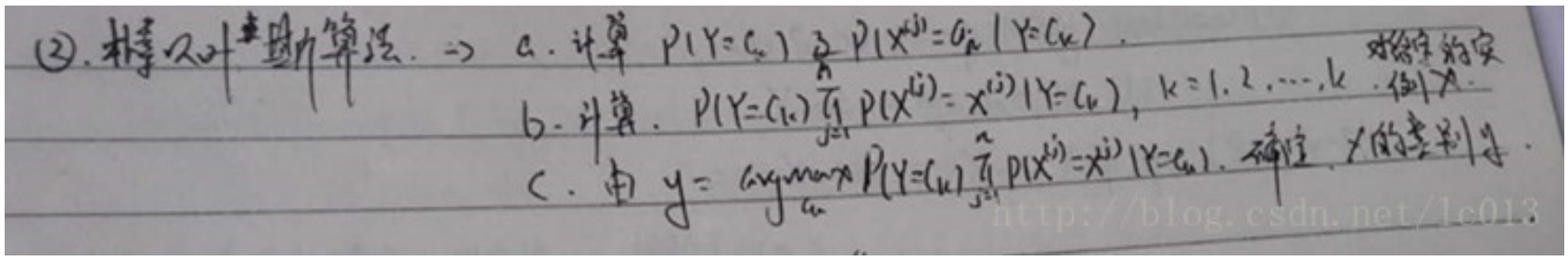
极大似然估计

朴素贝叶斯的学习意味着估计
$P(y_k)$
和
$P(x_i | y_k)$
,可以通过**极大似然估计**来估计相应的概率。



如上图所示，分别是
$P(y_k)$
和
$P(x_i | y_k)$
的极大似然估计。

当求解完上述两个概率，就可以对测试样本使用朴素贝叶斯分类算法来预测其所属于的类别，简单总结的算法流程如下所示：



贝叶斯估计/多项式模型

用**极大似然估计可能会出现所要估计的概率值为0**的情况，这会影响到后验概率的计算，使分类产生偏差。解决这个问题的办法是使用**贝叶斯估计**，也被称为多项式模型。

当**特征是离散的时候，使用多项式模型**。多项式模型在计算先验概率
$P(y_k)$
和条件概率
$P(x_i \mid y_k)$
时，会做一些**平滑处理**，具体公式为：

$$P(y_k) = \frac{N_{y_k} + \alpha}{N + k\alpha}$$

> N
> 是总的样本个数，
> k
> 是总的类别个数，
> N_{y_k}
> 是类别为
> y_k
> 的样本个数，
> α
> 是平滑值。

$$P(x_i \mid y_k) = \frac{N_{y_k, x_i} + \alpha}{N_{y_k} + n\alpha}$$

> N_{y_k}
> 是类别为
> y_k
> 的样本个数，
> n

> 是特征的维数，
> 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，
> α
> 是平滑值。

当
$\alpha = 1$
时，称作**Laplace平滑**，当
$0 < \alpha < 1$
时，称作**Lidstone**平滑，α=0时不做平滑。

如果不做平滑，当某一维特征的值
x_i
没在训练样本中出现过时，会导致
$P(x_i \mid y_k) = 0$
，从而导致后验概率为0。加上平滑就可以克服这个问题。

高斯模型

当特征是连续变量的时候，运用多项式模型会导致很多
$P(x_i \mid y_k) = 0$
（不做平滑的情况下），即使做平滑，所得到的条件概率也难以描述真实情况，所以处理连续变量，应该采用高斯模型。

高斯模型是假设每一维特征都服从高斯分布（正态分布）：

$$P(x_i \mid y_k) = \frac{1}{\sqrt{2\pi\sigma_{y_k}^2}} exp(-\frac{(x_i - \mu_{y_k})^2}{2\sigma_{y_k}^2})$$

$\mu_{y_k, i}$
表示类别为
y_k
的样本中，第
i
维特征的均值；

$\sigma^2_{y_k,i}$
表示类别为
y_k
的样本中，第
i
维特征的方差。

伯努利模型

与多项式模型一样，伯努利模型适用于**离散特征**的情况，所不同的是，**伯努利模型中每个特征的取值只能是1和0**(以文本分类为例，某个单词在文档中出现过，则其特征值为1，否则为0).

伯努利模型中，条件概率
$P(x_i \mid y_k)$
的计算方式是：

当特征值
x_i
为1时，
$P(x_i \mid y_k) = P(x_i = 1 \mid y_k)$
；

当特征值
x_i
为0时，
$P(x_i \mid y_k) = 1 - P(x_i = 1 \mid y_k)$
；

工作流程

1. 准备阶段
   确定特征属性，并对每个特征属性进行适当划分，然后由人工对一部分待分类项进行分类，形成训练样本。
2. 训练阶段
   计算每个类别在训练样本中的出现频率及每个特征属性划分对每个类别的条件概率估计
3. 应用阶段
   使用分类器进行分类，输入是分类器和待分类样本，输出是样本属于

的分类类别

属性特征

1. 特征为离散值时直接统计即可（表示统计概率）
2. 特征为连续值的时候假定特征符合高斯分布，则有

$$P(x_i \mid y_k) = \frac{1}{\sqrt{2\pi\sigma_{y_k}^2}} exp(-\frac{(x_i - \mu_{y_k})^2}{2\sigma_{y_k}^2})$$

优缺点

优点

1. 对小规模的数据表现很好，适合多分类任务，适合增量式训练。

缺点

1. 对输入数据的表达形式很敏感（离散、连续，值极大极小之类的）。

代码实现

下面是使用sklearn的代码例子，分别实现上述三种模型,例子来自朴素贝叶斯的三个常用模型：高斯、多项式、伯努利。
下面是高斯模型的实现

```
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> iris.feature_names  # 四个特征的名字
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width
>>> iris.data
array([[ 5.1,  3.5,  1.4,  0.2],
       [ 4.9,  3. ,  1.4,  0.2],
       [ 4.7,  3.2,  1.3,  0.2],
       [ 4.6,  3.1,  1.5,  0.2],
       [ 5. ,  3.6,  1.4,  0.2],
       [ 5.4,  3.9,  1.7,  0.4],
       [ 4.6,  3.4,  1.4,  0.3],
       [ 5. ,  3.4,  1.5,  0.2],
       ......
```

```
       [ 6.5,  3. ,  5.2,  2. ],
       [ 6.2,  3.4,  5.4,  2.3],
       [ 5.9,  3. ,  5.1,  1.8]]) #类型是numpy.array
>>> iris.data.size
600  #共600/4=150个样本
>>> iris.target_names
array(['setosa', 'versicolor', 'virginica'],
      dtype='|S10')
>>> iris.target
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,....., 0, 1, 1, 1,
>>> iris.target.size
150
>>> from sklearn.naive_bayes import GaussianNB
>>> clf = GaussianNB()
>>> clf.fit(iris.data, iris.target)
>>> clf.predict(iris.data[0])
array([0])   # 预测正确
>>> clf.predict(iris.data[149])
array([2])   # 预测正确
>>> data = numpy.array([6,4,6,2])
>>> clf.predict(data)
array([2])  # 预测结果很合理
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26

- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36

多项式模型如下：

```
>>> import numpy as np
>>> X = np.random.randint(5, size=(6, 100))
>>> y = np.array([1, 2, 3, 4, 5, 6])
>>> from sklearn.naive_bayes import MultinomialNB
>>> clf = MultinomialNB()
>>> clf.fit(X, y)
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
>>> print(clf.predict(X[2]))
[3]
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

值得注意的是，多项式模型在训练一个数据集结束后可以继续训练其他数据集而无需将两个数据集放在一起进行训练。在sklearn中，MultinomialNB()类的partial_fit()方法可以进行这种训练。这种方式特别适合于训练集大到内存无法一次性放入的情况。

在第一次调用`partial_fit()`时需要给出所有的分类标号。

```
>>> import numpy
>>> from sklearn.naive_bayes import MultinomialNB
>>> clf = MultinomialNB()
>>> clf.partial_fit(numpy.array([1,1]), numpy.array(['aa']), ['aa','bb'])
```

```
GaussianNB()
>>> clf.partial_fit(numpy.array([6,1]), numpy.array(['bb']))
GaussianNB()
>>> clf.predict(numpy.array([9,1]))
array(['bb'],
      dtype='|S2')
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

伯努利模型如下：

```
>>> import numpy as np
>>> X = np.random.randint(2, size=(6, 100))
>>> Y = np.array([1, 2, 3, 4, 4, 5])
>>> from sklearn.naive_bayes import BernoulliNB
>>> clf = BernoulliNB()
>>> clf.fit(X, Y)
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
>>> print(clf.predict(X[2]))
[3]
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

朴素贝叶斯的总结就到这里为止。