简介

贝叶斯分类器是基于贝叶斯理论的分类器，在NLP（自然语言处理）领域有着广泛的应用，如垃圾邮件检测，个人邮件排序，文本分类，色情内容检等等。由于贝叶斯分类器是基于贝叶斯理论的，因此使用该分类器时有一个基本假设，即：数据的各特征之间是条件独立的。

假设数据集 $D = \{d_1, d_2, \ldots, d_n\}$ 的特征集合为 $X = \{x_1, x_2, \ldots, x_m\}$ ，类别集合为 $C = \{c_1, c_2, c_k\}$. 即对任意一条数据 d_i ，均有大小为 m 一维特征向量，数据 d_i 的类别为 $c_j\ (\leq j \leq k)$. 那么 $P(x_i|C)$ 相互之间是条件独立的，即 $P(x_1, x_2, \ldots, x_k|C) = \prod_{i=1}^{k} P(x_i|C)$.

贝叶斯定理（Naïve Bayes Theorem）

贝叶斯定理指：对于事件A和B，它们之间的概率关系满足：

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \qquad (1)$$

贝叶斯定理说明，通常事件A在事件B（发生）的条件下的概率，与时间B在事件A的条件下的概率是不一样的，但两者之间有确定的关系，这个关系可用贝叶斯定理来描述。

通常在数据分类的应用中，我们会替换上述公式的一些符号以方便描述。我们假设 X 是数据的特征，C 是数据的类别，则上式可以写成：

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)} \quad (2)$$

其中$P(C|X)$ 的含义是：对于给定的一个文本，已知它的特征是X ，那么这个文本属于类别C的概率是多少。这个值就是我们最终需要的值。

$P(C|X)$ 是贝叶斯分类器要计算出的结果，我们就是通过这个概率来确定这个文本属于哪个类别。这个概率称为**后验概率（posterior probability**即我们只有在知道文本的特征X **之后**，才会知道这个文本属于哪个类别。

$P(C)$ 是**先验概率（prior probability）**，表示在观察到文本的特征X **之前**，我们就已经知道了类别C概率，即这个概率跟X 完全无关。$P(X)$ 同理

$P(X|C)$ 称为**相似度（likelihood）**。这个概率表示的意思是我们已经确定了一个类别C ，那么在C中的文本出现特征值为X 的概率是多少。

在实际的应用中，$P(X|C)$ ，$P(C)$ 和$P(X)$ 都可以直接或间接获得，或者通过估计得到。

Multinomial Naïve Bayes

贝叶斯分类器有三种，分别是Multinomial Naive Bayes， Binarized Multinomial Naive Bayes以及Bernoulli Naive Bayes. 本文讲述第一种贝叶斯
类器，该分类器主要用于文本的主题分类。Multinomial Naive Bayes中会考虑单词出现的次数，即词频（term frequency）；而第二种——Binari
Multinomial Naive Bayes——不考虑词频，只考虑这个单词有没有出现，主要用于文本情绪分析。例如，一段文本提到bad这个单词，使用第二种
器不会考虑bad出现了几次，它只关注bad这个**有没有出现**。

Multinomial Naïve Bayes分类器算法

假设有文本数据集$D = \{d_1, d_2, \ldots, d_n\}$，其中$d_i (1 \le i \le n)$表示第$i$个文本，因此这个文本数据集一共有$n$个文本。

该文本数据集D有一个特征集$X = \{x_1, x_2, \ldots, x_m\}$，表示对于任意一个文本$d_i$，都有一个特征集$X_i$，文本$d_i$的特征大小为$m$，即$d_i$有$m$个特征
这个特征可以是单词（这是最简单的情况），也可以是N-gram或自己设计的特征。本文使用单词作为特征。若使用单词，那么特征集X就是文本数
D中的所有单词的集合，x_j就是**第j个单词**，m就是所有单词的数量，可记为$m = |X|$。

例如有两个文本d_1和d_2
d_1：*This article is about Bayes.*
d_2：*Thomas Bayes provided Bayes equation.*
在这个例子中，$D = \{d_1, d_2\}$，$X = \{this, article, is, about, bayes, thomas, provided, equation\}$，$|X| = 8$

根据**公式(2)**，要得到一段文本属于某一类的概率，需要先计算$P(C)$， $P(X)$ 和$P(X|C)$。

应用贝叶斯分类器之前，我们通常会有training data，以上三个概率就从training data中获得。

如何计算$P(C)$

设有类别集合$C = \{c_1, c_2, \ldots, c_k\}$，$P(C)$表示取得某一个类别的概率，比如不同文本可能属于不同的话题类别，如sport, politics, science等。
我们假设文本的话题类别归属随机，假设有k个类别，那么取到任一个类别的概率（用频率来近似概率）就是$\frac{1}{k}$. 因此$P(C)$的值是固定的，
$P(C) = \frac{1}{k}$，k是类别的数量，这个是事先定义好的。

这是最简单的情况，在一些应用上，$P(C)$的值是随不同类别的变化而变化的，或者c_i服从某种分布，视具体情况而定。有时可以在training data中
各类的出现的频率作为$P(C)$的估计值。

如何计算$P(X)$

在本文例子中，$X = \{x_1, x_2, \ldots, x_m\}$ 是所有文本的单词集合，m表示单词数量。通常情况下，$P(X)$ 的值是固定，即$P(X) = \frac{1}{m}$ （用频率来近概率）。该值的确定是基于词袋模型（bag-of-words），即一篇文章是由若干单词组成，每个单词均是从词库中随机等概率抽取而来。像是将所有单放到一个袋子里（bag），写文章就从这个袋子里随机抽取单词。

但是该模型的缺陷显而易见，通常写作用的单词并不是等概率抽取的。然而实际应用中这样简化问题并取得令人满意的结果。

如何计算$P(X|C)$

$P(X|C)$ 表示对于指定的类别C，在这个类别中的文本出现的特征值等于X 的概率是多少。注意到X 和C都是集合，要确定一段文本属于哪一类，需计算所有类别的$P(X|C)$ ，即对第i类，计算$P(X|c_i)$ 。如此最后才能比较文本属于哪个类别的概率最大。

而X 是单词的集合，一段文本是由若干单词组成，整个单词串连在一起才能作为这段文本的特征值向量。因此对第i类，我们需要对每个文本计算$P(x_1, x_2, \ldots, x_m|c_i)$ 。注意到前面提到贝叶斯分类器的假设是各特征变量之间条件独立，因此有

$$P(x_1, x_2, \ldots, x_m|c_i) = \prod_{j=1}^{m} P(x_j|c_i) \qquad (3)$$

计算类别归属（$P(C|X)$ ）

如此，判断文本d属于第i类的概率可以写成：

$$P(D_d|c_i) = \frac{P(c_i)\prod_{j=1}^{m} P(x_j|c_i)}{P(X)} \tag{4}$$

因为$P(X)$ 是常数项（简单情况下$P(c_i)$也可以是常数项），因此上式可以简化为：

$$P(D_d|c_i) \propto P(c_i)\prod_{j=1}^{m} P(x_j|c_i) \tag{5}$$

因为上式去掉了分母，得到的结果已不是概率，因此将等号替换为$\propto$（approximately proportional to）。上式含有积分项，不方便计算且在计算机可能有溢出的危险，因此对上式右项取对数，简化为：

$$P(D_d|c_i) \propto \log P(c_i) + \sum_{j=1}^{m} \log P(x_j|c_i) \tag{6}$$

此时公式已化至最简，**但加号右边的单项$P(x|c)$如何计算？**

$P(x|c)$的计算以及平滑因子的引入

可以将单词x_j在c_i类中出现的次数转化为频率，用该频率来估计$P(x_j|c_i)$。下面是个例子。
假设有以下文本：

Text	**Class**	**Doc**	
Chinese Beijing Chinese	ZH	d_1	**Training**

Text	Class	Doc	
Chinese Chinese Shanghai	ZH	d_2	
Chinese Macao	ZH	d_3	
Tokyo Japan Chinese	JP	d_4	
Chinese Chinese Chinese Tokyo Japan	?	d_5	**Test**

上述例子的training data中，$X = \{Chinese, Beijing, Shanghai, Macao, Tokyo, Japan\}$，$C = \{ZH, JP\}$ 。
则*Chinese*在*ZH*类中出现的次数$count(Chinese, ZH) = 5$，在*JP*类中出现的次数$count(Chinese, JP) = 1$。类似地，有：

$$count(Chinese, ZH) = 5$$

$$count(Beijing, ZH) = 1$$

$$count(Shanghai, ZH) = 1$$

$$count(Macao, ZH) = 1$$

$$count(Tokyo, JP) = 1$$

$$\text{count}(\text{Japan}, \text{JP}) = 1$$

$$\text{count}(\text{Chinese}, \text{JP}) = 1$$

将上述单词出现的次数转换为频率，即除以该类别的单词数量。类别*ZH*的单词数量为8（重复的单词也算），*JP*类别的单词数量为3. 实际上我们要算是Test数据分别属于*ZH*和*JP*的概率，因此只考虑Test出现的单词即可（忽略Beijing, Shanghai和Macao）。计算得到：

$$\text{P}(\text{Chinese}|\text{ZH}) \simeq \text{freq}(\text{Chinese}, \text{ZH}) = \frac{5}{8}$$

$$\text{P}(\text{Tokyo}|\text{ZH}) \simeq \text{freq}(\text{Tokyo}, \text{ZH}) = 0$$

$$\text{P}(\text{Japan}|\text{ZH}) \simeq \text{freq}(\text{Japan}, \text{ZH}) = \frac{1}{8}$$

$$\text{P}(\text{Chinese}|\text{JP}) \simeq \text{freq}(\text{Chinese}, \text{JP}) = \frac{1}{3}$$

$$\text{P}(\text{Tokyo}|\text{JP}) \simeq \text{freq}(\text{Tokyo}, \text{JP}) = \frac{1}{3}$$

$$P(Japan|JP) \simeq freq(Japan, JP) = \frac{1}{3}$$

以上是计算单项$P(x_j|c_i)$的简单过程。

一般地，设T_{cx}是某个文档d中的单词x在c类中出现的次数。设$\sum_{x^+ \in X} T_{cx^+}$是所有文档中（即D）单词$x^+$出现在c类中的次数，X表示D的单词集合。则上述过程的计算可用下式表示：

$$P(x|c) = \frac{T_{cx}}{\sum_{x^+ \in X} T_{cx^+}} \tag{7}$$

但注意到上式的分子分母都可能为0，如$P(Tokyo|ZH)$的值为0，显然0的对数无意义。因此引入平滑因子α，上式改写为：

$$P(x|c) = \frac{T_{cx} + \alpha}{\sum_{x^+ \in X}(T_{cx^+} + \alpha)} \tag{8}$$

令$\alpha = 1$，则有：

$$P(x|c) = \frac{T_{cx} + 1}{\sum_{x^+ \in X}(T_{cx^+} + 1)} = \frac{T_{cx} + 1}{\sum_{x^+ \in X}(T_{cx^+}) + |X|} \tag{9}$$

上式即为计算$P(x|c)$的最终公式。$\alpha = 1$称为**Laplace平滑**。

如此，将**公式(9)**带入**公式(6)**，即可进行文本分类。当然，需要先有training data.

使用Python做文本情绪分类的实例

现在有1000多条已经标注好类别的tweets，根据每条tweet中包含的hashtag确定这条tweet属于哪一类。例如一条tweet中包含“*#happy*”，那么这条tweet就会被分类到happy的类别。

在这个例子中，所有的tweet组成了文本集合$D = \{d_1, d_2, \ldots, d_n\}$（$n = 20000$）。

将这些tweet分成两部分，一部分用作training data，有700条tweet，剩下的用于test data. 先看看这些tweet长什么样。

	A	B	
1	ID	text	emotion
2	1	Ben almost locked Steven out of their room. #angry #drunk #vindictive #podcast https://t.co/oqCr3DoxCa	anger_2
3	2	Traitor beware. #angry	anger_2
4	3	Disappointed in #unitedairlines phone service, dumped me into a blank phone line after a 1:22min call! #angry !	anger_2
5	4	I feel always being #angry could be good and bad but it'd be #helpful	anger_2
6	5	#technology #Angry gorilla simulator: You need to do to adapt to the environment. The need to tear the city. Mat	anger_2
7	6	Bloody pool boilers are knackered so no swimming #Angry	anger_2
8	7	How much do you #like this #angry face? BMW 525 D https://t.co/m6hKgVHllj https://t.co/u4XDyLoL4a	anger_2
9	8	"Immigrant" and "refugee" are not synonyms for ffffffssssss sakeeeeee. #angry	anger_2
10	9	#Triumph yes, but #NoWayToElectAPresident: #USA needs 2 get #Past #angry #old #men https://t.co/X0gzWcD	anger_2
11	10	my Twitter accound was BLOCKED AGAIN! Can someone explain WHY?? #angry #frustrated	anger_2
12	11	RT @NewsBreaksLive: #Angry and frustrated upstate #NewYork swings behind #Trump https://t.co/g8mYS7h0C	anger_2
13	12	#colorcap #Angry bull up: You will dominate your bull and you will direct it whatever you want it to do. When you	anger_2
14	13	.@YoHi64 @sudu59 I sure hope for our sake he keeps up the fight. His campaign is hinting he won't though. htt	anger_2
15	14	RT @Drudge_Report_: #Angry and frustrated upstate #NY swings behind #Trump... https://t.co/ebaV4QtoPb	anger_2
16	15	WTF ! @YouTube why the hell are you messing up ? #angry #fuck #hashtag	anger_2
17	16	@MikelelelArteta no I have not!!!!!! #angry	anger_2
18	17	Help 4 #Caregivers who are #Angry https://t.co/NfJSxmFj0T #BabyBoomers #Caregiving #Aging #Seniors https	anger_2
19	18	RT @becubed: Really, guys? "The problem isn't to fix single-serve coffee garbage, it's to stop people feeling BA	anger_2
20	19	It's tough falling in love with a writer. They would use such words when in a fight, you won't even know you've be	anger_2
21	20	I'm feeling very #harassed and #angry that my """"friend's""""" responses have been called a jock and "slightly e	anger_2
22	21	RT @shakin_myhead: #Hilarious Notes From #Angry #Neighbors https://t.co/XjDBfTUhsW	anger_2

数据预处理

在对tweet分类之前，先对tweet进行了预处理。首先对每个单词做lemmalization，即将动词和名词复数转为原形，形容词加ly后缀变成的副词都转容词原形，这个步骤通过Python的库可以实现。因为tweet的用语十分不规范，存在很多网络语言，因此对每条tweet去掉了在英文字典中不存在的!（其实这个步骤可以省略，不应该剔除特殊词汇。但是为了方便，我还是去掉了）。

Python的sklearn开发包

sklearn包含很多机器学习的库，multinomial Bayes位于sklearn.naive_bayes.MultinomialNB中。先看看这个**类（不是函数）**的定义：

```
sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)
```

参数alpha就是上文提到的平滑因子α。fit_prior=True指是否根据training data学习$P(C)$的值，若赋值False，则$P(C)$的值是固定的，即上文提到$P(C) = \frac{1}{k}$，k是类别数。class_prior指是否指定$P(C)$的值。

使用方法如下。具体的使用说明可以参考这个链接。

定义一个MultinomialNB的对象

```
clf = MultinomialNB(alpha, fit_prior, class_prior)
```

输入训练样本数据

```
clf.fit(X_train_tf, Y_train)
```

预测

```
predicted = clf.predict(X_pre_tf)
```

以上的难点其实在于如何获取X_train_tf. 最后处理的结果accuracy是0.7197. 这部分的代码和数据可以参考这里。

这个例子的缺陷

这个例子中使用的tweet无论是training data还是test data都是已经标注好的数据，而标注的依据便是tweet中包含的hashtag，因此每一条tweet其都包含了某一个emotion类别的关键字，这样可能会无形中提高分类的accuracy. 同时对文本也没有进行去除stopword等预处理，也没有计算TF-ID