文本分类技术基础及实践

2019年5月1日

目录

[文本分类技术基础及实践 0](#_Toc7893251)

[目录 1](#_Toc7893252)

[基础知识篇 1](#_Toc7893253)

[思维导图 1](#_Toc7893254)

[第一章 文本分类技术 2](#_Toc7893255)

[1.1 分类算法 2](#_Toc7893256)

[1.1.1 多项式朴素贝叶斯 2](#_Toc7893257)

[1.1.2 贝努利朴素贝叶斯 4](#_Toc7893258)

[1.1.3 支持向量机 7](#_Toc7893259)

[1.1.4 线性分类器 11](#_Toc7893260)

[1.2 多类目预测 13](#_Toc7893261)

[1.2.1 单标签分类 13](#_Toc7893262)

[1.2.2 多标签分类 13](#_Toc7893263)

[1.3 模型选择 14](#_Toc7893264)

[1.3.1 交叉验证 14](#_Toc7893265)

[1.3.2 模型选择图 15](#_Toc7893266)

[1.3.3 模型选择表 15](#_Toc7893267)

[1.3.4 实践检验 16](#_Toc7893268)

[1.4 模型评价 16](#_Toc7893269)

[1.4.1 训练误差与测试误差 16](#_Toc7893270)

[1.4.2 过拟合与欠拟合 17](#_Toc7893271)

[1.4.3 混淆矩阵 18](#_Toc7893272)

[1.4.4 精确率、准确率、召回率、F值、宏平均与微平均 20](#_Toc7893273)

[1.4.5 ROC曲线与AUC值 23](#_Toc7893274)

[1.5 简单示例 25](#_Toc7893275)

[1.5.1 英文新闻文本分类 25](#_Toc7893276)

[1.5.2 英文影评情感分类 26](#_Toc7893277)

[1.5.3 英文垃圾邮件分类 32](#_Toc7893278)

[第二章 文本聚类技术 34](#_Toc7893279)

[2.1 数据规范化 35](#_Toc7893280)

[2.1.1 中心化变换 35](#_Toc7893281)

[2.1.2 极差正规化变换 35](#_Toc7893282)

[2.1.3 极差标准化变换 35](#_Toc7893283)

[2.1.4 标准化变换 35](#_Toc7893284)

[2.1.5 向量归一化 36](#_Toc7893285)

[2.2 相似性度量 36](#_Toc7893286)

[2.2.1 明氏距离 36](#_Toc7893287)

[2.2.2 曼哈顿距离 37](#_Toc7893288)

[2.2.3 欧式距离 37](#_Toc7893289)

[2.2.4 切比雪夫距离 37](#_Toc7893290)

[2.2.5 相关系数 38](#_Toc7893291)

[2.3 聚类算法 38](#_Toc7893292)

[2.3.1 K-Means 38](#_Toc7893293)

[2.3.2 MiniBatches 39](#_Toc7893294)

[2.3.3 Single-Pass 39](#_Toc7893295)

[2.4 简单示例 40](#_Toc7893296)

[2.4.1 K-Means示例 40](#_Toc7893297)

[2.4.2 Single-Pass示例 41](#_Toc7893298)

[第三章 特征工程 44](#_Toc7893299)

[3.1 文本表示 44](#_Toc7893300)

[3.1.1 词袋模型 44](#_Toc7893301)

[3.1.2 TF-IDF 45](#_Toc7893302)

[3.2 特征选择 47](#_Toc7893303)

[3.2.1 互信息 47](#_Toc7893304)

[3.2.2 卡方统计量 48](#_Toc7893305)

[3.2.3 频率 49](#_Toc7893306)

[3.2.4 信息增益 55](#_Toc7893307)

[3.2.5 不同特征选择方法的比较 57](#_Toc7893308)

[3.2.6 多类问题的特征选择方法 58](#_Toc7893309)

[3.3 关键词抽取 58](#_Toc7893310)

[3.3.1 TF-IDF关键词抽取 58](#_Toc7893311)

[3.3.2 基于互信息和左右信息熵的短语抽取 65](#_Toc7893312)

[第四章 sklearn库 67](#_Toc7893313)

[4.1 数据集 67](#_Toc7893314)

[4.1.1 数据集工具类 67](#_Toc7893315)

[4.1.2 文本数据集 68](#_Toc7893316)

[4.2 朴素贝叶斯 69](#_Toc7893317)

[4.2.1 MultinomialNB 69](#_Toc7893318)

[4.2.2 BernoulliNB 70](#_Toc7893319)

[4.3 支持向量机 71](#_Toc7893320)

[4.3.1 LinearSVC 71](#_Toc7893321)

[4.4 线性模型 72](#_Toc7893322)

[4.4.1 SGDClassifier 72](#_Toc7893323)

[4.5 K-Means聚类 74](#_Toc7893324)

[4.5.1 Kmeans 74](#_Toc7893325)

[4.6 模型选择 75](#_Toc7893326)

[4.6.1 StratifiedShuffleSplit 75](#_Toc7893327)

[4.6.2 train\_test\_split 76](#_Toc7893328)

[4.7 特征抽取 76](#_Toc7893329)

[4.7.1 CountVectorizer 76](#_Toc7893330)

[4.7.2 TfidfVectorizer 78](#_Toc7893331)

[4.8 模型评价 80](#_Toc7893332)

[4.8.1 confusion\_matrix 80](#_Toc7893333)

[4.8.2 classification\_report 80](#_Toc7893334)

[4.8.3 roc\_curve 81](#_Toc7893335)

[4.8.4 auc 81](#_Toc7893336)

[第五章 HanLP库 82](#_Toc7893337)

[5.1 词法分析器 82](#_Toc7893338)

[5.2 简繁转换 82](#_Toc7893339)

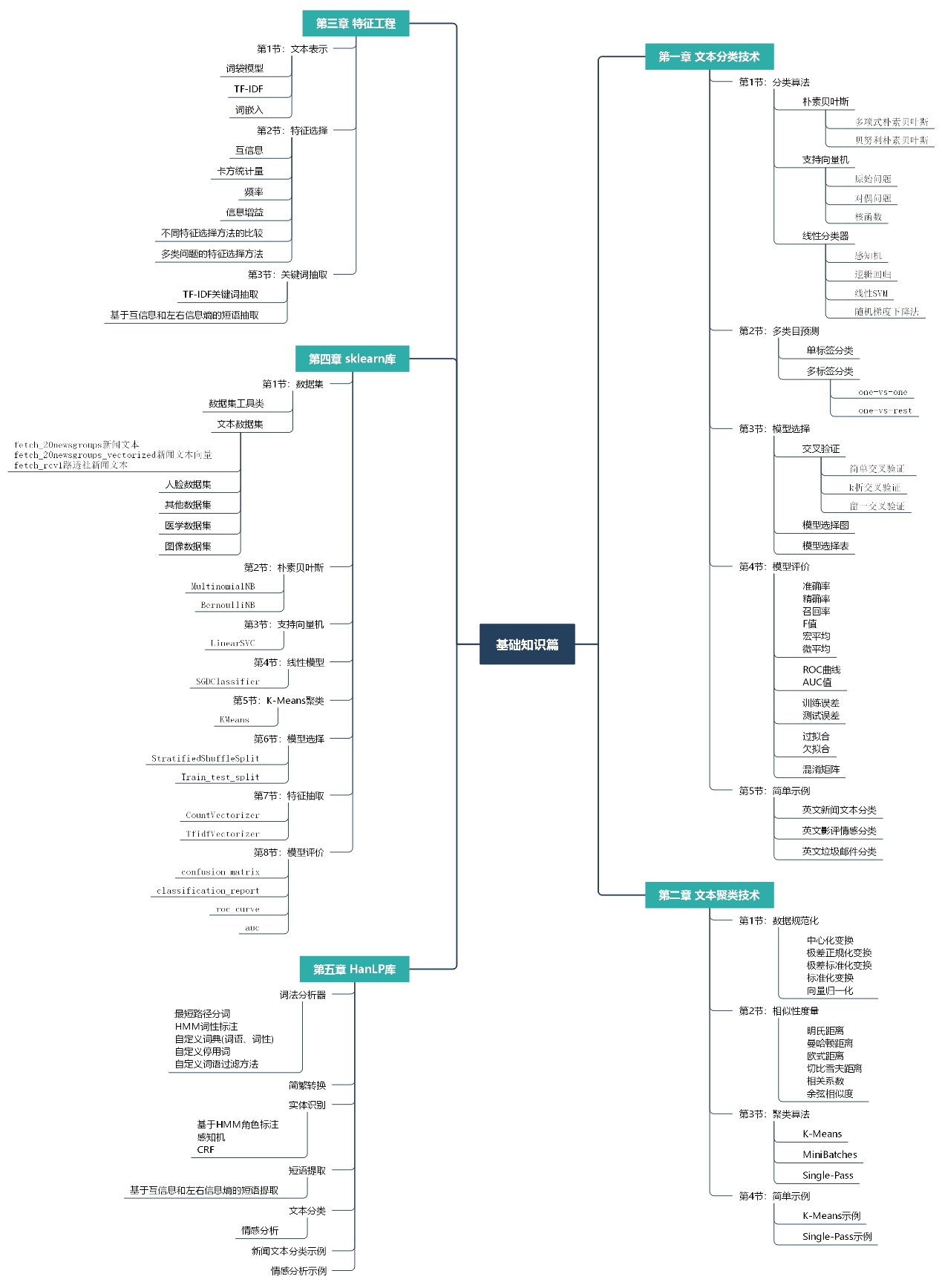
[5.3 实体识别 82](#_Toc7893340)

[5.4 短语抽取 82](#_Toc7893341)

[5.5 文本分类 82](#_Toc7893342)

基础知识篇

# 思维导图



# 文本分类技术

## 分类算法

### 多项式朴素贝叶斯

朴素贝叶斯法利用贝叶斯定理首先求出联合概率分布，再求出条件概率分布。这里的朴素是指在计算似然估计时假定了条件独立。基本原理可以用下面的公式给出：

其中，叫做后验概率，叫做先验概率，叫做似然概率，叫做证据。

在文本相关的分类预测中，常见的朴素贝叶斯形式有多项式朴素贝叶斯和下面将要提到的贝努利朴素贝叶斯。多项式朴素贝叶斯的公式为：

* 训练阶段
* 先验概率
* 条件概率

注：

1. 条件概率表示的是词在类别c中的权重。
2. 条件概率的计算引入了位置独立性假设。也就是说在c类任意一篇文档内不同位置的词的条件概率是相等的。
3. 上边两种概率值的计算都是利用的最大似然估计。它实际算出的是相对频率值，这些值能使训练数据的出现概率最大。

* 拉普拉斯平滑（加1平滑）

加1平滑可以认为是采用均匀分布作为先验分布，即每个词项在每个类别中出现一次，然后根据训练数据对得到的结果进行更新。也就是说未登陆词的估计值为 1/词汇表长度。

* 预测阶段

由计算式可以看出，预测阶段需要求和的项数为被预测文本所含词语个数+1

例：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 文档ID | 文档词列表 | 是否属于China类 |
| 训练集 | 1 | Chinese Beijing Chinese | Yes |
| 2 | Chinese Chinese Shanghai | Yes |
| 3 | Chinese Macao | Yes |
| 4 | Tokyo Japan Chinese | No |
| 测试集 | 5 | Chinese Chinese Chinese Tokyo Japan | ? |

训练：

先验概率 P(c=China)=3/4=0.75，P(c!=China)=1/4=0.25

类别c=China文档集词列表，词语总数8

Chinese Beijing Chinese Chinese Chinese Shanghai Chinese Macao

类别c!=China文档集词列表，词语总数3

Tokyo Japan Chinese

条件概率

|  |  |  |  |
| --- | --- | --- | --- |
| 词语序号 | 词语 |  |  |
| 1 | Chinese | (5+1)/(8+6)=0.429 | (1+1)/(3+6)=0.222 |
| 2 | Beijing | (1+1)/(8+6)=0.143 | (0+1)/(3+6)=0.111 |
| 3 | Shanghai | (1+1)/(8+6)=0.143 | (0+1)/(3+6)=0.111 |
| 4 | Macao | (1+1)/(8+6)=0.143 | (0+1)/(3+6)=0.111 |
| 5 | Tokyo | (0+1)/(8+6)=0.071 | (1+1)/(3+6)=0.222 |
| 6 | Japan | (0+1)/(8+6)=0.071 | (1+1)/(3+6)=0.222 |

显然，训练阶段需要估计的参数个数为 词汇表长度\*类别数量=6\*2=12。

预测：

待预测文档词语列表

Chinese Chinese Chinese Tokyo Japn

P(c=China| Chinese Chinese Chinese Tokyo Japn)=

P(c=China)\*P(Chinese|c=China)\*P(Chinese|c=China)\*P(Chinese|c=China)\*P(Tokyo|c=China)\*P(Japan|c=China)=0.75\*0.429\*0.429\*0.429\*0.071\*0.071=0.000299

P(c!=China| Chinese Chinese Chinese Tokyo Japn)=

P(c!=China)\*P(Chinese|c!=China)\*P(Chinese|c!=China)\*P(Chinese|c!=China)\*P(Tokyo|c!=China)\*P(Japan|c!=China)=0.25\*0.222\*0.222\*0.222\*0.222\*0.222=0.000135

因为P(c=China| Chinese Chinese Chinese Tokyo Japn)> P(c!=China| Chinese Chinese Chinese Tokyo Japn)模型认为待预测文档属于c=China类。

sklearn演示

由于这里我们预测的类别只有2个，所以按照线性分类来说只需要1个分类器，因此系数矩阵是1行的，也就是coef\_属性是1行的矩阵。但我们猜测内部还是一个有两行的矩阵，否则无法计算属于另一个类别的概率值。这里我们利用sklearn训练得到的模型的系数矩阵演示计算p(c=Japan)的过程。

首先定义数据集

|  |
| --- |
| # 训练集  texts = ['Chinese Beijing Chinese',  'Chinese Chinese Shanghai',  'Chinese Macao',  'Tokyo Japan Chinese']  # 各文本对应类别-1为China类，1为Japan类  y = [-1, -1, -1, 1]  # 测试集  test = ['Chinese Chinese Chinese Tokyo Japan'] |

接着我们进行文本表示

|  |
| --- |
| from sklearn.feature\_extraction.text import CountVectorizer  tf = CountVectorizer()  X = tf.fit\_transform(texts) |

之后我们实例化并训练多项式朴素贝叶斯

|  |
| --- |
| from sklearn.naive\_bayes import MultinomialNB  m = MultinomialNB(fit\_prior=True)  m.fit(X, y) |

查看训练得到的截距intercept以及多项式系数coef\_，以及各系数对应的特征词

|  |
| --- |
| tf.get\_feature\_names()返回  ['beijing', 'chinese', 'japan', 'macao', 'shanghai', 'tokyo']  m.intercept\_ = -1.38629436  m.coef\_ = [-2.19722458, -1.5040774 , -1.5040774 , -2.19722458, -2.19722458,-1.5040774 ] |

说明一下截距和系数的计算过程，也就是训练过程

p(c=Japan) = ln(1/4) = -1.3862943611198906

|  |  |
| --- | --- |
| 词语 |  |
| Chinese | ln((1+1)/(3+6))= -1.5040773967762742 |
| Beijing | ln((0+1)/(3+6))=-2.1972245773362196 |
| Shanghai | ln((0+1)/(3+6))= -2.1972245773362196 |
| Macao | ln((0+1)/(3+6))= -2.1972245773362196 |
| Tokyo | ln((1+1)/(3+6))= -1.5040773967762742 |
| Japan | ln((1+1)/(3+6))= -1.5040773967762742 |

表示测试集

|  |
| --- |
| X\_test = tf.transform(test)  对应的矩阵为  [0 3 1 0 0 1] |

预测的计算过程

prob\_log = b + w\*x

|  |
| --- |
| In [**18**]: b  Out[**18**]: -1.38629436  In [**19**]: w  Out[**19**]:  array([-2.19722458, -1.5040774 , -1.5040774 , -2.19722458, -2.19722458, -1.5040774 ])  In [**20**]: x  Out[**20**]: array([0, 3, 1, 0, 0, 1])  In [**21**]: b + np.dot(w,x)  Out[**21**]: -8.90668136 |

至此，就是sklearn中对多项式朴素贝叶斯的实现过程，可以对比一下前边的计算方法，只是所有概率值都取自然对数ln，所有概率相乘都变成取对数相加，会发现两种计算的过程是一致的。总结到这里，我们其实已经能够看出，sklearn中的多项式朴素贝叶斯就是针对文本分类而设计的，如果我们要应用多项式朴素贝叶斯到其他领域需要好好设计输入数据的向量化，否则计算过程就跟朴素贝叶斯的原理不一致了。还需要注意的是，sklearn为了使预测的两个类别的概率之和等于1，引入了规范化过程。

sklearn增加的规范化过程（使预测到两个类别的概率和为1）

|  |
| --- |
| In [**1**]: **import** **numpy** **as** **np**  In [**2**]: jll = np.array([np.log(0.000299), np.log(0.000135)])  In [**3**]: jll  Out[**3**]: array([-8.11506698, -8.91023578])  In [**4**]: log\_prob\_x = np.log(sum(np.exp(jll)))  In [**5**]: log\_prob\_x  Out[**5**]: -7.742466023863869  In [**6**]: log\_proba = jll - np.atleast\_2d(log\_prob\_x)  In [**7**]: log\_proba  Out[**7**]: array([[-0.37260096, -1.16776976]])  In [**8**]: proba = np.exp(log\_proba)  In [**9**]: proba  Out[**9**]: array([[0.68894009, 0.31105991]]) |

### 贝努利朴素贝叶斯

* 训练阶段
* 先验概率
* 条件参数
* 拉普拉斯平滑（加1平滑）

未登录词的估计值为 1/类别总数。

* 预测阶段

预测阶段需要的求和项数为 词汇表长度+1

例：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 文档ID | 文档词列表 | 是否属于China类 |
| 训练集 | 1 | Chinese Beijing Chinese | Yes |
| 2 | Chinese Chinese Shanghai | Yes |
| 3 | Chinese Macao | Yes |
| 4 | Tokyo Japan Chinese | No |
| 测试集 | 5 | Chinese Chinese Chinese Tokyo Japan | ? |

训练：

先验概率 P(c=China)=3/4=0.75，P(c!=China)=1/4=0.25

条件概率

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 词语序号 | 词语 | c=China包含的文档数 | c!=China包含的文档数 |  |  |
| 1 | Chinese | 3 | 1 | (3+1)/(3+2)=0.8 | (1+1)/(1+2)=0.667 |
| 2 | Beijing | 1 | 0 | (1+1)/(3+2)=0.4 | (0+1)/(1+2)=0.333 |
| 3 | Shanghai | 1 | 0 | (1+1)/(3+2)=0.4 | (0+1)/(1+2)=0.333 |
| 4 | Macao | 1 | 0 | (1+1)/(3+2)=0.4 | (0+1)/(1+2)=0.333 |
| 5 | Tokyo | 0 | 1 | (0+1)/(3+2)=0.2 | (1+1)/(1+2)=0.667 |
| 6 | Japan | 0 | 1 | (0+1)/(3+2)=0.2 | (1+1)/(1+2)=0.667 |

显然，训练阶段需要估计的参数个数为 词汇表长度\*类别数量=6\*2=12。

预测：

待预测文档词语集合

Chinese Tokyo Japn

P(c=China| Chinese Tokyo Japan)=

P(c=China)\*P(Chinese|c=China)\*P(Tokyo|c=China)\*P(Japan|c=China)\*(1-P(Beijing|c=China))+(1-P(Shanghai|c=China))\*(1-P(Macao|c=China))=0.75\*0.8\*0.2\*0.2\*(1-0.4)\*(1-0.4)\*(1-0.4)=0.005

P(c!=China|Chinese Tokyo Japan)=

P(c!=China)\*P(Chinese|c!=China)\*P(Tokyo|c!=China)\*P(Japan|c!=China)\*(1-P(Beijing|c!=China))\*(1-P(Shanghai|c!=China))\*(1-P(Macao|c!=China))=0.25\*0.667\*0.667\*0.667\*(1-0.333)\*(1-0.333)\*(1-0.333)=0.022

模型认为待预测文档属于c!=China类

注：

1. 当对长文档进行预测时，采用贝努利模型往往会因为某个词比如China在文档中出现一次而将整篇文档归于China类。
2. 词典中没有在被预测文档中出现的的词，对于多项式模型的预测没有影响，但是对于贝努利模型有影响，因为它的预测公式中考虑了这样的词发生的可能性。从预测计算过程可以看出，模型对于属于某类但未在待预测文本中出现的词给予了惩罚。

验证一下sklearn的实现结果

|  |
| --- |
| # 训练集  texts = ['Chinese Beijing Chinese',  'Chinese Chinese Shanghai',  'Chinese Macao',  'Tokyo Japan Chinese']  # 各文本对应类别-1为China类，1为Japan类  y = [-1, -1, -1, 1]  # 测试集  test = ['Chinese Chinese Chinese Tokyo Japan']  # 文本表示  from sklearn.feature\_extraction.text import CountVectorizer  tf = CountVectorizer(binary=True)  X = tf.fit\_transform(texts)  X\_test = tf.transform(test)  # 训练  from sklearn.naive\_bayes import BernoulliNB  m = BernoulliNB(fit\_prior=True)  m.fit(X, y)  # 预测  print(m.classes\_)  print(m.predict\_proba(X\_test))  print(m.predict\_log\_proba(X\_test))  # 输出结果  [-1 1]  [[0.19106679 0.80893321]]  [[-1.65513223 -0.21203892]] |

这里依然进行了规范化，我们将前边示例的结果按照规范化过程计算一遍

|  |
| --- |
| In [**20**]: jll = np.array([np.log(0.005), np.log(0.022)])  In [**21**]: jll  Out[**21**]: array([-5.29831737, -3.81671283])  In [**22**]: log\_prob\_x = np.log(sum(np.exp(jll)))  In [**23**]: log\_prob\_x  Out[**23**]: -3.611918412977808  In [**24**]: log\_proba = jll - np.atleast\_2d(log\_prob\_x)  In [**25**]: log\_proba  Out[**25**]: array([[-1.68639895, -0.20479441]])  In [**26**]: proba = np.exp(log\_proba)  In [**27**]: proba  Out[**27**]: array([[0.18518519, 0.81481481]]) |

与我们在示例中计算的结果是一致的。

多项式朴素贝叶斯模型与贝努利朴素贝叶斯模型的比较

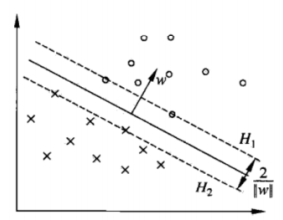
|  |  |  |
| --- | --- | --- |
|  | 多项式模型 | 贝努利模型 |
| 事件模型 | 词条生成模型 | 文档生成模型 |
| 随机变量 | ，当且仅当t出现在给定位置 | ，当且仅当t出现在文档中 |
| 文档表示 |  |  |
| 参数估计 |  |  |
| 决策规则：最大化 |  |  |
| 词项多次出现 | 考虑 | 不考虑 |
| 文档长度 | 能处理更长文档 | 最好处理短文档 |
| 特征数目 | 能够处理更多特征 | 特征数目较少效果更好 |
| 训练需要估计的参数个数 | |V|\*类别数量 | |V|\*类别数量 |
| 预测需要求和的项数 | 被预测文档所含词语数+1 | 词汇表长度+1 |

### 支持向量机

基于支持向量机的分类方法主要用于解决**二元模式**分类问题。SVM的基本思想是在向量空间中找到一个决策平面，这个平面能“最好”地分割两个类别中的数据点。支持向量机分类法就是要在训练集中找到具有**最大类间隔**的决策平面。这也是它与感知机的区别。

由于支持向量机算法是基于两类模式识别问题的，因而，对于多类模式识别问题通常需要建立多个两类分类器。与先行判别函数一样，它的结果强烈地依赖于已知模式样本集的构造，当样本容量不大时，这种依赖性尤其明显。此外，将分界面定在最大类间隔的中间，对于许多情况来说也不是最优的。对于线性不可分问题也可以采用类似于广义线性判别函数的方法，通过事先选择好的非线性映射将输入模式向量映射到一个高维空间，然后在这个高维空间中构造最优分类超平面。

#### 术语

间隔边界：图中虚线，

支持向量：间隔边界上的实例点

分离超平面：两条间隔边界的对称轴

间隔：两个间隔边界之间的距离

线性可分：存在超平面可以将两类样本分开（用数学式子描述就是支持向量机凸优化问题的约束条件）

上边的术语在平面上的数学描述为：

的直线方程为：，令，，，则直线方程为

的直线方程为：，令，，，则直线方程为

超平面（间隔边界）的直线方程为：

间隔（距离）的计算：

注：上边的直线方程中的y，都是实例的第2个维度坐标，并非类别标记。

#### 核心思想

核心思想是寻找一个超平面，使得两类实例点位于该超平面的两侧。由于这样的超平面不唯一，考虑到模型的鲁棒性，认为超平面应当使得所有实例点到超平面的距离最大。

如果超平面是间隔边界，则所有实例应该满足不等式，此不等式就是约束条件。距离最大就是目标函数，它等价于。

#### 线性可分支持向量机与线性支持向量机

|  |  |  |
| --- | --- | --- |
|  | 线性可分支持向量机 | 线性支持向量机 |
| **原始问题** | s.t. | s.t.  训练过程就是在最小化，因为表示的是对分隔边界移动的距离。之所以移动分隔边界就是使误分类点尽量少。  C值叫做惩罚系数，C值小，对误分点容忍度高，但损失精度；C值大，对误分点容忍度低，但容易过拟合。 |
| **对偶问题** | s.t. | s.t. |
| 对偶问题解表示原始问题解 | ，其中j是满足的下标 | ，其中j是满足的下标 |
| 决策函数 |  |  |

例：原始问题求解

已知一个训练数据集，其正实例点是x1=(3,3),x2=(4,3)，负实例点是x3=(1,1)，试求其最大间隔分离超平面。

解：按照原始问题，根据训练数据集构造约束最优化问题。

**显然这样的求解过程难以用机器实现，但可以将它转化为损失函数形式进行求解。**

求得此最优化问题的解。于是最大间隔分离超平面为。其中，x1=(3,3)与x3=(1,1)为支持向量。支持向量是间隔边界上的实例点，间隔边界的方程为wx+b=1以及wx+b=-1，凡是这两个方程的解都是支持向量。

例：对偶问题求解

已知一个训练数据集，其正实例点是x1=(3,3), x2=(4,3)，负实例点是x3=(1,1)，用对偶问题求其最大间隔分离超平面。

解：按对偶问题，根据训练数据集构造约束最优化问题。

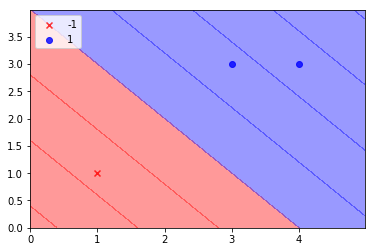
**对偶问题可用SMO算法求解**

**对偶形式将w,b的求解转化成对alpha的求解，当训练样本数量<=特征词数量时，推荐使用对偶形式。**

解这一最优化问题，将代入目标函数并记为，对，求偏导数并令其为0，解得极值点，，则。其中，，对应的实例点即为支持向量。于是，。分离超平面为。

sklearn演示

|  |
| --- |
| # 带有L2罚项的合页损失对偶形式  # 定义训练数据  X = np.array([[3,3],[4,3],[1,1]])  y = np.array([1, 1, -1])  # Liblinear使用正则化项penalty的求解  from sklearn.svm import LinearSVC  # 实例化模型，**系数coef\_唯一在但截距intercept\_不唯一，可以通过intercept\_scaling调节**  clf = LinearSVC(random\_state=0, fit\_intercept=True, **intercept\_scaling=2,penalty='l2', loss='hinge', dual=True**)  # 训练模型  clf.fit(X, y)  # 打印系数矩阵  print("w=", clf.coef\_)  # 打印截距  print("b=", clf.intercept\_)  **w= [[0.49997052 0.49997052]]**  **b= [-1.99994104]** |
| **# 验证线性支持向量机对偶形式的约束最小化问题等价于带有L2罚项的合页损失无约束最小化问题**。这里使用梯度下降法  learning\_rate = 0.01  max\_iter = 1000  C = 1  # 权值初始值  w = np.array([0, 0])  # 偏置初始值  b = 0  # 训练过程  iter\_cnt = 0  for j in range(1, max\_iter+1):  # 计算误差向量  ei\_li = []  for i in range(X.shape[0]):  ei = 1 - y[i] \* (np.dot(w, X[i, :]) + b)  ei\_li.append(ei)  # 取出误差最大项  max\_e = max(ei\_li)  max\_e\_index = ei\_li.index(max\_e)  # 最大误差项<=0 退出  if max\_e <= 0:  break  # 更新参数  w = (1-learning\_rate) \* w + learning\_rate \* C \* y[max\_e\_index] \* X[max\_e\_index, :]  b = b + learning\_rate \* C \* y[max\_e\_index]  iter\_cnt += 1  print("iter ", iter\_cnt)  print("w=", w)  print("b=", b)  iter 633  **w= [0.50190251 0.50188436]**  **b= -2.010000000000001** |



这里**liblinear使用损失函数的形式求解最优化问题**，下边对原始形式与损失函数形式的对应关系进行说明，**原始形式等价于损失函数形式**的证明可以参考《统计学习方法》p114页定理7.4的证明。原始形式为

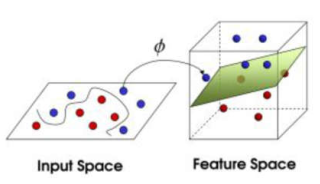
s.t.

这个约束最小化问题等价于下边的无约束最小化问题

这就是损失函数形式，其中第1项为L2正则化罚penalty项对应于原始问题的第1项，第2项为合页损失hinge项对应于原始问题的第2项。我们这个示例里初始化的是**liblinear中L2罚项、合页损失、对偶形式的线性支持向量机**。有关于**原始形式推导到对偶形式**可以参考《统计学习方法》p110页7.2.2节“学习的对偶算法”。

#### 非线性支持向量机

对于文本分类来说，一般情况下实例点都是线性可分的，也就是说即使采用非线性分类器，也不会使分类精度有明显提升。因此这里只是简单说明非线性支持向量机的原理。我们用一张非线性变换图来形象说明核函数的作用。



从图中可以看到原来的实例点在一个2维空间上，显然在这里只能使用一条曲线将两类实例点分离到曲线两侧。那么怎么才能用一个线性超平面将两类实例点分开呢？答案是使用非线性函数，将2维空间中的实例点映射到3维空间，也就是图中的feature space，这时就可以在3维空间内找到一个平面，将两种实例分离到平面的两侧。需要说明的是，当把数据映射到高维空间后，每一个实例点都需要更多的数值来表示，这样在计算对偶公式中的内积时会使计算量增大，这是不能忍受的，怎样解决这个问题呢？根据mercer定理，针对某一种可以找到一个核函数K，用这个核函数K在低维空间中计算出某两个实例的值，这个值刚好就是这两个实例在高维空间中的内积值，这样就可以避免在高维空间计算内积，从而减少了计算量。

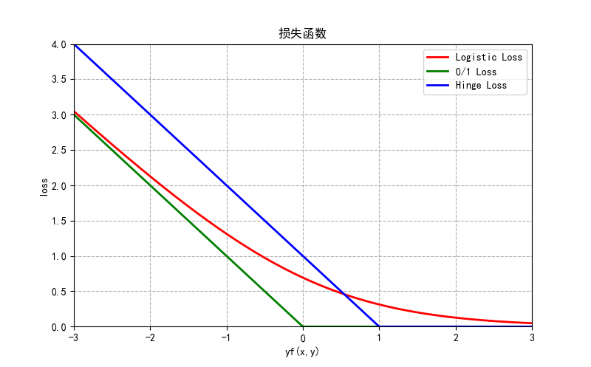
### 线性分类器

线性分类器指的是模型认为数据是按照类别线性可分的。通常这样的模型可以通过随机梯度下降法求解模型参数。线性模型是机器学习中应用最广泛的模型，指通过样本特征的线性组合来进行预测的模型。常见的线性模型有logistic回归，感知机，线性支持向量机。他们的线性组合函数为，预测函数为，其中sgn()为符号函数。

三种线性模型的区别只在于损失函数不同。为了比较这些损失函数，我们统一定义类别标签，并定义，这样对于样本(x, y)，若yf(x,w)>0，则分类正确，否则分类错误。于是，将它们的损失函数都定义为在yf(x,w)上的函数，下边分别列出三种模型的损失函数。

|  |  |
| --- | --- |
| logistic回归 |  |
| 感知机 |  |
| 线性软间隔支持向量机 |  |

损失函数曲线



代码

|  |
| --- |
| *#coding:utf-8* **import** numpy **as** np **import** math **import** matplotlib.pyplot **as** plt  plt.rcParams[**'font.sans-serif'**] = [**'SimHei'**] plt.rcParams[**'axes.unicode\_minus'**] = **False** plt.figure(figsize=(8, 5))  x = np.arange(-10, 10, 0.01) *# 逻辑回归损失函数* logi = np.log(1 + np.exp(-x)) *# 感知机损失函数* y\_p = -x y\_p[y\_p < 0] = 0 *# 线性支持向量机* y\_hinge = 1.0 - x y\_hinge[y\_hinge < 0] = 0  plt.xlim([-3, 3]) plt.ylim([0, 4]) plt.plot(x, logi, **'r-'**, mec=**'k'**, label=**'Logistic Loss'**, lw=2) plt.plot(x, y\_p, **'g-'**, mec=**'k'**, label=**'0/1 Loss'**, lw=2) plt.plot(x, y\_hinge, **'b-'**, mec=**'k'**, label=**'Hinge Loss'**, lw=2) plt.grid(**True**, ls=**'--'**) plt.legend(loc=**'upper right'**) plt.title(**'损失函数'**) plt.xlabel(**'yf(x,w)'**) plt.ylabel(**'loss'**) plt.show() |

梯度下降法按照计算训练误差时使用训练数据的方式分为以下三种：批量梯度下降法（Batch gradient descent, BGD），随机梯度下降法（Stochastic gradient descent, SGD），小批量梯度下降法（Mini-Batch gradient descent, MBGD）。

1. 批量梯度下降法。每次迭代使用所有的样本，这样做的好处是每次迭代都顾及了全部的样本，考虑的是全局最优化。需要注意的是这个名字并不确切，但是机器学习领域中都这样称。它的缺点是每次迭代都要计算训练集中所有样本的训练误差，当数据量很大时，这种方法效率不高。
2. 随机梯度下降法。每次迭代都随机从训练集中抽取出1个样本，在样本量极其大的情况下，可能不用抽取出所有样本，就可以获得一个损失值在可接受范围之内的模型了。**缺点是由于单个样本可能会带来噪声，导致并不是每次迭代都向着整体最优方向前进。**
3. 小批量梯度下降法。它介于批量梯度下降法与随机梯度下降法之间。每次迭代随机从训练集抽取一定数量的数据进行训练。

## 多类目预测

根据类别之间是否互斥，可以将多类问题的分类细分为两类问题。第1类问题是多标签分类问题，它指的是一个文本可以同时属于多个类别，类别之间不是互斥的。第2类问题是单标签问题，它指的是一个文本最多只能属于1个类别，即多个类别之间是互斥的。

### 单标签分类

处理单标签分类可以直接使用多分类模型，比如k近邻、朴素贝叶斯、决策树等。对于二类分类模型需要使用组合策略，算法描述如下：

1. 对每个类别建立一个分类器，此时训练集包含所有属于该类的文档和所有不属于该类的文档
2. 给定测试文档，分别使用每个分类器进行分类
3. 将文档分配给得分最高的类、置信度最高的类或概率最大的类

建立分类器的方法可以有两种，分别为one-vs-one和one-vs-rest。**one-vs-one**方法就是在建立分类器时随机抽取两个类别（不考虑顺序），因此分类器的数量为n(n-1)/2（n为类别数量）。**one-vs-rest**方法就是在建立分类器时选定一个类别作为一组，其余类别作为一组来构建分类器，因此分类器的数量为n。

### 多标签分类

1. 对每个类别建立一个分类器，此时训练集包含所有属于该类的文档和所有不属于该类的文档
2. 给定测试文档，分别使用每个分类器进行分类，每个分类器的分类结果并不影响其他分类器的结果

## 模型选择

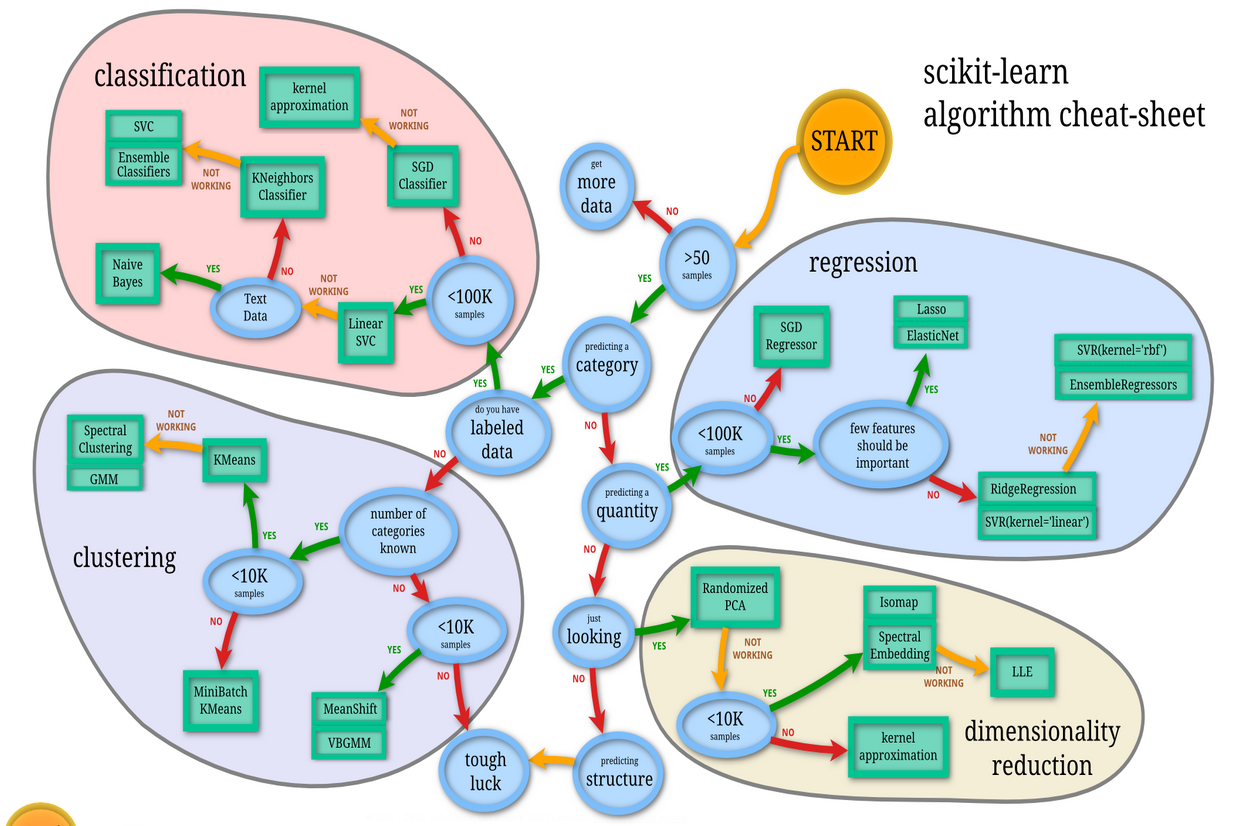
### 交叉验证

首先回答下边这个问题。为什么要将数据分为训练集、验证集（交叉验证数据集）和测试集？因为三种数据集有各自的用途。训练集用来训练模型的参数，验证集用来选择最合适的模型，测试集用来测试模型的准确性。一般情况下，我们将数据按6:2:2分成训练集、交叉验证集和测试集。

通过回答上边的问题我们知道，模型的选择是通过比较不同模型在验证集上的准确性来实现。也就是说在验证集上准确性最高的模型就认为是最优的模型。在验证集上评估模型准确性的方法有以下三种：简单交叉验证法，k折交叉验证法，留一法。为了方便说明总结成下表形式：

|  |  |  |
| --- | --- | --- |
| 类型 | 处理过程 | 优缺点 |
| 简单交叉验证 | 数据被随机分为两组，分别作为训练集和验证集，在训练集上训练，在验证集上计算准确率。 | 随机将数据划分为训练集和验证集，那么在验证集上的准确率会随着划分比例的变化而变化，因此这样得到的验证集上的准确率通常不具有说服力。 |
| K折交叉验证 | 数据被均等分为k组，每次取1组作为验证集，其他组作为训练集。分别在k个训练集上训练，并对应在k个验证集上计算出k个准确率，最后求平均值作为模型在验证集上最终的准确率。 | 有效避免了过拟合和欠拟合，最终结果更具说服力。 |
| 留一交叉验证 | 每次取1个实例作为测试集，其他实例作为训练集。分别在n（实例总数）个训练集上训练，并对应在n个验证集上计算出n个准确率，最后求平均值作为模型在验证集上最终的准确率。 | 结果相对可靠。确保实验过程可复现。缺点计算成本高，当数据量很大时，该方法实用性不强。 |

### 模型选择图



上图为sklearn提供的关于模型选择的参考图。该图的源网页为

（<https://scikit-learn.org/stable/tutorial/machine_learning_map/>）。图中左上为分类算法，左下为聚类算法。在聚类算法中，GMM、VBGMM为语音处理算法。MeanShift为图像处理算法。因此应用于文本的聚类算法有KMeans（数据量小于1万）、MiniBatchKMeans（数据量大于1万）、Spectral Clustering（数据量小于1万），它们都是用来解决已知簇数量的算法，对于未知簇数量的情况，可以采用增量聚类算法比如single-pass。在分类算法中当数据量小于10万时，使用分类器LinearSVC或者NaiveBayes。当数据量大于10万时采用随机梯度下降线性分类器。

### 模型选择表

|  |  |  |  |
| --- | --- | --- | --- |
| 模型 | 优点 | 缺点 | 应用 |
| **NB** | 1.**大数据量**  2.适用多类问题分类  3.可以进行增量训练  4.结果易解释便于分析误判 | 1.**特征之间有关联时效果受影响**。比如两词短语所表达的语义与单独的词语表达的语义不同时。或者两词短语不同的排列顺序具有不同的语义时。 | 1.欺诈检测  2.垃圾邮件  3.文本分类  4.情感分类 |
| LR | 1.预测速度快  2.模型占用内存小  3.多种正则化方法避免过拟合  4.不必担心特征间存在相关性 | 1.**不适合高维数据**  2.容易欠拟合  3.适用于二类分类  4.数据必须线性可分 | 1.检索排序  2.信用评估  3.产品收益  4.地震预测 |
| **KNN** | 1.非线性分类  2.训练时间快  3.异常数据不敏感  4.可以进行增量训练 | 1.**不平衡数据效果差**  2.模型占用内存大  3.**计算量大（相似度距离）**  4.**k值选择没有理论指导**  5.容易过拟合 | 1.文本分类  2.聚类分析 |
| 决策树 | 1.易于可视化  2.易于制定规则  3.同时处理标称型和离散型数据 | 1.容易过拟合  2.属性选择时易受取值数量影响 | 1.企业管理  2.投资 |
| SVM | 1.高维数据  2.**小样本**  3.非线性  4.**泛化能力强**  5.模型占用内存小 | 1.**二分类**  2.核函数不好找（当然一般在文本分类中使用线性SVM） | 1.文本分类 |

### 实践检验

1. 一个普遍的事实就是，采用领域相关的文本特征在效果上会比采用新的机器学习方法获得更大的提升。
2. Jackson和Moulinier（2002）指出：“对数据的理解是分类成功的关键之一”。
3. 当面对一个具体的分类需求时，第一个要问的问题就是：训练数据（已标注的数据）有多少？如果有足够多的时间用于系统实现的话，那么大部分时间可能要花在数据的准备上。
4. 在没有标注数据的情况下，一般首先采用编制规则的方法。一个基本合理的估计数字是每个类别需要标注两天的时间。
5. 在已标注数据较少的情况下，一般采用高偏差分类器，比如贝叶斯。当然，无论采用何种模型，模型的质量始终会因训练数据有限而受到不利影响。
6. 快速标注数据的方法bootstrapping方法。将分类器不太好分的文本交给人工进行标注。
7. 快速进行错误纠正的方法是在分类模型之上，再建立一个基于布尔规则的分类器。
8. 如果具有极大规模的数据，那么分类器的选择也许对最后的结果没有什么影响。一般可以从训练规模扩展性或运行效率上来选择分类器。
9. 一个通用的经验法则是，训练数据规模每增加一倍，那么分类器的效果将得到线性的提高。
10. 对于类别数目很多分类问题，可以采用分层策略。
11. 将特殊字符串（比如ISBN号，化学式等）按照类别统一成一种符号。
12. 用短语作为特征。用命名实体作为特征。

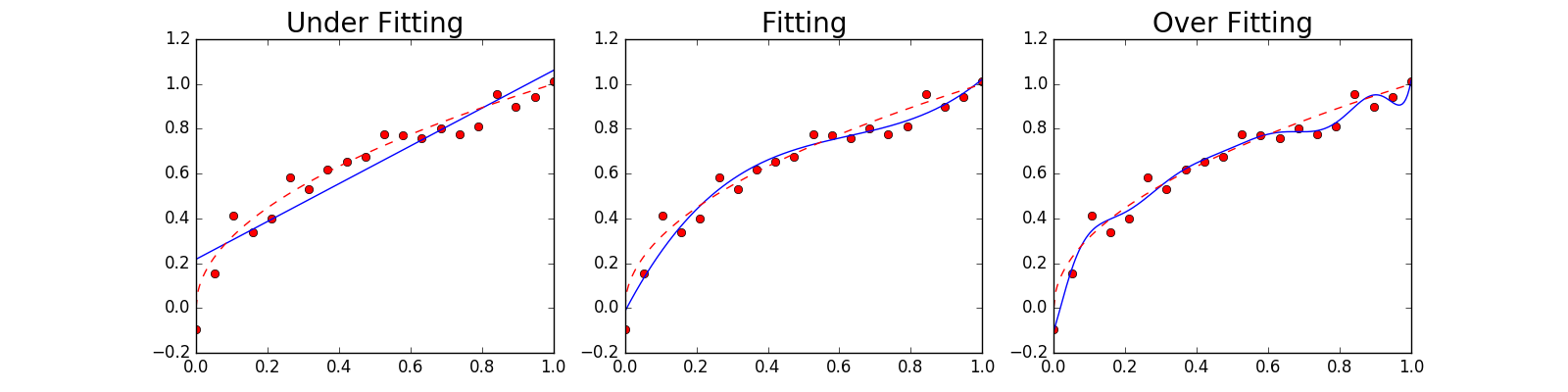
## 模型评价

### 训练误差与测试误差

**训练误差**就是模型在训练集上的误差平均值，度量了模型对训练集拟合的情况。训练误差大说明对训练集特性学习得不够，训练误差太小说明过度学习了训练集特性，容易发生过拟合。**测试误差**是模型在测试集上的误差平均值，度量了模型的泛化能力。在实践中，希望测试误差越小越好。

### 过拟合与欠拟合

如同上一小结的阐述，**过拟合**就是模型过度学习了训练集所有特性，导致模型认为训练集中的某些特性也是潜在测试实例具有的一般性质。从误差角度来说，过拟合时训练误差小但测试误差却很大。**欠拟合**就是说模型尚未学习完整训练集实例的普适特性。从误差角度来说，欠拟合时训练误差大，测试误差也大。为了防止过拟合出现，需要根据不同模型采用不同的方法。对于优化损失函数的模型比如感知机、逻辑回归、SVM等可以在损失函数中加入正则化项（罚项），正则化项一般是模型参数的范数。对于决策树这样的模型，可以通过剪枝达到避免过拟合的目的。下面用下边的实例来说明欠拟合和过拟合：



上图中，左边是欠拟合（underfitting），也称为高偏差（high bias）因为我们试图用一条直线来拟合样本数据。右边是过拟合（overfitting），也称为高方差（high variance），用了十阶多项式来拟合数据，虽然模型对现有的数据集拟合得很好，但对新数据预测误差却很大。只有中间的模型较好地拟合了数据集，可以看到虚线和实线基本重合。

示例代码

|  |
| --- |
| *#coding:utf-8 """ 过拟合和欠拟合演示 """* **import** matplotlib.pyplot **as** plt **import** numpy **as** np   *# 画出拟合出来的多项式所表达的曲线以及原始的点* **def** plot\_polynomial\_fit(x, y, order):  p = np.poly1d(np.polyfit(x, y, order))  t = np.linspace(0, 1, 200)  plt.plot(x, y, **'ro'**, t, p(t), **'-'**, t, np.sqrt(t), **'r--'**)  **return** p   *# 生成20个点的训练样本* n\_dots = 20 x = np.linspace(0, 1, n\_dots) *# [0, 1] 之间创建 20 个点* y = np.sqrt(x) + 0.2\*np.random.rand(n\_dots) - 0.1;  plt.figure(figsize=(18, 4)) titles = [**'Under Fitting'**, **'Fitting'**, **'Over Fitting'**] models = [**None**, **None**, **None**] **for** index, order **in** enumerate([1, 3, 10]):  plt.subplot(1, 3, index + 1)  models[index] = plot\_polynomial\_fit(x, y, order)  plt.title(titles[index], fontsize=20) plt.show() |

### 混淆矩阵

混淆矩阵对角线上的元素表示的是正确预测各类别的实例数量，而非对角线上的元素表示误分类实例数量。

对角线元素值之和越大，表示正确预测实例数量越多。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 预测类别实际类别 | money-fx | trade | interest | wheat | corn | grain |
| money-fx | 95 | 0 | 10 | 0 | 0 | 0 |
| trade | 1 | 1 | 90 | 0 | 1 | 0 |
| interest | 13 | 0 | 0 | 0 | 0 | 0 |
| wheat | 0 | 0 | 1 | 34 | 3 | 7 |
| corn | 1 | 0 | 2 | 13 | 26 | 5 |
| grain | 0 | 0 | 2 | 14 | 5 | 10 |

上表中有14篇属于grain类的文档被误分到wheat类中。

需要说明的几点:

1. 主对角线元素表示各类别中预测值与实际值相同的实例数量，因此各主对角线元素的值为tp
2. 去掉某一主对角线元素后，对其所在列求和的值为fp
3. 去掉某一主对角线元素后，对其所在行求和的值为fn

混淆矩阵的sklearn示例。这里在iris数据集上训练一个分类器，并用混淆矩阵来评价。图像分别对未经规范化和经过规范化的混淆矩阵进行了可视化。这种规范化后的可视化有利于评价不平衡分类。这里的分类器没有达到其应有的准确率。因为正则化参数C没有选择好。在真实的应用中，通常通过`grid\_search`来确定C的值。

|  |
| --- |
| """  ================  Confusion matrix  ================  混淆矩阵用法的例子。这里在iris数据集上训练一个分类器，并用混淆矩阵来评价。  矩阵对角线上的元素表示的是正确预测各类别的实例数量，而非对角线上的元素表示误分类实例数量。  对角线元素值之和越大，表示正确预测实例数量越多。  图像分别对未经规范化和经过规范化的混淆矩阵进行了可视化。  这种规范化后的可视化有利于评价不平衡分类。  这里的分类器没有达到其应有的准确率。因为正则化参数C没有选择好。  在真实的应用中，通常通过`grid\_search`来确定C的值。  """  print(\_\_doc\_\_)  import itertools  import numpy as np  import matplotlib.pyplot as plt  from sklearn import svm, datasets  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import confusion\_matrix  # import some data to play with  iris = datasets.load\_iris()  X = iris.data  y = iris.target  class\_names = iris.target\_names  # Split the data into a training set and a test set  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)  # Run classifier, using a model that is too regularized (C too low) to see  # the impact on the results  classifier = svm.SVC(kernel='linear', C=0.01)  y\_pred = classifier.fit(X\_train, y\_train).predict(X\_test)  def plot\_confusion\_matrix(cm, classes,  normalize=False,  title='Confusion matrix',  cmap=plt.cm.Blues):  """  This function prints and plots the confusion matrix.  Normalization can be applied by setting `normalize=True`.  """  if normalize:  cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  print("Normalized confusion matrix")  else:  print('Confusion matrix, without normalization')  print(cm)  plt.imshow(cm, interpolation='nearest', cmap=cmap)  plt.title(title)  plt.colorbar()  tick\_marks = np.arange(len(classes))  plt.xticks(tick\_marks, classes, rotation=45)  plt.yticks(tick\_marks, classes)  fmt = '.2f' if normalize else 'd'  thresh = cm.max() / 2.  for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):  plt.text(j, i, format(cm[i, j], fmt),  horizontalalignment="center",  color="white" if cm[i, j] > thresh else "black")  plt.tight\_layout()  plt.ylabel('True label')  plt.xlabel('Predicted label')  # Compute confusion matrix  cnf\_matrix = confusion\_matrix(y\_test, y\_pred)  np.set\_printoptions(precision=2)  # Plot non-normalized confusion matrix  plt.figure()  plot\_confusion\_matrix(cnf\_matrix, classes=class\_names,  title='Confusion matrix, without normalization')  # Plot normalized confusion matrix  plt.figure()  plot\_confusion\_matrix(cnf\_matrix, classes=class\_names, normalize=True,  title='Normalized confusion matrix')  plt.show() |

### 精确率、准确率、召回率、F值、宏平均与微平均

3类分类的混淆矩阵

实际值序列 -1，0，1，1，-1

预测值序列 -1，-1，1，0，-1

根据这两个序列，填写下边的混淆矩阵

|  |  |  |  |
| --- | --- | --- | --- |
|  | 预测-1 | 预测0 | 预测1 |
| 实际-1 | 2 | 0 | 0 |
| 实际0 | 1 | 0 | 0 |
| 实际1 | 0 | 1 | 1 |

根据这个混淆矩阵计算出每个类别的tp值，fp值，fn值

|  |  |  |  |
| --- | --- | --- | --- |
|  | tp | fp | fn |
| -1类 | 2 | 1 | 0 |
| 0类 | 0 | 1 | 1 |
| 1类 | 1 | 0 | 1 |

#### 准确率

考虑各个类别，计算整体的平均准确性。对于不平衡分类，准确率并不是一个好的衡量指标。这是因为一个将所有文档都不归于小类的分类器会获得很高的准确率，但是这显然并不能说明系统实际的准确性。因此对于不平衡分类来说，精确率、召回率和F值才是更好的衡量指标。该指标可以通过sklearn分类模型实例的score方法得到。

#### 精确率

度量分类器在某一个类别上的预测精确性。

#### 召回率

度量分类器在某一个类别上的预测覆盖面。

#### F值

度量分类器在某一个类别上的预测精确性以及覆盖面。它是精确率和召回率的调和平均值。

#### 宏平均与微平均

1. 宏平均
2. 微平均

微平均实际上是文档集中大类（含有很多数据的类目）上的一个效果度量指标，如果要度量小类上的效果，往往需要计算宏平均指标。

示例

3类分类的混淆矩阵

实际值序列 -1，0，1，1，-1

预测值序列 -1，-1，1，0，-1

根据这两个序列，填写下边的混淆矩阵

|  |  |  |  |
| --- | --- | --- | --- |
|  | 预测-1 | 预测0 | 预测1 |
| 实际-1 | 2 | 0 | 0 |
| 实际0 | 1 | 0 | 0 |
| 实际1 | 0 | 1 | 1 |

根据这个混淆矩阵计算出每个类别的tp值，fp值，fn值，precision值，recall值，F值，整体的macro avg，mico avg。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 预测-1 | 预测0 | 预测1 | precision | recall | F |
| 实际-1 | 2 | 0 | 0 | 2/(2+1)=0.67 | 2/(2+0)=1 | (2\*0.67\*1)/(0.67+1)=0.8 |
| 实际0 | 1 | 0 | 0 | 0/(0+1)=0 | 0/(0+1)=0 | 0 |
| 实际1 | 0 | 1 | 1 | 1/(1+0)=1 | 1/(1+1)=0.5 | (2\*0.5\*1)/(1+0.5)=0.67 |

macro\_avg=(0.67+0+1)/3=0.56

micro\_avg=(2+0+1)/(2+0+0+1+0+0+0+1+1)=0.6

sklearn中通过classification\_report实现上边各种指标，示例如下：

|  |
| --- |
| >>> y\_true = [-1, 0, 1, 1, -1]  >>> y\_pred = [-1, -1, 1, 0, -1]  >>> from sklearn.metrics import confusion\_matrix  >>> confusion\_matrix(y\_true, y\_pred)  array([[2, 0, 0],  [1, 0, 0],  [0, 1, 1]])  >>> from sklearn.metrics import classification\_report  >>> target\_names = ['class 0', 'class 1', 'class 2']  >>> print(classification\_report(y\_true, y\_pred, target\_names=target\_names))  precision recall f1-score support  class 0 0.67 1.00 0.80 2  class 1 0.00 0.00 0.00 1  class 2 1.00 0.50 0.67 2  micro avg 0.60 0.60 0.60 5  macro avg 0.56 0.50 0.49 5  weighted avg 0.67 0.60 0.59 5 |

这里混淆矩阵中行以及列的序号对应的类别整数依次是预测值或预测值去重升序排列后对应的类别整数值。因此第0行对应的类别是-1，第1行对应的类别是0，第2行对应的类别是1，列与此相同。classification\_report中target\_names列表中的类别名称依次对应的是预测值去重生序排列后对应的类别整数值。因此’class 0’对应的类别是-1，’class 1’对应的类别是0，’class 2’对应的类别是1。

### ROC曲线与AUC值

ROC曲线和AUC值是处理不平衡分类问题的评价方法。显示分类器真正率和假正率之间折中的一种图形化方法。真正率和假正率的定义将在下边介绍。一个好的分类模型的ROC曲线应尽可能靠近面积为1的正方形的左上角。AUC值是ROC曲线下的面积。AUC值越大，分类器效果越好。

首先给出二类分类的混淆矩阵

|  |  |
| --- | --- |
| 真阳性TP | 假阴性 FN |
| 假阳性FP | 真阴性 TN |

**真正率（true positive rate, TPR）或灵敏度（sensitivity）**定义为被模型正确预测的正样本的比例TPR=TP/(TP+FN)。

**假正率（false positive rate, FPR）**定义为被预测为正类的负样本比例FPR=FP/(TN+FP)。

在度量不平衡分类的分类模型时，将稀有类定义为正类，常见类定义为负类。

ROC曲线的绘制过程：

为了能够绘制ROC曲线，分类器需要能提供预测类别的得分值，用来对预测为正类的实例按得分排序，最不肯定的排在前，最肯定的排在后。需要注意的是，这个得分是预测为正类（稀有类）的分值，而不是正、负类中得分最高的值。

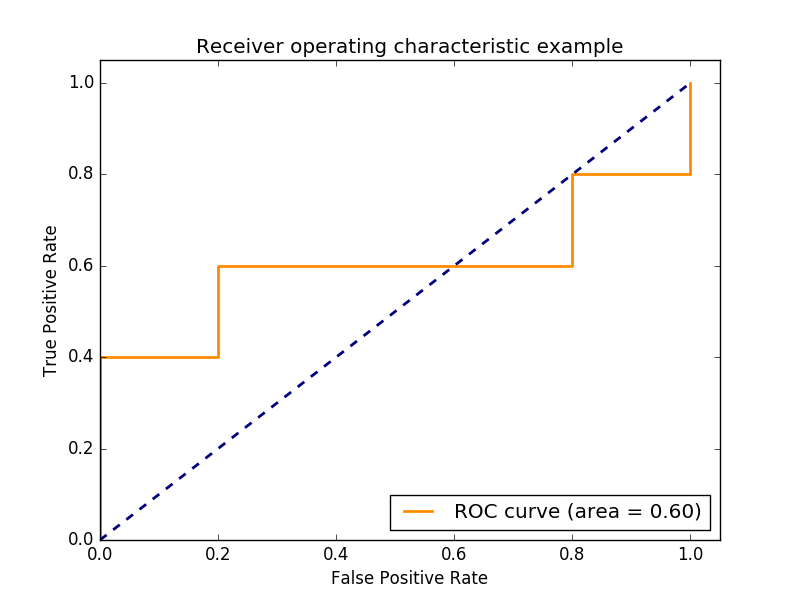
1. 让模型对每一个实例进行预测，记录正类得分，并按得分将实例升序排列。
2. 从排序列表中按顺序选择第1个得分最小的记录，从该记录开始到列表结束的所有记录都被指定为正类，其他实例指定为负类，计算混淆矩阵并计算TPR，FPR。此时，TPR=FPR=1。
3. 从排序列表中选择下1个记录，从该记录开始到列表结束的所有记录都被指定为正类，其他实例指定为负类，计算混淆矩阵并计算TPR，FPR
4. 重复步骤（3），直到列表中所有实例都被选择过。
5. 以FPR为横轴，TPR为纵轴，描点绘制ROC曲线。

示例

下表中，每一列表示一个实例，已经按照预测为正类的得分升序排列。第1行为实例的实际类别，第2行为实例被模型预测为正类的得分。请计算出第2行之后的各行表示的混淆矩阵元素值以及TPR、FPR值。

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **实例的实际类别** | **+** | **-** | **+** | **-** | **-** | **-** | **+** | **-** | **+** | **+** |
| **模型预测为正类得分** | **0.25** | **0.43** | **0.53** | **0.76** | **0.85** | **0.85** | **0.85** | **0.87** | **0.93** | **0.95** |
| **TP** | 5 | 4 | 4 | 3 | 3 | 3 | 3 | 2 | 2 | 1 |
| **FP** | 5 | 5 | 4 | 4 | 3 | 2 | 1 | 1 | 0 | 0 |
| **TN** | 0 | 0 | 1 | 1 | 2 | 3 | 4 | 4 | 5 | 5 |
| **FN** | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 4 |
| **FPR** | 1 | 0.8 | 0.8 | 0.6 | 0.6 | 0.6 | 0.6 | 0.4 | 0.4 | 0.2 |
| **TPR** | 1 | 1 | 0.8 | 0.8 | 0.6 | 0.4 | 0.2 | 0.2 | 0 | 0 |

解：首先选择第1个实例，按照绘制过程的第（2）个步骤，此时所有实例都被指定为+，则比较表中第1行的实际类别，可以计算出TP=5, FP=5, TN=0, FN=0, TPR=1, FPR=1。将计算得到的值填入表中第1列相应位置。接着按照第（3）个步骤，选择第2个实例，此时从第2到第10的8个实例指定为正类，其余实例即第1个实例指定为负类，计算TP=4,FP=5,TN=0,FN=1,TPR=4/(4+1)=0.8, FPR=5/(5+0)=1。依次类推计算第3-8列的各行元素值。接下来便可以FPR为横轴，TPR为纵轴，描出表中给(fpr,tpr)点，绘制模型的ROC曲线。



|  |
| --- |
| #coding:utf-8  """  绘制ROC曲线，计算AUC值示例  """  import numpy as np  from sklearn.metrics import roc\_curve,auc  import matplotlib.pyplot as plt  # 实际类别只能取{0,1}或{1,-1}  y = np.array([1,0,1,0,0,0,1,0,1,1])  # 对应预测为正（即1）的得分。注意：得分相同的实例只保留一个  scores = np.array([0.25,0.43,0.53,0.76,0.85,0.86,0.87,0.89,0.93,0.95])  # pos\_label 假定为正类的类别标记，这里是1  fpr, tpr, thresholds = roc\_curve(y\_true = y, y\_score = scores, pos\_label=1)  print("tpr=", tpr)  print("fpr=", fpr)  print("thresholds=", thresholds)  # 计算auc值  roc\_auc = auc(fpr, tpr)  plt.figure()  lw = 2  plt.plot(fpr, tpr, color='darkorange',lw=lw, label='ROC curve (area = %.2f)' % roc\_auc)  plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')  plt.xlim([0.0, 1.05])  plt.ylim([0.0, 1.05])  plt.xlabel('False Positive Rate')  plt.ylabel('True Positive Rate')  plt.title('Receiver operating characteristic example')  plt.legend(loc="lower right")  plt.show() |

问题：在机器学习中AUC和accuracy有什么内在关系？

auc代表的是分类或者排序能力，与分类阈值无关；准确率是和阈值有关的。auc高，准确率低，可能的原因是分类阈值的选择引起的。极端来讲，默认阈值是0.5，但模型输出的值全部小于0.5，那正样本是全错的；但由于auc很高，正负样本还是可以分开，把分类阈值调小即可。auc低，准确率高，这种一般发生在分布不平衡的问题中。比例少的那类分类错误多，但由于数量少整体的准确率还是很高，但代表分类能力的auc就会很低。参考：<https://www.zhihu.com/question/313042288>

## 简单示例

### 英文新闻文本分类

|  |
| --- |
| #coding:utf-8  """  朴素贝叶斯示例-英文新闻文本分类  """  from sklearn.datasets import fetch\_20newsgroups  **from sklearn.model\_selection import train\_test\_split**  from sklearn.naive\_bayes import MultinomialNB,BernoulliNB  **from sklearn.metrics import classification\_report, confusion\_matrix**  from sklearn.feature\_extraction.text import TfidfVectorizer  ## 导入数据集以及数据集的简单处理  # 导入全部的训练集和测试集  news = fetch\_20newsgroups(subset="all")  # 打印类目名称列表  print("类目名称列表\n",u'\n'.join(news.target\_names))  # 打印类目数量  print("类目数量\n", len(news.target\_names))  # 打印数据X量  print("训练集文本数量\n", len(news.data))  # 打印类标Y量  print("标记了类别的文本数量\n", len(news.target))  # 打印第0篇文本  print("第1篇文本内容\n", news.data[0])  # 打印类目序号  print("第1篇文本的类别序号\n", news.target[0])  # 打印类目序号所对应的类目名称  print("第1篇文本的类别名称\n", news.target\_names[news.target[0]])  **# 数据集切分**  **x\_train, x\_test, y\_train, y\_test = train\_test\_split(news.data, news.target)**  # 向量化  vectorizer = TfidfVectorizer()  X\_train = vectorizer.fit\_transform(x\_train)  X\_test = vectorizer.transform(x\_test)  print("="\*50)  # 构建多项式朴素贝叶斯实例  mul\_nb = MultinomialNB()  # 训练模型  mul\_nb.fit(X\_train, y\_train)  **# 打印测试集上的分类报告**  **print("分类报告\n", y\_test, classification\_report(mul\_nb.predict(X\_test)))**  **# 打印测试集上的混淆矩阵**  **print("混淆矩阵\n", y\_test, confusion\_matrix(mul\_nb.predict(X\_test)))** |

tf值：词语在某篇文本中的出现次数

idf值：不加平滑 1+log(训练文本数/包含词语的文本数)

加平滑 1+log(训练文本数+1/包含词语的文本数+1)

tf\_idf值：tf值\*idf值

### 英文影评情感分类

|  |
| --- |
| #coding:utf-8  """  朴素贝叶斯影评情感分析  """  from nltk.corpus import movie\_reviews  **from sklearn.model\_selection import StratifiedShuffleSplit**  import nltk  from nltk.corpus import stopwords  from nltk.collocations import BigramCollocationFinder  from nltk.metrics import BigramAssocMeasures  def get\_data():  """  获取影评数据  """  dataset = []  y\_labels = []  # 遍历类别  for cat in movie\_reviews.categories():  # 遍历每个类目的评论id  for fileid in movie\_reviews.fileids(cat):  # 读取评论词语列表  words = list(movie\_reviews.words(fileid))  dataset.append((words, cat))  y\_labels.append(cat)  return dataset, y\_labels  def get\_train\_test(input\_dataset, ylabels):  """  划分数据为训练集和测试集  """  **train\_size = 0.7**  **test\_size = 1-train\_size**  **stratified\_split = StratifiedShuffleSplit(n\_splits=10, test\_size=test\_size, random\_state=77)**  **for train\_indx, test\_indx in stratified\_split.split(input\_dataset, ylabels):**  **train = [input\_dataset[i] for i in train\_indx]**  **train\_y = [ylabels[i] for i in train\_indx]**  **test = [input\_dataset[i] for i in test\_indx]**  **test\_y = [ylabels[i] for i in test\_indx]**  return train, test, train\_y, test\_y  def build\_word\_features(instance):  """  构建特征词典  one-hot, 特征名称为词语本身，特征值为bool类型值  """  # 存储特征的词典  feature\_set = {}  # instance的第1个元素为词语列表  words = instance[0]  # 填充特征词典  for word in words:  feature\_set[word] = 1  # instance的第2个元素为类别名称  return feature\_set, instance[1]  def build\_negate\_features(instance):  """  如果一个词语前有否定关键词(not或no)修饰，则对词语加前缀Not\_, 否定关键词不再被添加到特征词典  """  # Retreive words, first item in instance tuple  words = instance[0]  final\_words = []  # A boolean variable to track if the  # previous word is a negation word  negate = False  # List of negation words  negate\_words = ['no', 'not']  # On looping throught the words, on encountering  # a negation word, variable negate is set to True  # negation word is not added to feature dictionary  # if negate variable is set to true  # 'Not\_' prefix is added to the word  for word in words:  if negate:  word = 'Not\_' + word  negate = False  if word not in negate\_words:  final\_words.append(word)  else:  negate = True  # Feature dictionary  feature\_set = {}  for word in final\_words:  feature\_set[word] = 1  return feature\_set, instance[1]  def remove\_stop\_words(in\_data):  """  去除停用词  Utility function to remove stop words  from the given list of words  """  stopword\_list = stopwords.words('english')  negate\_words = ['no', 'not']  # We dont want to remove the negate words  # Hence we create a new stop word list excluding  # the negate words  new\_stopwords = [word for word in stopword\_list if word not in negate\_words]  label = in\_data[1]  # Remove stopw words  words = [word for word in in\_data[0] if word not in new\_stopwords]  return words, label  def build\_keyphrase\_features(instance):  """  构建短语特征  """  feature\_set = {}  # 应用map迭代器  instance = remove\_stop\_words(instance)  words = instance[0]  # 使用nltk.collocations的BigramCollocationFinder  bigram\_finder = BigramCollocationFinder.from\_words(words)  # 2grams按词频降序排列，前400个作为关键短语抽取  bigrams = bigram\_finder.nbest(BigramAssocMeasures.raw\_freq, 400)  for bigram in bigrams:  feature\_set[bigram] = 1  return feature\_set, instance[1]  def build\_model(features):  """  用给定特征集构建朴素贝叶斯模型（NLTK的朴素贝叶斯分类器）  """  model = nltk.NaiveBayesClassifier.train(features)  return model  def probe\_model(model, features, dataset\_type='Train'):  """  计算测试集准确率, nltk新版里已经没有nltk.classify.accuracy()方法，  所以这里自己编写precision值  """  right\_cnt = 0  sum\_cnt = 0  for feature in features:  if model.classify(feature[0]) == feature[1]:  right\_cnt += 1  sum\_cnt += 1  if sum\_cnt > 0:  accuracy = right\_cnt \* 100.0 / sum\_cnt  print("\n" + dataset\_type + " Accuracy = %0.2f" % accuracy + "%")  def show\_features(model, no\_features=5):  """  显示对分类有帮助的特征（NLTK中显示显著特征的方法）  """  print("\nFeature Importance")  print("===================\n")  print(model.show\_most\_informative\_features(no\_features))  def build\_model\_cycle\_1(train\_data, dev\_data):  """  用build\_word\_features构建特征训练模型  """  # Build features for training set  train\_features = map(build\_word\_features, train\_data)  # Build features for test set  dev\_features = map(build\_word\_features, dev\_data)  # Build model  model = build\_model(train\_features)  # Look at the model Python3的map返回的是迭代器，而不是列表，所以在训练使用后想再使用，需要再调用一次  train\_features = map(build\_word\_features, train\_data)  print("\n词语特征训练集准确率", end='')  probe\_model(model, train\_features)  print("词语特征验证集准确率", end='')  probe\_model(model, dev\_features, 'Dev')  return model  def build\_model\_cycle\_2(train\_data, dev\_data):  """  用build\_negate\_features构建特征训练模型  """  # Build features for training set  train\_features = map(build\_negate\_features,train\_data)  # Build features for test set  dev\_features = map(build\_negate\_features,dev\_data)  # Build model  model = build\_model(train\_features)  # Look at the model  train\_features = map(build\_negate\_features, train\_data)  print("\n否定词修饰特征训练集准确率", end='')  probe\_model(model, train\_features)  print("否定词修饰特征验证集准确率", end='')  probe\_model(model, dev\_features,'Dev')    return model    def build\_model\_cycle\_3(train\_data, dev\_data):  """  用build\_keyphrase\_features构建特征训练模型  """    # Build features for training set  train\_features = map(build\_keyphrase\_features, train\_data)  # Build features for test set  dev\_features = map(build\_keyphrase\_features, dev\_data)  # Build model  model = build\_model(train\_features)  # Look at the model  train\_features = map(build\_keyphrase\_features, train\_data)  print("\n2gram特征训练集准确率", end='')  probe\_model(model, train\_features)  print("2gram特征验证集准确率", end='')  probe\_model(model, dev\_features, 'Dev')  test\_features = map(build\_keyphrase\_features, test\_data)  print("2gram特征测试集准确率", end='')  probe\_model(model, test\_features, 'Test')  return model  if \_\_name\_\_ == "\_\_main\_\_":    # Load data  input\_dataset, y\_labels = get\_data()  # Train data  train\_data, all\_test\_data, train\_y, all\_test\_y = get\_train\_test(input\_dataset, y\_labels)  # Dev data  dev\_data, test\_data, dev\_y, test\_y = get\_train\_test(all\_test\_data, all\_test\_y)  # Let us look at the data size in our different  # datasets  print("\nOriginal Data Size =", len(input\_dataset))  print("\nTraining Data Size =", len(train\_data))  print("\nDev Data Size =", len(dev\_data))  print("\nTesting Data Size =", len(test\_data))  # 用词语特征训练验证模型  model\_cycle\_1 = build\_model\_cycle\_1(train\_data, dev\_data)  # 打印显著特征  print("词语显著特征", end='')  show\_features(model\_cycle\_1)  # 用否定词修饰的词语特征训练验证模型  model\_cycle\_2 = build\_model\_cycle\_2(train\_data, dev\_data)  # 打印显著特征  print("否定词修饰显著特征", end='')  show\_features(model\_cycle\_2)  # 用2gram搭配特征训练验证模型  model\_cycle\_3 = build\_model\_cycle\_3(train\_data, dev\_data)  # 打印显著特征  print("2gram显著特征", end='')  show\_features(model\_cycle\_3) |

### 英文垃圾邮件分类

|  |
| --- |
| # coding:utf-8  """  朴素贝叶斯垃圾邮件检测  """  import numpy as np  import csv  import nltk  from nltk.corpus import stopwords  from nltk.stem import WordNetLemmatizer  import csv  import codecs  # 文本预处理  def preprocessing(text):  # 分词  tokens = [word for sent in nltk.sent\_tokenize(text) for word in nltk.word\_tokenize(sent)]  # 去除停用词  stop = stopwords.words('english')  tokens = [token for token in tokens if token not in stop]  # 移除少于3个字母的单词  tokens = [word for word in tokens if len(word) >= 3]  # 大写字母转小写  tokens = [word.lower() for word in tokens]  # 词干还原  lmtzr = WordNetLemmatizer()  tokens = [lmtzr.lemmatize(word) for word in tokens]  preprocessed\_text = ' '.join(tokens)  return preprocessed\_text  def modelbuilding(sms\_data, sms\_labels):  """  构建分类器的流水线示例  1. 构建训练集和测试集  2. TFIDF向量化器  3. 构建朴素贝叶斯模型  4. 打印准确率和其他评测方法  5. 打印最相关特征  :param sms\_data:  :param sms\_labels:  :return:  """  # 构建训练集和测试集步骤  trainset\_size = int(round(len(sms\_data) \* 0.70))  # 我选择70：30的比例  print('训练集大小： ' + str(trainset\_size) + '\n')  x\_train = np.array([''.join(el) for el in sms\_data[0:trainset\_size]])  y\_train = np.array([el for el in sms\_labels[0:trainset\_size]])  x\_test = np.array([''.join(el) for el in sms\_data[trainset\_size + 1:len(sms\_data)]])  y\_test = np.array([el for el in sms\_labels[trainset\_size + 1:len(sms\_labels)]])  # We are building a TFIDF vectorizer here  from sklearn.feature\_extraction.text import TfidfVectorizer  vectorizer = TfidfVectorizer(min\_df=2, ngram\_range=(1, 2), stop\_words='english', strip\_accents='unicode', norm='l2')  X\_train = vectorizer.fit\_transform(x\_train)  X\_test = vectorizer.transform(x\_test)  from sklearn.naive\_bayes import MultinomialNB  clf = MultinomialNB().fit(X\_train, y\_train)  y\_nb\_predicted = clf.predict(X\_test)  print(y\_nb\_predicted)  # 输出测试集上的混淆矩阵  **from sklearn.metrics import confusion\_matrix**  print(' \n 混淆矩阵 \n ')  **cm = confusion\_matrix(y\_test, y\_nb\_predicted)**  print(cm)  # 输出测试集上的分类结果报告  **from sklearn.metrics import classification\_report**  print('\n 分类报告')  **print(classification\_report(y\_test, y\_nb\_predicted))**  # 输出正负类的前10重要特征  print("正负类前10重要特征")  coefs = clf.coef\_  intercept = clf.intercept\_  feature\_names = vectorizer.get\_feature\_names()  coefs\_with\_fns = sorted(zip(clf.coef\_[0], feature\_names))  n = 10  top = zip(coefs\_with\_fns[:n], coefs\_with\_fns[:-(n + 1):-1])  for (coef\_1, fn\_1), (coef\_2, fn\_2) in top:  print('\t%.4f\t%-15s\t\t%.4f\t%-15s' % (coef\_1, fn\_1, coef\_2, fn\_2))  if \_\_name\_\_ == '\_\_main\_\_':  sms\_data = []  sms\_labels = []  with codecs.open('data/SMSSpamCollection', 'rb', 'utf8', 'ignore') as infile:  csv\_reader = csv.reader(infile, delimiter='\t')  for line in csv\_reader:  # 添加类别标记  sms\_labels.append(line[0])  # 添加预处理后的文本  sms\_data.append(preprocessing(line[1]))  print("原始数据大小：", len(sms\_data))  print("原始标签大小：", len(sms\_labels))  # 我们正则调用model构建函数  modelbuilding(sms\_data, sms\_labels) |

# 文本聚类技术

与文本相关的另一个问题簇系是无监督式分类问题。关于这类问题，最常见的一种问题描述是“我手里有数以百万计的（非结构化）文档，是否能找到一种方式将它们分组，以便赋予其有意义的类别？”在这里，需要使用无监督的方式来对这些文本文档进行分组。文本聚类法（有时也叫聚类法）是目前最为常见的无监督式分组方式之一。从历年论文文献上来看，基本没有介绍文本聚类技术的研究综述类文献，说明文本聚类技术本身并没有太多可作文章的点，或者说它本身并不复杂。文本聚类技术可以应用的场景有

1. **语料提纯**。在准备好文本分类训练语料后，当我们用任意一种特征选择的方法（）选出了类目关键词后，有时候可以发现一个类目下含有不属于该类目的关键词，说明这个类目中可能含有不属于该类目的文本，这时可以对该类目的文本做一次聚类，再对每个簇抽取关键词，此时借助这些关键词中可以比较容易地发现不属于该类目地1个或多个文本，剔除这些文本或者把他们放到属于他们的类目中，此时这个类目就较之前纯净了一些。
2. 在没有类目体系和关键词的情况下，可以首先对文本做聚类，然后提取每个簇的关键特征词，将这些词作为类目关键词，进而**确定类目体系**。
3. 在已知类目体系和关键词但是没有训练语料的时候，可以首先对这些类目关键词做聚类，一方面可以验证类目之间是否具有语义上的差异，另一方面可以用多组关键词来表示一个类目，这样可以在细粒度层面来表示一个类目，之后便可以通过关键词匹配来确定文本类别，或者计算待标注文本与各类目相似度来预测文本类别，从而实现**自动构建文本分类语料库**。

## 数据规范化

### 中心化变换

中心化变换是对矩阵进行坐标轴平移处理方法。先求出每列的平均值，再将每列的各行数据都减去该列的平均值。

其中，，为矩阵中的原始值。

### 极差正规化变换

极差变换是找出每一个特征在所有文本中的最大值和最小值，这两者之差称为极差，然后使用每一个特征的每一个原始数据减去该特征分布的极小值，再除以极差，就得到变换后的数据。经过变换后，数据矩阵中每列即每个特征的最大数值为1，最小数值为0，其余数据取值均为0～1；并且变换后的数据都不再具有量纲，即不再具有物理意义。

其中，。

### 极差标准化变换

极差标准变换同样先求出每个特征在所有文本中的极差，然后使用每个特征的每一个原始数据减去该特征分布中的平均值，再除以极差，就得到变换后的数据。经变换后，各个特征的均值为0，极差均为1，数据也不再具有量纲。

其中，，。

### 标准化变换

标准化变换要求先求出每个特征在所有文本中的均值及标准差，然后使用该特征的原始数据减去均值，并除以标准差。经过标准化变换处理后，每个特征即数据矩阵中每列数据的平均值为0，方差为1，且也不再具有量纲。

其中，，。

### 向量归一化

向量归一化是将每个对象的p个特征视为该对象在p维向量空间中的值，使用原始数据除以该对象p个特征值的平方和开方后的值，即为归一化后的数据。中心化变换、极差正规化变换、极差标准化变换与标准化变换都是从特征出发，对数据进行变换。向量归一化并不是从特征出发，而是从文本出发，对文本中的各个特征值进行归一化处理。将原来由若干个特征组成的空间向量转换为单位向量（变换后数据矩阵中的每行都是单位向量），即每个特征值均变换为在(0,1)之间。

其中，i表示第i篇文档序号，j表示第j个特征。

实践中使用最多的是向量归一化，因为其他4种方法都是对整个文本集按特征变换，在内存空间有限的情况下，当文本数量很大或者词语特征很多时，不可能将整个文本集读入内存，再进行变换。但是向量归一化是对单一文本进行的，因此可以只加载一篇文本向量便可以进行归一化处理。

示例

|  |
| --- |
| In [**1**]: **import** **numpy** **as** **np**  In [**2**]: a = np.array([0,3,4])  In [**3**]: np.linalg.norm(a)  Out[**3**]: 5.0  In [**4**]: a\*(1.0/np.linalg.norm(a))  Out[**4**]: array([0. , 0.6, 0.8]) |

在sklearn中的TfidfVectorizer类的参数norm=’l2’时，就是用该方法对向量进行规范化。

### 矩阵归一化

有的时候我们需要对矩阵中的每个行（列）向量进行归一化。

|  |
| --- |
| In [35]: a  Out[35]:  array([[1, 1, 1],  [3, 4, 0],  [1, 1, 0]])  **# 对每个行向量进行归一化**  **In [37]: b = (a.T/np.linalg.norm(a,axis=1)).T**  In [38]: b  Out[38]:  array([[0.57735027, 0.57735027, 0.57735027],  [0.6 , 0.8 , 0. ],  [0.70710678, 0.70710678, 0. ]])  In [46]: b[0]  Out[46]: array([0.57735027, 0.57735027, 0.57735027])  In [39]: np.linalg.norm(b[0])  Out[39]: 1.0  **# 对每个列向量进行归一化**  **In [40]: c = (a/np.linalg.norm(a,axis=0))**  In [41]: c  Out[41]:  array([[0.30151134, 0.23570226, 1. ],  [0.90453403, 0.94280904, 0. ],  [0.30151134, 0.23570226, 0. ]])  In [43]: c[:,0]  Out[43]: array([0.30151134, 0.90453403, 0.30151134])  In [44]: np.linalg.norm(c[:,0])  Out[44]: 1.0 |

## 相似性度量

### 明氏距离

明氏距离不是一种距离，而是一组距离，它的定义式为

当p=1时，就是曼哈顿距离，当p=2时，就是欧式距离，当p->时，就是切比雪夫距离。明氏距离有两个缺陷（1）将各个分量的量纲也就是单位，当作相同的看待。（2）没有考虑各个分量的分布（期望、方差等）可能是不同的。

### 曼哈顿距离

示例

|  |
| --- |
| In [9]: vector1  Out[9]: array([1, 2, 3])  In [10]: vector2  Out[10]: array([4, 5, 6])  In [6]: np.linalg.norm(vector1-vector2, ord=1)  Out[6]: 9.0 |

### 欧式距离

示例

|  |
| --- |
| In [**9**]: vector1  Out[**9**]: array([1, 2, 3])  In [**10**]: vector2  Out[**10**]: array([4, 5, 6])  In [**7**]: np.linalg.norm(vector1-vector2)  Out[**7**]: 5.196152422706632 |

### 切比雪夫距离

示例

|  |
| --- |
| In [**9**]: vector1  Out[**9**]: array([1, 2, 3])  In [**12**]: vector2  Out[**12**]: array([4, 7, 5])  In [**13**]: np.linalg.norm(vector1-vector2, ord=np.inf)  Out[**13**]: 5.0 |

### 相关系数

相关系数是衡量随机变量A与B相关程度的一种方法，相关系数的取值范围是[-1,1]。相关系数的绝对值越大，则表明A与B相关程度越高。当A与B线性相关时，相关系数取值为1（正线性相关）或-1（负线性相关）。

示例

|  |
| --- |
| In [**23**]: vector1  Out[**23**]: array([1, 2, 4])  In [**24**]: vector2  Out[**24**]: array([-2, -4, -8])  In [**22**]: np.corrcoef(vector1,vector2)  Out[**22**]:  array([[ 1., -1.],  [-1., 1.]])  In [**26**]: vector1  Out[**26**]: array([1, 2, 4])  In [**27**]: vector2  Out[**27**]: array([2, 4, 8])  In [**28**]: np.corrcoef(vector1, vector2)  Out[**28**]:  array([[1., 1.],  [1., 1.]]) |

### 余弦相似度

几何中夹角余弦用来衡量两个向量方向的差异，借用这一概念来衡量样本向量之间的差异，定义为

需要**注意**在计算行向量a与矩阵B的每个列向量的余弦相似度时，我们通常使用行向量乘以矩阵，即aB=c，此时向量c就是向量a与矩阵B各列向量计算相似度的结果列表。在用numpy实现aB之前，注意对向量a以及矩阵B的各列向量进行单位化。

|  |
| --- |
| In [59]: np.arange(1,-1)  Out[59]: array([], dtype=int32)  In [60]: vector1 = np.array([1,2,3])  In [61]: vector2 = np.array([4,7,5])  In [62]: np.dot(vector1,vector2)/(np.linalg.norm(vector1)\*np.linalg.norm(vector2))  Out[62]: 0.9296696802013682 |

## 聚类算法

### K-Means

该方法非常直观，从其名称就可以看出它需要试着找出k组围绕着若干数据点的平均值。因此，该算法首先要拾取一些数据点来充当所有数据点的中心。接下来，该算法会将所有数据点各自分配给离其最近的那个中心。在这过程中，每完成一次迭代，其中心就要重新计算一次，然后继续迭代，直到达到中心不再变化的状态（即达到算法饱和）。

算法过程描述如下

1. 先随机选取K个元组作为中心。
2. 计算每个实例点到中心的距离（欧式距离）K\*实例数量次，分配每个实例点到最近的中心。
3. 更新中心。每个簇中所有实例点各维度的均值。
4. 重复（2）（3）直到中心不再变化或迭代次数已到达。

K-Means算法的缺陷有

1. 对k个初始质心的选择比较敏感，容易陷入局部最小值。例如算法运行多次，有可能会得到不同的结果。解决该问题的方法是，使用多次的随机初始化，计算每一次建模得到的代价函数的值，选取代价函数最小结果作为聚类结果。代价函数为：

其中m表示样本的数量，表示样本i，表示样本i所属簇的质心。

1. k的取值会影响聚类质量。此时可以使用肘部法则，也就是说选择不同的k值进行聚类并记录代价，最后选择处于肘部的代价值所对应的k值作为簇的个数。当肘部不存在时，需要根据经验进行选择。
2. 非球形簇无法使用K-Means算法，此时可以使用密度聚类。

### MiniBatches

Mini Batch K-Means算法是K-Means算法的变种，采用小批量的数据子集减小计算时间。这里所谓的小批量是指每次训练算法时所随机抽取的数据子集，采用这些随机产生的子集进行算法训练，大大减小了计算时间，结果一般只略差于标准算法。K-Means算法一般适用于已知簇数量，数据量小于1万条的情境，Mini Batch K-Means算法适用于已知簇数量，数据量大于等于1万条的情境。Mini Batch K-Means算法的迭代步骤是

1. 从数据集中随机抽取一些数据形成小批量，把他们分配给最近的质心。
2. 更新质心，与K-Means相比，数据的更新是在每一个小的样本集上。

### Single-Pass

话题发现与跟踪（topic detection and tracking, TDT）的评测中常用的聚类方法是single-pass聚类，其原理简单、计算速度快，然而该算法的缺点也很明显：受输入顺序的影响，且聚类结果精度差。single-pass聚类的基本流程如下：

1. 接收一篇互联网文本向量d；
2. d逐一与已有的话题中各报道进行相似度计算，并取最大者作为与该话题的相似度（single-link策略）；
3. 在所有话题间选出与d相似度最大的一个，以及此时的相似度值；
4. 如果相似度大于阈值TC，d所对应的互联网文本被分配给这个话题模型文本类，跳至（6）；
5. 如果相似度值小于阈值TC，d所对应的文本不属于已有的话题，创建新话题，同时把这篇文本归属创建的新话题模型文本类；
6. 本次聚类结束，等待文本到来。

有两篇single-pass算法改进的文章。殷风景2011年提出了ICIT算法。改进方面有（1）词频统计针对具有实际意义的名词和动词，避免文本向量维度太高。（2）两篇文本的相似度=0.7\*标题相似度+0.3\*正文相似度，考虑了标题对于文本主题的概括性。（3）到达数据按代添加到聚类过程中，每一代包含200条数据，先在本代成员之间进行初步的相似度比较和聚类，再将这些初步类与已有话题进行比较和聚合，避免因数据到达顺序不同而使聚类结果有变化。（4）相似度计算采用了average-link，准确度更佳，有效减少大类出现。（5）在当前代内完成聚类后加入一个比较调整的步骤，代内成员依次计算当前聚类结果下最相似的类簇是否就是自己所处的簇，不是则调整。陶舒怡2014年提出基于簇相合性的文本增量聚类算法。该算法的改进措施有（1）基于词项语义相似度的文本表示模型。（2）计算新增文本与已有簇的相合性实现增量聚类，它不仅计算了文本与簇的相似度，而且考虑了簇分布特征。（3）增量处理完成后对错分可能性大的文本重新指派类别。

## 简单示例

### K-Means示例

|  |
| --- |
| #coding:utf-8  """  Kmeans算法聚类文本示例  """  import matplotlib.pyplot as plt  import numpy as np  # 加载文本数据  from time import time  from sklearn.datasets import load\_files  print("loading documents ...")  t = time()  docs = load\_files('datasets/clustering/data')  print("summary: {0} documents in {1} categories.".format(len(docs.data), len(docs.target\_names)))  print("done in {0} seconds".format(time() - t))  # 文本向量化表示  from sklearn.feature\_extraction.text import TfidfVectorizer  max\_features = 20000  print("vectorizing documents ...")  t = time()  vectorizer = TfidfVectorizer(max\_df=0.4, min\_df=2, max\_features=max\_features, encoding='latin-1')  X = vectorizer.fit\_transform((d for d in docs.data))  print("n\_samples: %d, n\_features: %d" % X.shape)  print("number of non-zero features in sample [{0}]: {1}".format(docs.filenames[0], X[0].getnnz()))  print("done in {0} seconds".format(time() - t))  # 文本聚类  from sklearn.cluster import Kmeans, MiniBatchKMeans  print("clustering documents ...")  t = time()  n\_clusters = 4  kmean = KMeans(n\_clusters=n\_clusters, max\_iter=100, tol=0.01, verbose=1, n\_init=3)  kmean.fit(X)  print("kmean: k={}, cost={}".format(n\_clusters, int(kmean.inertia\_)))  print("done in {0} seconds".format(time() - t))  # 打印实例数量  print(len(kmean.labels\_))  # 打印实例1000到1009的簇号  print(kmean.labels\_[1000:1010])  # 打印实例1000到1009的文件名  print(docs.filenames[1000:1010])  # 打印每个簇的前10个显著特征  print("Top terms per cluster:")  order\_centroids = kmean.cluster\_centers\_.argsort()[:, ::-1]  terms = vectorizer.get\_feature\_names()  for i in range(n\_clusters):  print("Cluster %d:" % i, end='')  for ind in order\_centroids[i, :10]:  print(' %s' % terms[ind], end='')  print() |

### Single-Pass示例

|  |
| --- |
| #coding:utf-8  """  single-pass增量聚类演示  """  import numpy as np  from sklearn.datasets import load\_files  from pyhanlp import \*  import re  import codecs  NotionalTokenizer = JClass("com.hankcs.hanlp.tokenizer.NotionalTokenizer")  # 以文本在文本集中的顺序列出的文本向量矩阵（用300维向量表示）  text\_vec = None  # 以文本在文本集中的顺序列出的话题序号列表  topic\_serial = None  # 话题数量  topic\_cnt = 0  # 加载词语向量词典  word\_dict = dict()  with codecs.open('dictionary/cc.zh.300.vec', 'rb', 'utf-8', 'ignore') as infile:  infile.readline()  for line in infile:  line = line.strip()  if line:  items\_li = line.split()  word = items\_li[0]  word\_vec = np.array([float(w) for w in items\_li[1::]])  word\_dict[word] = word\_vec  print("load cc.zh.300.vec len = %d" % len(word\_dict))  # 仅保留中文字符  def translate(text):  p2 = re.compile(u'[^\u4e00-\u9fa5]') # 中文的编码范围是：\u4e00到\u9fa5  zh = " ".join(p2.split(text)).strip()  zh = ",".join(zh.split())  res\_str = zh # 经过相关处理后得到中文的文本  return res\_str  # 预处理，实词分词器分词，查询词语向量，并返回文本向量  def preprocess(text):  sen\_vec = np.zeros((1, 300))  # 去掉非中文字符  text = translate(text)  # 将\r\n替换为空格  text = re.sub(u'[\r\n]+', u' ', text)  # 分词与词性标注，使用实词分词器  word\_li = NotionalTokenizer.segment(text)  word\_li = [w.word for w in word\_li]  # 去掉单字词  word\_li = [w for w in word\_li if len(w)>1]  # 查询每个词语的fasttext向量，计算句子向量  valid\_word\_cnt = 0  for word in word\_li:  if word in word\_dict:  sen\_vec += word\_dict[word]  valid\_word\_cnt += 1  if valid\_word\_cnt > 0:  sen\_vec = sen\_vec\*(1.0/valid\_word\_cnt)  # 单位化句子向量  sen\_vec = sen\_vec\*(1.0/np.linalg.norm(sen\_vec))  return text, sen\_vec  # single-pass  def single\_pass(sen\_vec, sim\_threshold):  global text\_vec  global topic\_serial  global topic\_cnt  if topic\_cnt == 0: # 第1次送入的文本  # 添加文本向量  text\_vec = sen\_vec  # 话题数量+1  topic\_cnt += 1  # 分配话题编号，话题编号从1开始  topic\_serial = [topic\_cnt]  else: # 第2次及之后送入的文本  # 文本逐一与已有的话题中的各文本进行相似度计算  sim\_vec = np.dot(sen\_vec, text\_vec.T)  # 获取最大相似度值  max\_value = np.max(sim\_vec)  # 获取最大相似度值的文本所对应的话题编号  topic\_ser = topic\_serial[np.argmax(sim\_vec)]  print("topic\_ser", topic\_ser, "max\_value", max\_value)  # 添加文本向量  text\_vec = np.vstack([text\_vec, sen\_vec])  # 分配话题编号  if max\_value >= sim\_threshold:  # 将文本聚合到该最大相似度的话题中  topic\_serial.append(topic\_ser)  else:  # 否则新建话题，将文本聚合到该话题中  # 话题数量+1  topic\_cnt += 1  # 将新增的话题编号（也就是增加话题后的话题数量）分配给当前文本  topic\_serial.append(topic\_cnt)  def main():  # 加载数据  data\_all = load\_files(container\_path=r'data/news', categories=u'Sports',  encoding=u'gbk', decode\_error=u'ignore')  # 获取文本数据集  data = data\_all.data  # 预处理后的文本数据集  preprocessed\_data = []  # 进行增量聚类  for text in data:  text, text\_vec = preprocess(text)  single\_pass(text\_vec, 0.9)  preprocessed\_data.append(text)  # 输出聚类结果  with open('res\_single\_pass.txt', 'wb') as outfile:  sorted\_text = sorted(zip(topic\_serial, preprocessed\_data), key=lambda x:x[0])  for topic\_ser, text in sorted\_text:  out\_str = u'%d\t%s\n' % (topic\_ser, text)  outfile.write(out\_str.encode('utf-8', 'ignore'))  print("program finished")  # 在mac下释放向量内存时间较长，可以直接ctrl+c强制退出程序  if \_\_name\_\_ == '\_\_main\_\_':  main() |

# 特征工程

## 文本表示

### 词袋模型

bag-of-words(BOW model)最早出现在自然语言处理(Natual Language Processing)和信息检索(Information Retrieval)领域。该模型**忽略掉文本的语法和语序**等要素，将其仅仅看作是若干个词汇的集合，文档中的每个单词的出现都是独立的。BOW使用一组无序的单词(words)来表达一段文字或一个文档。

例：给定下边两个文档：

John likes to watch movies. Mary likes too.

John also likes to watch football games.

用词袋模型表示。

构建词典(dictionary)

{“John”:1, “likes”:2, “to”:3, “watch”:4, “movies”, 5, “also”:6, “football”:7, “games”:8, “Mary”:9, “too”:10}

词袋表示文本1：[1,2,1,1,1,0,0,0,1,1]

词袋表示文本2：[1,1,1,1,0,1,1,1,0,0]

|  |
| --- |
| 为什么要用词袋模型表示文本？  文本的长度有长有短，为了用机器学习学习，需要文本的表示长度一样，因此在使用了词袋模型表示后，可以将不同长度的文本转成相同长度(词典的长度)的向量。 |

在sklearn中可以使用CountVectorizer方法构建单词的字典，每个单词实例被转成为特征向量的一个数值特征，每个元素是特定单词在文本中出现的次数。

Python示例

|  |
| --- |
| ## 词袋模型  In [1]: from sklearn.feature\_extraction.text import CountVectorizer  # 文本  In [2]: texts = ["dog cat fish", "dog cat cat", "fish bird", "bird"]  # 实例化计数向量器  In [3]: cv = CountVectorizer()  # 统计文本集  In [4]: cv\_fit = cv.fit\_transform(texts)  # 获取文本集词汇表  In [5]: cv.get\_feature\_names()  Out[5]: ['bird', 'cat', 'dog', 'fish']  # 输出文本集矩阵，行表示文本，列依次表示上一步词汇表中的词语  In [6]: cv\_fit.toarray()  Out[6]:  array([[0, 1, 1, 1],  [0, 2, 1, 0],  [1, 0, 0, 1],  [1, 0, 0, 0]], dtype=int64)  # 计算每个特征词语在整个文本集中频次  In [7]: cv\_fit.toarray().sum(axis=0)  Out[7]: array([2, 3, 2, 2], dtype=int64) |

### TF-IDF

TF-IDF又可以用在文本的关键词抽取上，也就是下一节中介绍的特征选择的一种方法。它的核心思想是一篇文本中出现频率高的词认为是重要的词，这里需要注意的是，在统计前需要去除掉文本中的停用词。另一方面对于在一篇文本频率相同的词，如果词A在多篇文本中出现，而词B仅在很少的几篇文本中出现，则认为B比A更重要。用统计学语言表达，就是在词频(TF，文档频率)的基础上，要对每个词分配一个“重要性”权重。最常见的词（“的”、“是”、“在”）给予最小的权重，较常见的词（“中国”）给予较小的权重，较少见的词（“蜜蜂”、“养殖”）给予较大的权重。这个权重叫做“逆文档频率”(Inverse Document Frequency，缩写为IDF)，它的大小与一个词的常见程度成反比。TF-IDF最大的缺陷是向量的维度高，其实所有词袋模型都有这个严重缺陷。维度高造成了后续相似度或者文本分类的计算量非常大，同时数据稀疏也导致了相似度区分不明显。

TF值的计算方法：

IDF值的计算方法：

TF-IDF值的计算方法：

在sklearn中TfidfVectorizer使用了一个高级的计算方法，称为Term Frequence Inverse Document Frequence(TF-IDF)。这是一个衡量一个词在文本或语料中重要性的统计方法。直觉上讲，该方法通过比较整个语料库的词的频率，寻求在当前文档中频率较高的词。这是一种将结果进行标准化的方法，可以**避免因为有些词出现太过频繁而对一个实例的特征化作用不大的情况**。

Python示例

|  |
| --- |
| ## TF-IDF  In [8]: from sklearn.feature\_extraction.text import TfidfVectorizer  # 文档集  In [9]: texts = ["The quick brown fox jumped over the lazy dog.",  ...: "The dog.",  ...: "The fox."]  # 实例化向量化器  In [10]: vectorizer = TfidfVectorizer()  # 词条话文档集以及创建词汇表  In [11]: vectorizer.fit(texts)  Out[11]:  TfidfVectorizer(analyzer='word', binary=False, decode\_error='strict',  dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',  lowercase=True, max\_df=1.0, max\_features=None, min\_df=1,  ngram\_range=(1, 1), norm='l2', preprocessor=None, smooth\_idf=True,  stop\_words=None, strip\_accents=None, sublinear\_tf=False,  token\_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use\_idf=True,  vocabulary=None)  # 输出词汇表  In [12]: vectorizer.vocabulary\_  Out[12]:  {'brown': 0,  'dog': 1,  'fox': 2,  'jumped': 3,  'lazy': 4,  'over': 5,  'quick': 6,  'the': 7}  # 输出词汇表中每个词对应的idf值  In [13]: vectorizer.idf\_  Out[13]:  array([1.69314718, 1.28768207, 1.28768207, 1.69314718, 1.69314718,  1.69314718, 1.69314718, 1. ])  # 向量化一篇文本  In [14]: vector = vectorizer.transform([texts[0]])  # 输出一篇文本向量的长度  In [15]: vector.shape  Out[15]: (1, 8)  # 输出一篇文本向量  In [16]: vector.toarray()  Out[16]:  array([[0.36388646, 0.27674503, 0.27674503, 0.36388646, 0.36388646,  0.36388646, 0.36388646, 0.42983441]]) |

## 特征选择

### 互信息

两个随机变量X、Y的互信息定义为X、Y的联合概率分布和各自独立分布乘积的相对熵，用I(X,Y)表示。互信息可以看成是一个随机变量中包含的关于另一个随机变量的信息量。或者说一个随机变量由于已知另一个随机变量而减少的不确定性。信息量、条件熵、熵之间的关系 H(Y|X)=H(Y)-I(X,Y)，H(Y|X)= H(X,Y)-H(X)，由此两个公式即可得I(X,Y)=H(X)+H(Y)-H(X,Y)。

在文本的特征选择中互信息是一个常见的度量方法，它衡量的是词项t对于类别c不确定性的减少程度，显然I(t,c)的值越大，说明词项越重要。具体的词项与类别的互信息的定义式为

用MLE估计概率值后，得到互信息公式

其中，表示的是词项（）和类别（）情况下所对应的文档数目，于是，表示包含词项t且属于类c的文档数；表示包含词项t且不属于类c的文档数；表示不包含词项t且属于类c的文档数；表示不包含词项t且不属于类c的文档数；是所有文档的数目。

例：在含有6个类别的文档集中进行互信息特征抽取的例子

|  |  |  |
| --- | --- | --- |
| Medical  治疗 368.0099534384237  疗效 248.5683626864011  患者 241.35098664035846  病人 169.57704337451065  肿瘤 148.35584261922057  医院 143.7391567175393  癌症 142.88300058328016  手术 136.34570477247502  外科 125.96090451366103  临床 121.06540837233051 | Sports  **农业 1435.160285416867**  **农产品 797.438645422512**  **产量 652.6958119639446**  **粮食 638.809265570511**  训练 597.2331401001437  农村 588.0870413378718  比赛 578.6929977131628  **农户 573.7547140476672**  **土地 545.0782615025894**  **耕地 506.81372161370456** | Agriculture  农业 1917.364197004155  农产品 1045.8375308438606  粮食 836.8183024183533  产量 820.2077664482362  农村 797.9503296860242  农户 755.201145705948  土地 733.2857160468188  耕地 666.7895067793634  面积 610.7652521669077  品种 590.2273092862546 |
| Education  六一 67.49722950858728  今天下午 52.76692698902033  后天下之乐而乐 52.45705681207393  公而忘私 52.45705681207393  阿姨 52.45705681207393  拚命 52.45705681207393  无政府主义 52.45705681207393  办得更好 52.45705681207393  教育电视台 52.45705681207393  倡议书 52.45705681207393 | Electronics  集成电路 403.36287092958247  半导体 399.8186372766048  电子产品 399.8186372766048  电子部 284.4972962524497  电子 240.2675387388617  工业园区 199.58052041685784  电子元件 199.58052041685784  科研开发 131.17323206699157  录像机 125.30261148932676  ２７００万 117.77841292622638 | Communication  邮电部 686.5554648733478  通信网 599.2090220100024  通信 590.1858207617084  邮电 394.13444597832074  卫星通信 255.1420080188027  光缆 237.09923663286878  移动电话 215.84702089109126  交换机 215.84702089109126  电信 180.3210448149559  覆盖全国 142.02536989415796 |

### 卡方统计量

在统计学中统计量常常用于检测两个事件的独立性。在特征选择中，两个事件分别是指词项的出现和类别的出现。定义式为：

其中，表示词项出现与否的随机变量的具体值，它的取值范围为0到1。表示是否属于类别的随机变量的具体值，它的取值范围为0到1。表示语料中词项类别组合实际的出现次数，表示词项类别组合期望的出现次数。。一个算术上更简单的计算公式如下：

式子中N11表示包含词项t且属于类别c的文档数，N10表示包含词项t且不属于类别c的文档数，N00不包含词项t且不属于类别c的文档数，N01不包含词项t且属于类别c的文档数。

度量的是期望值E和观察值N的偏离程度。值大则意味着独立性假设不成立，此时期望值和观察值相差不大。值越大，则词项与类目相关度越大。

例子：在含有6个类别的文档集中进行卡方特征抽取的例子

|  |  |  |
| --- | --- | --- |
| Medical  治疗 368.0099534384237  疗效 248.5683626864011  患者 241.35098664035846  病人 169.57704337451065  肿瘤 148.35584261922057  医院 143.7391567175393  癌症 142.88300058328016  手术 136.34570477247502  外科 125.96090451366103  临床 121.06540837233051 | Sports  **农业 1435.160285416867**  **农产品 797.438645422512**  **产量 652.6958119639446**  **粮食 638.809265570511**  训练 597.2331401001437  **农村 588.0870413378718**  比赛 578.6929977131628  **农户 573.7547140476672**  **土地 545.0782615025894**  **耕地 506.81372161370456** | Agriculture  农业 1917.364197004155  农产品 1045.8375308438606  粮食 836.8183024183533  产量 820.2077664482362  农村 797.9503296860242  农户 755.201145705948  土地 733.2857160468188  耕地 666.7895067793634  面积 610.7652521669077  品种 590.2273092862546 |
| Education  六一 67.49722950858728  今天下午 52.76692698902033  后天下之乐而乐 52.45705681207393  公而忘私 52.45705681207393  阿姨 52.45705681207393  拚命 52.45705681207393  无政府主义 52.45705681207393  办得更好 52.45705681207393  教育电视台 52.45705681207393  倡议书 52.45705681207393 | Electronics  集成电路 403.36287092958247  半导体 399.8186372766048  电子产品 399.8186372766048  电子部 284.4972962524497  电子 240.2675387388617  工业园区 199.58052041685784  电子元件 199.58052041685784  科研开发 131.17323206699157  录像机 125.30261148932676  ２７００万 117.77841292622638 | Communication  邮电部 686.5554648733478  通信网 599.2090220100024  通信 590.1858207617084  邮电 394.13444597832074  卫星通信 255.1420080188027  光缆 237.09923663286878  移动电话 215.84702089109126  交换机 215.84702089109126  电信 180.3210448149559  覆盖全国 142.02536989415796 |

### 频率

频率计算方法有两种，第1种是词项t在c类文档集中出现次数/c类文档集包含的词项总数。第2种是c类文档集中包含词项t的文档数/c类文档集文档总数。

例子：在含有6个类别的文档集中进行频率特征抽取的例子

|  |  |  |
| --- | --- | --- |
| Medical  治疗 0.5098039215686274  专家 0.3333333333333333  医疗 0.29411764705882354  医院 0.27450980392156865  患者 0.27450980392156865  发现 0.2549019607843137  百分之 0.23529411764705882  临床 0.21568627450980393  疗效 0.21568627450980393  疾病 0.19607843137254902 | Sports  文献 0.7086991221069433  中的 0.6065442936951316  原刊期 0.6065442936951316  原刊页 0.6033519553072626  培养 0.4764565043894653  **学习 0.4533120510774142**  训练 0.4301675977653631  第一 0.4205905826017558  重视 0.4158020750199521  **学校 0.4142059058260176** | Agriculture  农业 0.9520078354554359  农村 0.66307541625857  文献 0.6258570029382958  中的 0.6199804113614104  农产品 0.5690499510284035  原刊期 0.5582761998041136  原刊页 0.5582761998041136  作者 0.5288932419196866  土地 0.5004897159647405  措施 0.49167482859941236 |
| Education  学校 0.6440677966101694  学生 0.4915254237288136  培养 0.4406779661016949  学习 0.423728813559322  教师 0.423728813559322  北京 0.3389830508474576  文化 0.288135593220339  知识 0.2711864406779661  事业 0.2711864406779661  教学 0.2711864406779661 | Electronics  电子 0.6296296296296297  计算机 0.4074074074074074  美元 0.37037037037037035  设备 0.2962962962962963  集成电路 0.2962962962962963  工业 0.2962962962962963  １０ 0.25925925925925924  一家 0.25925925925925924  芯片 0.25925925925925924  本报 0.25925925925925924 | Communication  通信 0.72  邮电 0.4  通信网 0.4  电话 0.36  北京 0.36  设备 0.32  业务 0.32  邮电部 0.32  中心 0.28  上海 0.28 |

代码

|  |
| --- |
| #coding:utf-8  """  互信息特、卡方、频率征抽取示例  """  import codecs  import os  import re  from math import log2  from sklearn.datasets import load\_files  import sys  from pyhanlp import \*  def loadStopWords(file\_name):  """  加载停用词表  :param file\_name: 文件名称  :return:  """  global stopwords  with codecs.open(file\_name, 'rb', 'gbk', 'ignore') as infile:  for line in infile:  line = line.strip()  if line:  stopwords.add(line)  def getDocuments(root\_path, file\_path\_li):  """  读取原始文档集并进行预处理  :param file\_path: 文档集所在路径  :return: 预处理后的文档列表  """  all\_text = []  all\_data = load\_files(container\_path=root\_path, categories=file\_path\_li,  encoding="gbk", decode\_error = "ignore")  for label, raw\_text in zip(all\_data.target, all\_data.data):  word\_li = preprocess(raw\_text)  label = all\_data.target\_names[label]  all\_text.append((label, set(word\_li)))  return all\_text  def preprocess(raw\_text):  """  预处理  :param raw\_text:  :return:  """  global stopwords  # 将换行回车符替换为空格  raw\_text = re.sub(u'\r|\n', ' ', raw\_text)  # 去掉数值字母  raw\_text = re.sub(u'[0-9a-zA-z\.]+', u'', raw\_text)  # 分词  word\_li = [w.word for w in HanLP.segment(raw\_text)]  # 去除空白符  word\_li = [w.strip() for w in word\_li if w.strip()]  # 移除单字词  word\_li = [w for w in word\_li if len(w)>1]  # 去除停用词  word\_li = [w for w in word\_li if w not in stopwords]  return word\_li  def getVocabulary(all\_text):  """  获取文档集词汇表  :param all\_text:  :return:  """  global vocabulary  for label, word\_set in all\_text:  vocabulary |= word\_set  def multual\_infomation(N\_10, N\_11, N\_00, N\_01):  """  互信息计算  :param N\_10:  :param N\_11:  :param N\_00:  :param N\_01:  :return: 词项t互信息值  """  N = N\_11 + N\_10 + N\_01 + N\_00  I\_UC = (N\_11 \* 1.0 / N) \* log2((N\_11 \* N \* 1.0) / ((N\_11 + N\_10) \* (N\_11 + N\_01))) + \  (N\_01 \* 1.0 / N) \* log2((N\_01 \* N \* 1.0) / ((N\_01 + N\_00) \* (N\_01 + N\_11))) + \  (N\_10 \* 1.0 / N) \* log2((N\_10 \* N \* 1.0) / ((N\_10 + N\_11) \* (N\_10 + N\_00))) + \  (N\_00 \* 1.0 / N) \* log2((N\_00 \* N \* 1.0) / ((N\_00 + N\_10) \* (N\_00 + N\_01)))  return I\_UC  def chi\_square(N\_10, N\_11, N\_00, N\_01):  """  卡方计算  :param N\_10:  :param N\_11:  :param N\_00:  :param N\_01:  :return: 词项t卡方值  """  fenzi = (N\_11 + N\_10 + N\_01 + N\_00)\*(N\_11\*N\_00-N\_10\*N\_01)\*(N\_11\*N\_00-N\_10\*N\_01)  fenmu = (N\_11+N\_01)\*(N\_11+N\_10)\*(N\_10+N\_00)\*(N\_01+N\_00)  return fenzi\*1.0/fenmu  def freq\_select(t\_doc\_cnt, doc\_cnt):  """  频率特征计算  :param t\_doc\_cnt: 类别c中含有词项t的文档数  :param doc\_cnt: 类别c中文档总数  :return: 词项t频率特征值  """  return t\_doc\_cnt\*1.0/doc\_cnt  def selectFeatures(documents, category\_name, top\_k, select\_type="chi"):  """  特征抽取  :param documents: 预处理后的文档集  :param category\_name: 类目名称  :param top\_k: 返回的最佳特征数量  :param select\_type: 特征选择的方法，可取值chi,mi,freq，默认为chi  :return: 最佳特征词序列  """  L = []  # 互信息和卡方特征抽取方法  if select\_type == "chi" or select\_type == "mi":  for t in vocabulary:  N\_11 = 0  N\_10 = 0  N\_01 = 0  N\_00 = 0  N = 0  for label, word\_set in documents:  if (t in word\_set) and (category\_name == label):  N\_11 += 1  elif (t in word\_set) and (category\_name != label):  N\_10 += 1  elif (t not in word\_set) and (category\_name == label):  N\_01 += 1  elif (t not in word\_set) and (category\_name != label):  N\_00 += 1  else:  print("N error")  exit(1)  if N\_00 == 0 or N\_01 == 0 or N\_10 == 0 or N\_11 == 0:  continue  # 互信息计算  if type == "mi":  A\_tc = multual\_infomation(N\_10, N\_11, N\_00, N\_01)  # 卡方计算  else:  A\_tc = chi\_square(N\_10, N\_11, N\_00, N\_01)  L.append((t, A\_tc))  # 频率特征抽取法  elif select\_type == "freq":  for t in vocabulary:  # C类文档集中包含的文档总数  doc\_cnt = 0  # C类文档集包含词项t的文档数  t\_doc\_cnt = 0  for label, word\_set in documents:  if category\_name == label:  doc\_cnt += 1  if t in word\_set:  t\_doc\_cnt += 1  A\_tc = freq\_select(t\_doc\_cnt, doc\_cnt)  L.append((t, A\_tc))  else:  print("error param select\_type")  return sorted(L, key=lambda x:x[1], reverse=True)[:top\_k]  # 定义停用词表  stopwords = set()  # 定义词汇表  vocabulary = set()  if \_\_name\_\_ == "\_\_main\_\_":  # if len(sys.argv) != 2:  # print("python 特征选择.py chi|mi|freq")  # exit(1)  # feature\_select\_type = sys.argv[1]  # 加载停用词  loadStopWords(r"dictionary/stopwords.txt")  print("stopwords len = ", len(stopwords))  # 读取文档集  category\_name\_li = ["Medical", "Sports", "Agriculture",  "Education", "Electronics", "Communication"]  # 输出语料  # outputCorpus(category\_name\_li)  # 获取文本  all\_text = getDocuments(r"data/news", category\_name\_li)  print("all\_text len = ", len(all\_text))  # 读取词汇表  getVocabulary(all\_text)  print("vocabulary len = ", len(vocabulary))  # 获取特征词表  print("="\*20, '\n', " 卡方特征选择 \n", "="\*20)  feature\_select\_type = "chi"  for category\_name in category\_name\_li:  # 特征抽取，最后一个参数可选值 "chi"卡方,"mi"互信息,"freq"频率  feature\_li = selectFeatures(all\_text, category\_name, 10, feature\_select\_type)  print(category\_name)  for t, i\_uc in feature\_li:  print(t, i\_uc)  print("="\*10)    print("="\*20, '\n', " 互信息特征选择 \n", "="\*20)  feature\_select\_type = "mi"  for category\_name in category\_name\_li:  # 特征抽取，最后一个参数可选值 "chi"卡方,"mi"互信息,"freq"频率  feature\_li = selectFeatures(all\_text, category\_name, 10, feature\_select\_type)  print(category\_name)  for t, i\_uc in feature\_li:  print(t, i\_uc)  print("="\*10)  print("="\*20, '\n', " 频率特征选择 \n", "="\*20)  feature\_select\_type = "freq"  for category\_name in category\_name\_li:  # 特征抽取，最后一个参数可选值 "chi"卡方,"mi"互信息,"freq"频率  feature\_li = selectFeatures(all\_text, category\_name, 10, feature\_select\_type)  print(category\_name)  for t, i\_uc in feature\_li:  print(t, i\_uc)  print("="\*10)  print("program finished") |

### 信息增益

利用信息增益进行特征选择的方法的基本原理是选择能为分类系统带来最多信息量的特征，从另一个角度来说，就是选择那些使分类系统不确定性减少的特征。对分类系统来说，类别C是变量，它可能的取值是C1,C2,…,Cn，而每个类别出现的概率是P(C1),P(C2),…,P(Cn)，此时分类系统的熵为

套用到通信系统的信源，则系统输出的类别就是信源输出的符号种类。如何将特征t结合到系统熵公式中以描述t给系统熵带来的变化呢？答案就是使用条件熵。也就是固定特征t的条件下计算系统条件熵。这里需要说明熵和信息量在值上相等，只不过熵用来描述信源的不确定程度，信息量描述信宿收到信源信息后得到的信息量。条件熵的计算公式为

其中，同理。

因此，最终特征t能够给系统带来的信息增益为

其中，，,。

信息增益也是考虑了特征出现和不出现两种情况，但信息增益最大的问题还在于它仅仅能考察特征对整个系统的贡献，而不能详细到某个类别上，这就使得它仅仅适合用来做所谓“全局”的特征选择。

详细内容可以阅读<https://www.cnblogs.com/bhlsheji/p/4580439.html>

|  |
| --- |
| #coding:utf-8  """  特征选择信息增益法演示  """  from sklearn.datasets import load\_files  from pyhanlp import \*  from math import log2  import numpy as np  import re  NotionalTokenizer = JClass("com.hankcs.hanlp.tokenizer.NotionalTokenizer")  print("加载文件...")  data = load\_files(r'data/news', encoding='gbk', decode\_error='ignore')  # 预处理  def preprocess(text):  # 只保留中文字符  text = translate(text)  # 分词  word\_li = NotionalTokenizer.segment(text)  # 保留字数>1的词，去掉人名  word\_li = [w.word for w in word\_li if len(w.word) > 1 and word.nature != 'nr']  return word\_li  # 仅保留中文字符  def translate(text):  p2 = re.compile(u'[^\u4e00-\u9fa5]') # 中文的编码范围是：\u4e00到\u9fa5  zh = " ".join(p2.split(text)).strip()  zh = ",".join(zh.split())  res\_str = zh # 经过相关处理后得到中文的文本  return res\_str  # 计算类别熵  label\_entropy = 0.0  label\_dict = dict()  for label in data.target:  label\_dict.setdefault(label, 0)  label\_dict[label] += 1  for ci in label\_dict.keys():  p\_r\_ci = label\_dict[ci]\*1.0/len(data.target)  label\_entropy += -1.0\*p\_r\_ci\*log2(p\_r\_ci)  print('label\_entropy = ', label\_entropy)  # 统计词语的出现信息  word\_dict = dict()  for text, label in zip(data.data, data.target):  word\_li = preprocess(text)  word\_li = list(set(word\_li))  for word in word\_li:  word\_dict.setdefault(word, [0]\*len(label\_dict))  word\_dict[word][label] += 1  print("vocabulary len = ", len(word\_dict))  # 计算每个词语的信息增益  word\_ig\_li = []  for t, cnt\_li in word\_dict.items():  # 词语至少出现10次  if sum(cnt\_li) < 10:  pass  p\_t = sum(cnt\_li)\*1.0/len(data.data)  p\_t\_n = 1 - p\_t  ci\_t\_entropy = 0.0  ci\_t\_n\_entropy = 0.0  for cnt in cnt\_li:  p\_ci\_t = cnt/sum(cnt\_li)  p\_ci\_n\_t = 1 - p\_ci\_t  if p\_ci\_t == 0:  ci\_t\_entropy += 0  else:  ci\_t\_entropy += p\_ci\_t \* log2(p\_ci\_t)  if p\_ci\_n\_t == 0:  ci\_t\_n\_entropy += 0  else:  ci\_t\_n\_entropy += p\_ci\_n\_t \* log2(p\_ci\_n\_t)  t\_entropy = p\_t \* ci\_t\_entropy  t\_n\_entropy = p\_t\_n \* ci\_t\_n\_entropy  ig\_t = label\_entropy + t\_entropy + t\_n\_entropy  word\_ig\_li.append((t, ig\_t))  word\_ig\_li = sorted(word\_ig\_li,key=lambda x:x[1], reverse=True)  for word, ig in word\_ig\_li[:60]:  print(word, ig) |

### 不同特征选择方法的比较

对于卡方特征选择法，即使词项t几乎不携带任何有关文档归属类别c的信息，t和c的独立性假设有时也可能在置信度很高的情况下被拒绝。对于罕见词项尤其如此，罕见词就代表着统计显著性。由于卡方基于显著统计性来选择特征，因此它会比互信息选出更多的罕见词，而这些词对于分类是不太可靠的。尽管互信息和卡方有很多不同之处，基于两者的分类精度看上去并没有系统上的太大不同。不论是互信息、卡方还是频率方法，都是基于贪心的策略，筛选出的词语之间存在较高的相关性，也就是存在冗余信息，尽管这种冗余会对分类精度造成负面影响，但是由于非贪心策略计算开销大，因而在文本分类中很少使用。卡方特征抽取方法和互信息特征抽取方法都需要至少2个类别的数据，频率特征抽取方法需要至少1个类别的数据。

### 多类问题的特征选择方法

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 词1 | 词2 | … | 词m |
| c1 | A(t1,c1) | A(t2,c1) | … | A(tm,c1) |
| c2 | A(t1,c2) | A(t2,c2) | … | A(tm,c2) |
| … |  |  |  |  |
| cn | A(t1,cn) | A(t2,c2) | … | A(tm,cn) |
| 平均值 | a(t1) | a(t2) | … | a(tm) |

把每个类别ci看作一个二分类问题，对每个类ci计算每个词项ti的效用指标。最后对表中的每个列（即每个词项ti）求平均值a(ti)作为词项ti的最终的效用指标。

## 关键词抽取

### TF-IDF关键词抽取

在“特征选择”那节中，我们用互信息法、频率法、卡方法、信息增益法抽取出了6个类目关的关键词，从抽取结果看出“运动”类中含有“农业”、“教育”类的关键词，说明我们的“运动”类语料是不纯的或者说文本本身在内容上就既有“运动”类关键词又有“教育”类关键词，于是我们想到了用聚类方法将“运动”类下所有文本划分为几个簇，去看看具体是哪些簇含有了“农业”类以及“教育”类的关键词。对于完全属于“农业”或“教育”类的文本我们将其直接剔除，而对于含有两个类目（“运动”+“农业”或“运动”+“教育”）关键词的文本我们只能通过规则的方法予以处理。

|  |
| --- |
| **心得：实践中造成误分类的原因主要有两种，第1种是由于语料不纯导致，也就是训练文本放错了类目。第2种是由于文本所描述的内容自身就含有两个以上类目的关键词。第2种问题不论文本长短都有可能发生。比如说非常经典的几个商品示例“电话机插座”，“佳能相机专用相机套”，“苹果笔记本电脑包”等等等。对于这种问题，需要具体领域具体分析，针对商品类就涉及到识别核心物品词，通俗地说就是要让机器明白到底是电话机还是插座，到底是相机，还是相机套，到底是笔记本还是电脑包。不要想当然认为这个问题非常好解决，从规则上可以解决一部分问题，但是中文的表述的乱序就是无规律可循。** |

我们用tf-idf法看看对“运动”类文本聚类（聚类使用的是single-pass方法，可以看前single-pass 示例）得到的每个簇的前10和后10关键词，看看是否有“农业”类和“教育”类的词语。tf-idf法抽取关键词的代码见下，tf值的计算我们没有采用NLTK库的TextCollection的tf方法，因为该方法的计算速度实在太慢了，原因是它直接遍历文本单词列表做的，我们使用的是NLTK库的FreqDist计算的tf值。当然idf值的计算使用的是TextCollection，但是它的速度依然很慢，我没有具体看里边的实现，估计是每计算一个词的idf值，都会遍历一遍整个文档集。说到这里恐怕大家对TextCollection方法很失望，我也是特别失望，不过实在不想自己去写这样的代码，原因是第1写了也留不下来，第2写了还得检查，第3统计词频单线程的话肯定慢，综上，还是用现成的可用的代码吧。

|  |
| --- |
| #coding:utf-8  """  对"运动"类增量聚类结果中的每个簇抽取关键词  方法为tf-idf法  """  import codecs  from pyhanlp import \*  import re  from nltk.probability import FreqDist  from nltk.text import TextCollection  import time  NotionalTokenizer = JClass("com.hankcs.hanlp.tokenizer.NotionalTokenizer")  # 仅保留中文字符  def translate(text):  p2 = re.compile(u'[^\u4e00-\u9fa5]') # 中文的编码范围是：\u4e00到\u9fa5  zh = " ".join(p2.split(text)).strip()  zh = ",".join(zh.split())  res\_str = zh # 经过相关处理后得到中文的文本  return res\_str  # 预处理，实词分词器分词，查询词语向量，并返回文本向量  def preprocess(text):  # 去掉非中文字符  text = translate(text)  # 将\r\n替换为空格  text = re.sub(u'[\r\n]+', u' ', text)  # 分词与词性标注，使用实词分词器  word\_li = NotionalTokenizer.segment(text)  word\_li = [w.word for w in word\_li]  # 去掉单字词  word\_li = [w for w in word\_li if len(w)>1]  return word\_li  text\_dict = dict()  with codecs.open('data/res\_single\_pass.txt', 'rb', 'utf-8', 'ignore') as infile:  for line in infile:  line = line.strip()  if line:  cluster\_ser, text = line.split(u'\t')  text\_dict.setdefault(cluster\_ser, [])  text\_dict[cluster\_ser].append(text)  outfile = open('data/cluster\_keywords.txt', 'wb')  outfile2 = open('data/cluster\_keywords2.txt', 'wb')  for cluster\_ser, text\_li in text\_dict.items():  print("cluster", cluster\_ser, "text cnt=", len(text\_li))  # if cluster\_ser == "3" or cluster\_ser == "5":  # continue  t0 = time.time()  vocabulary\_set = set()  cluster\_text\_li = []  for text\_ser, text in enumerate(text\_li):  word\_li = preprocess(text)  cluster\_text\_li.append(tuple(word\_li))  vocabulary\_set |= set(word\_li)  t1 = time.time()  print("预处理簇内文本 %.2f s, 词汇表长度 = %d" % ((t1-t0), len(vocabulary\_set)) )  stats = TextCollection(cluster\_text\_li)  fdist = FreqDist([w for text in cluster\_text\_li for w in text])  t0 = time.time()  word\_li = []  for word in vocabulary\_set:  # 计算词语在簇内的tf值  word\_tf = fdist.freq(word)  # 计算词语在簇内文档间的idf值  word\_idf = stats.idf(word)  # 计算词语tf-idf值  word\_tf\_idf = word\_tf \* word\_idf  if len(cluster\_text\_li) > 1:  word\_li.append((word, word\_tf\_idf))  else:  word\_li.append((word, word\_tf))  t1 = time.time()  print("计算词语tf值idf值tf\_idf值 %.2f s" % (t1 - t0))  word\_li = sorted(word\_li, key=lambda x:x[1], reverse=True)  out\_str = u'%s\t%s\n' %(cluster\_ser, u' '.join([u'%s:%.3f' % (w[0],w[1]) for w in word\_li[:10]]))  outfile.write(out\_str.encode('utf-8', 'ignore'))  out\_str = u'%s\t%s\n' %(cluster\_ser, u' '.join([u'%s:%.3f' % (w[0],w[1]) for w in word\_li[-10:]]))  outfile2.write(out\_str.encode('utf-8', 'ignore'))  outfile.close()  outfile2.close() |

抽取结果见下

|  |
| --- |
| 1 噪声:0.012 线粒体:0.011 赛场:0.009 举重:0.007 心肌:0.007 运动员:0.007 睾酮:0.007 肌球蛋白:0.006 赛前:0.006 皮质醇:0.005  2 射箭:0.089 世界纪录:0.067 山本博:0.067 日本:0.067 单轮:0.044 获得:0.044 成绩:0.044 运动员:0.044 男子:0.044 今天:0.044  21 日本:0.030 古桥:0.017 参观:0.016 参加:0.012 奥运会:0.012 主席:0.012 友谊:0.012 举办:0.012 体育:0.011 相信:0.011  38 万人:0.137 上年:0.074 增加:0.059 百分点:0.025 学校:0.025 增长:0.025 全国:0.025 在校生:0.021 初中:0.020 专任教师:0.018  12 女队:0.094 男队:0.094 赛区:0.065 出线权:0.036 朝鲜:0.029 苏联:0.029 形势:0.022 比赛:0.022 淘汰:0.022 明朗:0.022  6 南华:0.056 东华:0.044 元老:0.044 愉园:0.044 刘锦波:0.022 上半时:0.022 体力:0.022 何耀强:0.022 足球:0.022 邀请赛:0.022  14 美国:0.035 铅球:0.035 男子:0.035 竞走:0.023 女子:0.023 举行:0.012 克里:0.012 萨克斯比:0.012 田径赛:0.012 巴恩斯:0.012  27 运动会:0.040 沙漠:0.040 沙丘:0.030 宁夏:0.030 黄河:0.020 比赛:0.020 米高:0.020 大漠:0.020 接力赛:0.020 机关:0.010  18 心理:0.120 调控:0.080 运动员:0.080 应激:0.060 赛前:0.060 射击:0.040 研究:0.040 方法:0.040 提供:0.020 达到:0.020  39 卡巴迪:0.088 印度:0.059 举行:0.059 比赛:0.059 参加:0.059 亚运会:0.059 卡纳塔克邦:0.029 日电:0.029 加拉邦:0.029 新华社:0.029  24 高尔夫球:0.135 缅甸:0.081 选手:0.081 参加:0.081 亚运会:0.081 仰光:0.054 选出:0.054 比赛:0.054 协会:0.054 日电:0.027  **3 学生:0.009 教育:0.008 课程:0.008 幼儿:0.005 运动员:0.004 孩子:0.004 教学:0.004 学校:0.004 动作:0.003 学科:0.003**  35 上海:0.083 东华:0.067 元老:0.067 台北:0.050 足球:0.050 邀请赛:0.033 足球赛:0.033 孔雀:0.033 日电:0.017 海东:0.017  9 登山队:0.066 和平:0.053 成功:0.053 成都:0.053 中苏:0.039 总领事:0.026 登上:0.026 欢迎:0.026 艾德:0.026 队伍:0.026  8 成绩:0.012 中国队:0.011 金牌:0.011 世界:0.011 项目:0.011 锦标赛:0.010 选手:0.010 中国:0.010 朝鲜:0.009 女子:0.008  31 南朝鲜队:0.081 比赛:0.054 快讯:0.054 日电:0.027 汤姆斯杯:0.027 中国羽毛球队:0.027 金文秀:0.027 记者:0.027 中国男队:0.027 名古屋:0.027  15 尼泊尔:0.075 向导:0.075 成功:0.050 登上:0.050 登山:0.050 瑞典:0.050 日电:0.025 旅游部:0.025 基尔:0.025 峰顶:0.025  29 澳门:0.141 国际奥委会:0.054 绍祖:0.043 体育:0.033 加入:0.033 小可:0.022 支持:0.022 主席:0.022 访问:0.022 重申:0.022  13 选手:0.171 西德:0.049 金牌:0.049 比赛:0.037 银牌:0.037 苏联:0.037 法国:0.024 决出:0.024 夺走:0.024 英国:0.024  25 飞马:0.015 比赛:0.011 复赛:0.011 进行:0.011 机械厂:0.011 自行车:0.011 机床:0.008 足球赛:0.008 广东:0.008 顺德:0.008  33 亚运会:0.073 领导:0.055 记者:0.036 领导人:0.036 党和国家:0.036 捐款:0.036 同志:0.036 张百发:0.036 都向:0.036 今天:0.036  **34 信鸽:0.220 归巢:0.100 北京:0.040 参加:0.040 今天:0.040 日电:0.020 运到:0.020 新县:0.020 春季:0.020 山下:0.020**  **5 教育:0.006 学生:0.005 学校:0.004 教学:0.003 教师:0.003 体育:0.003 俱乐部:0.003 体育产业:0.003 冬泳:0.003 运动:0.003**  32 加布:0.015 贵桑:0.012 顶峰:0.012 罗则:0.012 报告:0.012 下撤:0.011 达穷:0.009 留在:0.009 中国队:0.009 洛泽:0.009  28 山峰:0.020 干城章嘉峰:0.015 和平:0.010 沙峰:0.010 运动健将:0.010 登山队:0.010 纳木那尼峰:0.010 宁金抗:0.010 活动:0.010 斯科特:0.005  40 纪念币:0.135 世界杯:0.054 足球赛:0.054 发行:0.054 面值:0.054 一面:0.054 银质:0.027 和平:0.027 日电:0.027 图案:0.027  20 登山队:0.054 少先队员:0.054 和平:0.033 红领巾:0.022 大本营:0.022 三国:0.022 糖果:0.022 少先队:0.022 队旗:0.022 队员:0.022  10 印度:0.111 金牌:0.083 亚运会:0.083 最近:0.056 游泳:0.056 辛格:0.056 表示:0.056 奖牌:0.056 日电:0.028 汉城亚运会:0.028  41 赞助:0.103 亚运会:0.077 佳能公司:0.077 合同:0.051 第十一届:0.051 日本:0.051 北京:0.051 签字仪式:0.051 传送:0.026 日电:0.026  23 九江:0.020 广东:0.013 外贸:0.013 今年:0.013 结束:0.013 青年:0.013 湖北:0.013 机械:0.013 广东队:0.013 收兵:0.013  16 足球队:0.082 佛山:0.049 广东:0.049 客队:0.049 广州:0.049 留尼旺:0.033 友谊赛:0.033 法国:0.033 去年:0.016 日电:0.016  22 广东电视台:0.081 亚运:0.081 播出:0.054 亚洲:0.054 系列片:0.054 去年:0.027 日电:0.027 收集:0.027 国家:0.027 张家昌:0.027  17 鞍钢:0.079 朝鲜人民军:0.063 比赛:0.048 女篮:0.048 沈阳军区:0.048 篮球:0.032 结束:0.032 男队:0.032 访问:0.032 俱乐部:0.032  19 发奖:0.133 小组:0.048 工作:0.038 亚运会:0.038 组建:0.038 进行:0.029 竞赛部:0.029 组委会:0.019 基本:0.019 包括:0.019  4 营地:0.083 突击组:0.083 到达:0.062 队员:0.042 登山队:0.042 仍然:0.021 和平:0.021 日电:0.021 夏令时:0.021 弥漫:0.021  36 伊万诺娃:0.021 外国:0.018 苏联:0.015 世界:0.015 佟璐:0.012 登上:0.010 山岳:0.009 北京:0.009 日本京都大学:0.009 征服:0.009  30 男排:0.080 喀麦隆:0.060 国际排联:0.060 非洲:0.060 资助:0.040 世界:0.040 小时:0.040 训练:0.040 提高:0.040 日电:0.020  37 集体舞:0.070 亚运:0.070 北京:0.053 握手:0.035 时而:0.035 舞蹈:0.035 成功:0.018 日电:0.018 友谊:0.018 和平:0.018  11 苏联队:0.014 世界杯:0.014 分组:0.014 国际:0.014 乒乓球:0.014 排球:0.014 男队:0.014 联赛:0.014 团体赛:0.014 分在:0.014  26 中国:0.031 东药:0.021 辽宁:0.021 中国队:0.016 北京:0.016 奥林匹克:0.016 哈尔滨:0.015 一队:0.012 齐齐哈尔:0.011 广东:0.010  7 太极:0.053 太极拳:0.041 推手:0.021 研究:0.019 八门:0.018 状态:0.013 体现:0.011 学者:0.011 不同:0.010 描述:0.009 |

第1列为簇序号，后边的为关键词以及对应的tf或tf-idf值。这里说明一下值的计算，对于只有一个文本的簇，关键词采用tf值，因为我们没法计算idf。对于含有2个或2个以上的簇，关键词采用tf-idf值。

关键词抽取速度总结下表，4万词种，75万词形

|  |  |  |
| --- | --- | --- |
|  | TextCollection速度 | FreqDist速度 |
| 统计每种词tf值 | 80词种/秒 | 400词种每秒 |
| 统计idf值 | 80词种/秒 |  |

再来大致分析一下抽取出的关键词，3号簇10个词中有7个都是教育类的词汇，34号簇既没有教育类的词汇也没有运动类词汇，5号簇教育类的词也比较多，几乎占了一半，它就是前边说的含有两个类别的关键词，对于这样的簇，我们应该在分类时单独加规则识别，在训练时还是先把它剔除。有一个问题细心的你一定发现了，就是没发现农业类的词，其实这也就再一次说明农业类词汇在“运动”类目下是低频的，因此互信息法，卡方法都能抽取出农业类词，而频率法抽不到农业类的词。在任何一篇文本中低频词的数量都是很多的。因此，我们这里抽取出的前10低频词中没有看到“农村”这个词语。因此用grep "农村" res\_single\_pass.txt >nongcun.txt ，将含有“农村”的文本全部输出到nongcun.txt中（注意要求res\_single\_pass.txt编码为utf-8），发现只有3、5、27号簇含有关键词“农村”，这里我们直接把这些簇剔除， 以上人工通过簇关键词筛选得到了新的“运动”类语料，语料提纯代码如下

|  |
| --- |
| #coding:utf-8  """  对运动类语料提纯  """  import codecs  import os    stay\_corpus\_set = set([1, 2, 21, 12, 6, 14, 18, 39, 24, 35,  9, 8, 31, 15, 29, 13, 25, 32, 28, 40, 20,  10, 41, 16, 17, 19, 4, 36])  # 清空目录下所有文件  def del\_file(path):  ls = os.listdir(path)  for i in ls:  c\_path = os.path.join(path, i)  if os.path.isdir(c\_path):  del\_file(c\_path)  else:  os.remove(c\_path)  del\_file('data/new\_sports')  with codecs.open('data/res\_single\_pass.txt', 'rb', 'utf-8', 'ignore') as infile:  text\_cnt = 0  for line in infile:  line = line.strip()  if line:  cluster\_ser, text = line.split(u'\t')  if int(cluster\_ser) in stay\_corpus\_set:  with open('data/new\_sports/%d.txt'%(text\_cnt), 'wb') as outfile:  out\_str = u'%s' % line  outfile.write(out\_str.encode('gbk', 'ignore'))  text\_cnt += 1 |

下边我们来看看经过对“运动”类语料提纯后再运行特征选择方法得到的各类目关键词结果

|  |  |  |
| --- | --- | --- |
| Medical  治疗 483.82508111754305  患者 329.5441583683698  疗效 232.32078121981053  病人 228.0573717119743  医院 158.78443782500455  手术 152.7793500538044  医疗 139.05622500204558  临床 136.36637647007447  医生 129.64116188363826  医科 103.2994557851549 | Sports  比赛 756.529632063384  运动员 389.48627723856083  女子 303.7101015482741  男子 230.47880399050968  夺得 211.42983524770855  金牌 203.02025140048127  队员 187.01747942596748  游泳 161.56311888789213  参赛 153.81783984886764  战胜 151.19434098140658 | Agriculture  农业 1065.509606328016  农村 366.4341144028495  比赛 340.3386834694424  文献 295.34074569850094  原刊页 273.61746511062864  原刊期 273.61746511062864  中的 246.74500463674588  土地 244.33215350975502  粮食 239.87128603306473  作者 221.95409830569403 |
| Education  学生 386.0600848032275  教师 357.80999233189794  学校 280.67573643742685  国家教委 267.9289775508538  中小学 184.93526011838424  家长 138.58955959083548  农业 133.72294593232553  办学 132.49823231418026  教育改革 130.0150488360589  入学 122.02098206328607 | Electronics  集成电路 343.72745459048497  半导体 247.78327316828214  电子 226.9263576801892  电子产品 215.47310620705775  电子部 153.41050513436446  电子技术 107.47058334307134  电子元件 107.41155595992849  工业园区 107.41155595992849  电脑 87.61078073538484  尖端 84.80204848993856 | Communication  通信 464.5466076853382  通信网 371.07075894676774  邮电 302.74846411150713  光缆 216.81463732719072  卫星通信 166.06814776259668  电话 142.02995097433362  电信 141.5474525052779  微波 116.30457747496287  交换机 116.30457747496287  简讯 75.67010452477689 |

互信息特征选择法

|  |  |  |
| --- | --- | --- |
| Medical  治疗 483.82508111754305  患者 329.5441583683698  疗效 232.32078121981053  病人 228.0573717119743  医院 158.78443782500455  手术 152.7793500538044  医疗 139.05622500204558  临床 136.36637647007447  医生 129.64116188363826  医科 103.2994557851549 | Sports  比赛 756.529632063384  运动员 389.48627723856083  女子 303.7101015482741  男子 230.47880399050968  夺得 211.42983524770855  金牌 203.02025140048127  队员 187.01747942596748  游泳 161.56311888789213  参赛 153.81783984886764  战胜 151.19434098140658 | Agriculture  农业 1065.509606328016  农村 366.4341144028495  比赛 340.3386834694424  文献 295.34074569850094  原刊页 273.61746511062864  原刊期 273.61746511062864  中的 246.74500463674588  土地 244.33215350975502  粮食 239.87128603306473  作者 221.95409830569403 |
| Education  学生 386.0600848032275  教师 357.80999233189794  学校 280.67573643742685  国家教委 267.9289775508538  中小学 184.93526011838424  家长 138.58955959083548  农业 133.72294593232553  办学 132.49823231418026  教育改革 130.0150488360589  入学 122.02098206328607 | Electronics  集成电路 343.72745459048497  半导体 247.78327316828214  电子 226.9263576801892  电子产品 215.47310620705775  电子部 153.41050513436446  电子技术 107.47058334307134  电子元件 107.41155595992849  工业园区 107.41155595992849  电脑 87.61078073538484  尖端 84.80204848993856 | Communication  通信 464.5466076853382  通信网 371.07075894676774  邮电 302.74846411150713  光缆 216.81463732719072  卫星通信 166.06814776259668  电话 142.02995097433362  电信 141.5474525052779  微波 116.30457747496287  交换机 116.30457747496287  简讯 75.67010452477689 |

频率特征选择法

|  |  |  |
| --- | --- | --- |
| Medical  治疗 0.5098039215686274  专家 0.3333333333333333  医疗 0.29411764705882354  医院 0.27450980392156865  患者 0.27450980392156865  发现 0.2549019607843137  百分之 0.23529411764705882  临床 0.21568627450980393  疗效 0.21568627450980393  １０ 0.19607843137254902 | Sports  比赛 0.6326530612244898  北京 0.4217687074829932  运动员 0.3401360544217687  冠军 0.3333333333333333  第一 0.32653061224489793  选手 0.3197278911564626  中国队 0.2925170068027211  女子 0.272108843537415  成绩 0.2653061224489796  决赛 0.25170068027210885 | Agriculture  农业 0.9520078354554359  农村 0.66307541625857  文献 0.6258570029382958  中的 0.6199804113614104  农产品 0.5690499510284035  原刊页 0.5582761998041136  原刊期 0.5582761998041136  作者 0.5288932419196866  土地 0.5004897159647405  措施 0.49167482859941236 |
| Education  学校 0.6440677966101694  学生 0.4915254237288136  培养 0.4406779661016949  教师 0.423728813559322  学习 0.423728813559322  北京 0.3389830508474576  文化 0.288135593220339  教学 0.2711864406779661  知识 0.2711864406779661  事业 0.2711864406779661 | Electronics  电子 0.6296296296296297  计算机 0.4074074074074074  美元 0.37037037037037035  集成电路 0.2962962962962963  设备 0.2962962962962963  工业 0.2962962962962963  芯片 0.25925925925925924  １０ 0.25925925925925924  本报 0.25925925925925924  一家 0.25925925925925924 | Communication  通信 0.72  通信网 0.4  邮电 0.4  电话 0.36  北京 0.36  设备 0.32  邮电部 0.32  业务 0.32  上海 0.28  网络 0.28 |

### 基于互信息和左右信息熵的短语抽取

短语抽取经常用于搜索引擎的**自动推荐**，**新词识别**。HanLP中在短语抽取中的预处理过程包括断句、分词、去除停用词。在这里我们对去除停用词这一步存疑，因为停用词去除后，原本没有接续关系的n阶共现（n=1,2,3）会被抽取出来。在HanLP中需要统计的共现频次有以下三种，分别是一阶共现（单个词语的频率），二阶共现（2grams频次），三阶共现（3grams频次）。

**互信息**体现了两个词语的相互依赖程度。二元互信息是指两个词语相关性的量。互信息计算公式为：

互信息值越高，表明X和Y的相关性越高，则词语X，Y组成短语的可能性越大。反之，互信息值越低，X和Y之间相关性越低，则X，Y组成短语的可能性越小。在HanLP系统中使用互信息来确定两个词语之间构成短语的可能性。

**左右信息熵**，熵是随机变量不确定性的度量。就像决策树中度量信息增益时对数据做的信息熵一样。在这里用左右信息熵度量2grams短语左右所接词语的不确定性。2grams左右所接词语种类数越多，则信息熵越小。左右信息熵计算公式为：

其中，，。

最后，一个**2grams短语的构成可能性分值**。具体原理可以查看HanLP的官网说明：

<http://www.hankcs.com/nlp/extraction-and-identification-of-mutual-information-about-the-phrase-based-on-information-entropy.html#短语提取> ，

<https://github.com/hankcs/pyhanlp/blob/master/tests/demos/demo_phrase_extractor.py> 。

官网说大部分2grams短语左右熵为0，因此最终生成的2grams短语的影响因子主要来源于互信息值，造成这样的原因是文本太短了。

示例：

|  |
| --- |
| In [**13**]: text  Out[**13**]: '\n ... 算法工程师\n ... 算法（Algorithm）是一系列解决问题的清晰指令，也就是说，能够对一定规范的输入，在有限时间内获得所要求的输出。\n ... 如果一个算法有缺陷，或不适合于某个问题，执行这个算法将不会解决这个问题。不同的算法可能用不同的时间、\n ... 空间或效率来完成同样的任务。一个算法的优劣可以用空间复杂度与时间复杂度来衡量。算法工程师就是利用算法处理事物的人。\n ...\n ... 1职位简介\n ... 算法工程师是一个非常高端的职位；\n ... 专业要求：计算机、电子、通信、数学等相关专业；\n ... 学历要求：本科及其以上的学历，大多数是硕士学历及其以上；\n ... 语言要求：英语要求是熟练，基本上能阅读国外专业书刊；\n ... 必须掌握计算机相关知识，熟练使用仿真工具MATLAB等，必须会一门编程语言。\n ...\n ... 2研究方向\n ... 视频算法工程师、图像处理算法工程师、音频算法工程师 通信基带算法工程师\n ...\n ... 3目前国内外状况\n ... 目前国内从事算法研究的工程师不少，但是高级算法工程师却很少，是一个非常紧缺的专业工程师。\n ... 算法工程师根据研究领域来分主要有音频/视频算法处理、图像技术方面的二维信息算法处理和通信物理层、\n ... 雷达信号处理、生物医学信号处理等领域的一维信息算法处理。\n ... 在计算机音视频和图形图像技术等二维信息算法处理方面目前比较先进的视频处理算法：机器视觉成为此类算法研究的核心；\n ... 另外还有2D转3D算法(2D-to-3D conversion)，去隔行算法(de-interlacing)，运动估计运动补偿算法\n ... (Motion estimation/Motion Compensation)，去噪算法(Noise Reduction)，缩放算法(scaling)，\n ... 锐化处理算法(Sharpness)，超分辨率算法(Super Resolution) 手势识别(gesture recognition) 人脸识别(face recognition)。\n ... 在通信物理层等一维信息领域目前常用的算法：无线领域的RRM、RTT，传送领域的调制解调、信道均衡、信号检测、网络优化、信号分解等。\n ... 另外数据挖掘、互联网搜索算法也成为当今的热门方向。\n ... 算法工程师逐渐往人工智能方向发展。\n ... '  In [**15**]: phrase\_list = HanLP.extractPhrase(text, 5)  In [**16**]: print(phrase\_list)  [算法工程师, 算法处理, 一维信息, 算法研究, 信号处理] |

# sklearn库

## 数据集

### 数据集工具类

clearn\_data\_home 清空指定目录

get\_data\_home 获取sklearn数据根目录

load\_files 加载类目数据

dump\_svmlight\_file 转化文件格式为svmlight/libsvm

load\_svmlight\_file 加载文件并进行格式转换

load\_svmlight\_files 加载文件并进行格式转换

#### load\_files

load\_files函数用于加载多类目文件到内存中。适合于读取分类问题的训练语料。语料的目录结构应该是，根目录中存储所有类别的目录，在每个类别的目录中，以文件的形式存储所有文本，一个文本占用一个文件。

**原型**

**sklearn.datasets.load\_files**(container\_path, description=None, categories=None, load\_content=True, shuffle=True, encoding=None, decode\_error=’strict’, random\_state=0)

**参数**

|  |
| --- |
| * **container\_path**：字符串。分类语料的根目录 * **categories**：字符串集合或None。默认为None * 如果为None，则所子目录也就是所有类别的语料都被加载进来； * 如果为字符串的集合，则指定的子目录（类别）下的语料被加载，其他子目录下的语料忽略掉 * **encoding**：字符串或者None。默认为None * 如果为None，不解码读入的文件； * 如果为字符串，则按照字符串表示的编码类型解码读入的文件 * **decode\_error**：’strict’,’ignore’,’replace’，给出当遇到非指定编码字符时所做的操作 * **random\_state**：一个整数或者一个RandomState实例，或者None * 如果为整数，则它指定随机数生成器的种子 * 如果为RandomState实例，则指定随机数生成器 * 如果为None，则使用默认的随机数生成器np.random |

**返回**

|  |
| --- |
| Bunch类型实例，它具有的属性如下   * **data**：列表，每个元素是字符串形式的一个原始文本 * **target**：列表，每个元素是data列表中对应位置的文本的类别编号 * **target\_names**：字典，键为类别编号，值为对应的类别名称 |

### 文本数据集

|  |  |
| --- | --- |
| 类别 | 数据集 |
| 文本 | fetch\_20newsgroups 新闻文本分类数据集  fetch\_20newsgroups\_vectorized 新闻文本向量化数据集  fetch\_rcv1 路透社英文新闻文本分类数据集 |
| 人脸 | fetch\_lfw\_pairs 人脸数据集  fetch\_lfw\_people 人脸数据集  fetch\_olivetti\_faces 人脸数据集 |
| 图像 | load\_sample\_image 图像数据集  load\_sample\_images 图像数据集  load\_digits 手写体数据集 |
| 医学 | load\_breast\_cancer 乳腺癌数据集  load\_diabetes 糖尿病数据集  load\_linnerud 体能训练数据集 |
| 其他 | load\_wine 葡萄酒数据集  load\_iris 鸢尾花数据集  load\_boston 波士顿房屋数据集  fetch\_california\_housing 加利福尼亚房屋数据集  fetch\_kddcup99 入侵检测数据集  fetch\_species\_distribution 物种分布数据集  fetch\_covtype 森林植被数据集  load\_mldata mldata.org在线下载的数据集 |

## 朴素贝叶斯

### MultinomialNB

**原型**

class **sklearn.naive\_bayes.MultinominalNB**(alpha=1.0, fit\_prior=True, class\_prior=None)

参数

|  |
| --- |
| * **alpha**: 一个浮点数，平滑值 * **fit\_prior**: 布尔值。如果为False，则不去学习，替代以均匀分布；如果为True，则去学习 * **class\_prior**: 一个数组。它指定了每个分类的先验概率，，…，。如果指定了该参数，则每个分类的先验概率不再从数据集中学得 |

**属性**

|  |
| --- |
| * **class\_log\_prior\_**: 一个数组对象，形状为(n\_classes,)。给出了每个类别调整后的经验概率分布的对数值 * **feature\_log\_prob\_**: 一个数组对象，形状为(n\_classes, n\_features)。给出了的经验概率分布的对数值 * **class\_count\_**: 一个数组，形状为(n\_classes,)，是每个类别包含的训练样本数量 * feature\_count\_: 一个数组，形状为(n\_classes, n\_features)。训练过程中，每个类别每个特征遇到的样本数 * **coef\_** : 将多项式模型解释为线性模型后的系数序列w1,w2,…,wn，每个类别的词语多项式权值向量，shpae=[类别数量，词汇表长度] * **intercept\_**: 将多项式模型解释为线性模型后的截距值b，每个类别的先验概率，shape=[类别数量] |

注：关于coef\_和intercept\_两个属性的详细解释，可以参考J. Rennie et al. (2003), Tackling the poor assumptions of naive Bayes text classifiers, ICML，在这篇文章中，作者将朴素贝叶斯模型看成是线性模型，则它与线性支持向量机，逻辑回归，感知机等具有了相同的决策超平面。计算一篇文档属于某个类别c的概率公式，其中P(ci)是类别ci的先验概率，fi是词语i在文档d中的频率，n表示词汇表长度，Nci是词语i在类别c文档集中出现次数，Nc是类别c文档集中词语总数，alphai是词语i的先验概率估计值，一般情况取1（认为词语的出现服从均匀分布），alpha是所有alphai的和，在alphai取1的情况下，alpha取值为词汇表长度。训练的过程就是计算这些数值的过程，当计算完成后，也就得到了一个权值矩阵coef\_，以及类别的概率分布向量intercept\_，对于一篇测试文档，只要将它表示为词汇表向量（向量值为词语的文档频率），然后跟权值矩阵做一次乘法，即可得到该文档属于每个类别的概率值了，取其中值最大的类作为预测类别。

**方法**

|  |
| --- |
| * **fit(X, y[, sample\_weight])**: 训练模型 * **partial\_fit(X, y[, classes, sample\_weight])**: 追加训练模型。该方法主要用于大规模数据集的训练。此时可以将大数据集划分成若干个小数据集，然后在这些小数据集上连续调用partial\_fit方法来训练模型 * **predict(X)**: 用模型进行预测，返回预测值 * **predict\_log\_proba(X)**: 返回一个数组，数组的元素依次是X预测为各个类别的概率的对数值 * **predict\_proba(X):** 返回一个数组，数组的元素依次是X预测为各个类别的概率值 * **score(X, y[, sample\_weight])**: 返回在(X, y)上预测的准确率 |

### BernoulliNB

多变量贝努利将某类别下的文档的生成看作是做m次独立的贝努利试验，其中m是词汇表的长度，每次试验都通过抛硬币（当然实际要通过训练集统计）决定这次对应的词语是否在文本中出现。因此它的似然概率计算公式为P(t|c)=类c文档集中包含词t的文档数/类c文档集中文档总数。而多项式朴素贝叶斯将某类别下的文档的生成看成从词汇表中有放回的抽样，每次随机抽一个词出来，一共抽取文档长度次（单词个数）。因此它的似然概率计算公式为P(t|c)=类c文档集中词语t出现的次数/类c文档集中词语总数。

**原型**

class **sklearn.naive\_bayes.BernoulliNB**(alpha=1.0, binarize=0.0, fit\_prior=True, class\_prior=None)

**参数**

|  |
| --- |
| * **alpha**: 一个浮点数，平滑值 * **binarize**: 一个浮点数或者None * 如果为None，那么会假定原始数据已经二元化了 * 如果是浮点数，那么会以该数值为界，特征取值大于它的作为1；特征取值小于它的作为0。采取这种策略来二元化 * **fit\_prior**: 布尔值。如果为True，则不去学习，替代以均匀分布；如果为False，则去学习 * **class\_prior**: 一个数组。它指定了每个分类的先验概率，，…，。如果指定了该参数，则每个分类的先验概率不再从数据集中学得 |

**属性**

|  |
| --- |
| * **class\_log\_prior\_**: 一个数组对象，形状为(n\_classes,)。给出了每个类别调整后的经验概率分布的对数值 * **feature\_log\_prob\_**: 一个数组对象，形状为(n\_classes, n\_features)。给出了的经验概率分布的对数值 * **class\_count\_**: 一个数组，形状为(n\_classes,)，是每个类别包含的训练样本数量 * **feature\_count\_**: 一个数组，形状为(n\_classes, n\_features)。训练过程中，每个类别每个特征遇到的样本数 |

**方法**

|  |
| --- |
| * **fit(X, y[, sample\_weight])**: 训练模型 * **partial\_fit(X, y[, classes, sample\_weight])**: 追加训练模型。该方法主要用于大规模数据集的训练。此时可以将大数据集划分成若干个小数据集，然后在这些小数据集上连续调用partial\_fit方法来训练模型 * **predict(X)**: 用模型进行预测，返回预测值 * **predict\_log\_proba(X)**: 返回一个数组，数组的元素依次是X预测为各个类别的概率的对数值 * **predict\_proba(X)**: 返回一个数组，数组的元素依次是X预测为各个类别的概率值 * **score(X, y[, sample\_weight])**: 返回在(X, y)上预测的准确率 |

## 支持向量机

### LinearSVC

LinearSVC实现了线性分类支持向量机，它是给根据liblinear实现的，可以用于二类分类，也可以用于多类分类。

**原型**

class **sklearn.svm.LinearSVC**(penalty=’l2’, loss=’squared\_hinge’, dual=True, tol=0.0001, C=1.0, multi\_class=’ovr’, fit\_intercept=True, intercept\_scaling=1, class\_weight=None, verbose=0, random\_state=None, max\_iter=1000)

**参数**

|  |
| --- |
| * **C**:一个浮点数，惩罚参数 * **loss**: 字符串。表示损失函数。可以为如下 * ‘hinge’: 此时为合页损失函数（它是标准SVM的损失函数） * ‘squared\_hing’: 合页损失函数的平方 * **penalty**: 字符串。指定’l1’或者’l2’，惩罚的范数。默认为’l2’（它是标准SVC采用的） * **dual**: 布尔值。如果为true，则解决对偶问题；如果是false，则解决原始问题。当n\_samples>n\_features时，倾向于采用false * **tol**: 浮点数，指定终止迭代的阈值 * **multi\_class**: 字符串，指定多分类问题的策略 * ‘ovr’: 采用one-vs-rest分类策略； * ‘crammer\_singer’: 多类联合分类，很少用。因为它的计算量大，而且精度不会更佳，此时忽略loss,penalty,dual参数 * **fit\_intercept**: 布尔值。如果为true，则计算截距，即决策函数中的常数项；否则忽略截距 * **intercept\_scaling**: 浮点值。如果提供了，则实例X变成向量[X,intercept\_scaling]。此时相当于添加了一个人工特征，该特征对所有实例都是常数值。   当sel.fit\_intercept为True时，实例向量x变为[x, self. intercept\_scale]，例如：一个等于intercept\_scaling的常量“混合”特征将被附加到实例向量末尾。截距变成intercept\_scaling \*合成特征权重。注意!混合特征权重与其他特征一样，服从l1/l2正则化。**为了减少正则化对合成特征权重(也就是对截距)的影响，必须增加intercept\_scaling值**。   * **class\_weight**: 可以是个字典，或者字符串’balanced’。指定各个类的权重，若未提供，则认为类的权重为1 * 如果是字典，则指定每个类标签的权重； * 如果是’balanced’，则每个类的权重是它出现频率的倒数 * **verbose**: 一个整数，表示是否开启verbose输出 * **random\_state**: 一个整数或者一个RandomState实例，或者None * 如果为整数，则它指定随机数生成器的种子 * 如果为RandomState实例，则指定随机数生成器 * 如果为None，则使用默认的随机数生成器 * **max\_iter**: 一个整数，指定最大的迭代次数 |

**属性**

|  |
| --- |
| * **coef\_**: 一个数组，它给出了各个特征的权重 * **intercept\_**: 一个数组，它给出了截距，即决策函数中的常数项 |

**方法**

|  |
| --- |
| * **fix(X,y)**: 训练模型 * **predict(X)**: 用模型进行预测，返回预测值 * **score(X,y[, sample\_weight])**: 返回在(X, y)上预测的准确率 |

## 线性模型

### SGDClassifier

该类实现了用SGD方法进行训练的线性分类器（比如线性SVM，逻辑回归等）。模型每次使用一个样本来估计损失函数梯度。模型的学习速率会随着迭代地进行而减小。模型允许minibatch（在线/离线）学习，详见partial\_fit函数。在使用默认学习速率策略的情况下，为了达到最好的效果，数据应当具有零均值和单位方差。模型的输入数据应当是数组，元素类型为浮点数。算法拟合的模型类型由参数loss决定，默认情况下拟合线性支持向量机。正则化器是添加到损失函数中的罚项，该罚项会将参数向量**向零向量压缩**，罚项可以是平方的欧式2范数，也可以是绝对值1范数，还可以两者的结合。如果由于调节因子使得参数变成0向量，那么更新将被终止，以得到离散模型并实现在线特征选择。有关于损失函数与模型可以参考<https://zhuanlan.zhihu.com/p/63771700>

**原型**

class **sklearn.linear\_model.SGDClassifier**(loss=’hinge’, penalty=’l2’, alpha=0.0001, l1\_ratio=0.15, fit\_intercept=True, max\_iter=None, tol=None, shuffle=True, verbose=0, epsilon=0.1, n\_jobs=1, random\_state=None, learning\_rate=’optimal’, eta0=0.0, power\_t=0.5, class\_weight=None, warm\_start=False, average=False, n\_iter=None)

**参数**

|  |
| --- |
| * **loss**：字符串，损失函数的类型。默认值为’hinge’ * ‘hinge’：合页损失函数，表示线性SVM模型 * ‘log’：对数损失函数，表示逻辑回归模型 * ‘modified\_huber’：’hing’和’log’损失函数的结合，表现两者的优点 * ‘squared\_hinge’：平方合页损失函数，表示线性SVM模型 * ‘perceptron’：感知机损失函数 * **penalty**：字符串，罚项类型 * ‘l2’：2-范数罚项，默认值，线性SVM的标准正则化函数 * ‘l1’：1-范数罚项 * ‘elasticnet’：l2和l1的组合。 * **alpha**：浮点数，罚项前的系数，默认值为0.0001。当参数learning\_rate被设置成optimal的时候，该参数参与learning\_rate值的计算 * **l1\_ratio**：浮点数，elasticnet罚项中l2和l1的权重。取值范围0<=l1\_ratio<=1。默认值为0.15 * **fit\_intercept**：布尔值，是否估计截距，如果为假，认为数据已经中心化 * **max\_iter**：整数，可选的。迭代的最大次数，只影响fit方法，默认值为5。从0.21版以后，如果参数tol不是空，则默认值为1000 * **tol**：浮点数或None，可选的。训练结束的误差边界。如果不是None，则当previous\_loss-cur\_loss<tol时，训练结束。默认值为None，从0.21版以后，默认值为0.001 * **shuffle**：布尔值，可选的。每轮迭代后是否打乱数据的顺序，默认为True * **verbose**：整数，可选的，控制调试信息的详尽程度 * **n\_jobs**：整数，可选的。训练多元分类模型时，使用CPUs的数量，-1为使用全部，默认值为1 * **random\_state**：打乱数据顺序的方式 * **learning\_rate**：字符串，可选的。学习速率的策略 * ‘constant’：eta=eta0 * ‘optimal’：eta=1.0/(alpha\*(t+t0))，默认值 * ‘invscaling’：eta=eta0/pow(t, power\_t) * **eta0**：浮点数，参与learning\_rate计算，默认值为0 * **power\_t**：参与learning\_rate计算，默认值为0.5 * **class\_weight**：词典{class\_label:weight}或’balanced’或None，可选的。类别的权重。如果为None，则所有类的权重为1，’balanced’则根据y自动调节权重，使其反比于类别频率n\_samples/(n\_classes\*np.bincount(y)) * **warm\_start**：布尔值，可选的。设置为True时，使用之前的拟合得到的解继续拟合 * **average**：布尔值，整数，可选的。True时，计算平均SGD权重并存储于coef\_属性中。设置为大于1的整数时，拟合使用过的样本数达到average时，开始计算平均权重 |

**属性**

|  |
| --- |
| * **coef\_**：数组，shape=(1, n\_features)二元分类；(n\_classes, n\_features)多元分类 * **intercept\_**：数组，决策函数中常量b。shape=(1, )二元分类；(n\_classes, )多元分类 * **n\_iter**：整数，训练结束时，实际的迭代次数。对于多元分类来说，该值为所有二元拟合过程中迭代次数最大的 * **loss\_function\_**：使用的损失函数 |

**方法**

|  |
| --- |
| * **decision\_function(X)**：对样本预测置信度得分 * **densify()**：将协方差矩阵转成数组 * **fit(X, y[, coef\_init, intercept\_init\_,…])**：随机梯度下降法拟合线性模型 * **get\_params([deep])**：返回分类器参数 * **partial\_fit(X, y[, classes, sample\_weight])**：增量拟合 * **score(X, y[, sample\_weight])**：返回模型平均准确率 * **set\_params(\*args, \*\*kwargs)**：设置模型参数 * **sparsify()**：将未知数矩阵w转成稀疏格式 |

## K-Means聚类

### Kmeans

**原型**

class **sklearn.cluster.Kmeans**(n\_cluster=8, init=’k-means++’, n\_init=10, max\_iter=300, tol=0.0001, precompute\_distances=’auto’, verbose=0, random\_state=None, copy\_x=True, n\_jobs=1)

**参数**

|  |
| --- |
| * **n\_cluster**: 一个整数，指定分类簇的数量 * **init**: 一个字符串，指定初始均值向量的策略。可以为如下： * ‘k-means++’: 初始化策略选择的初始均值向量之间距离较远，它的效果较好 * ‘random’: 从数据集中随机选择K个样本作为初始均值向量 * 或者提供一个数组，数组的形状为(n\_clusters.n\_features)，该数组作为初始均值向量   注：K均值算法总能够收敛，但是其收敛情况高度依赖于初始化的均值。有可能收敛到局部极小值。因此通常都是用多组初始化均值向量来计算若干次，选择其中最优的那一次。而k-means++策略选择的初始均值向量可以在一定程度上解决这个问题。   * **n\_init**: 一个整数，指定了K均值算法运行的次数。每一次都会选择一组不同的初始化向量，最终算法会选择最佳的分类簇作为最终的结果 * **max\_iter**: 一个整数，指定了单轮k均值算法中，最大的迭代次数。算法总的最大迭代次数为max\_iter\*n\_init * **precompute\_distances**: 可以为布尔值或者字符串’auto’。该参数指定是否提前计算好样本之间的距离（如果提取计算距离，则需要更多的内存，但是算法会运行得更快） * ‘auto’: 如果n\_samples\*n\_clusters>12million，则不提前计算 * True: 总是提前计算 * False: 总是不提前计算 * **tol**: 一个浮点数，指定了算法收敛的阈值 * **n\_jobs**: 一个正数。指定任务并行时指定的CPU数量。如果为-1则使用所有可用的CPU * **verbose**: 一个整数。如果为0，则不输出日志信息；如果为1，则每隔一段时间打印一次日志信息；如果大于1，则打印日志信息更频繁 * **random\_state**: 一个整数或者一个RandomState实例，或者None * 如果为整数，则它指定了随机数生成器的种子 * 如果为RandomState实例，则指定了随机数生成器 * 如果为None，则使用默认的随机数生成器 * **copy\_x**: 布尔值，主要用于precompute\_distances=True的情况 * 如果为True，则预计算距离的时候，并不修改原始数据 * 如果为False，则预计算距离的时候，会修改原始数据用于节省内存；然后当算法结束的时候，会将原始数据还原。但是可能会因为浮点数的表示，会有一些精度误差 |

**属性**

|  |
| --- |
| * **cluster\_centers\_**: 给出分类簇的均值向量 * **labels\_**: 给出了每个样本所属的簇的标记 * **inertia\_**: 给出了每个样本距离它们各自最近的簇中心的距离之和 |

**方法**

|  |
| --- |
| * **fit(X[,y])**: 训练模型 * **fit\_predict(X[,y])**: 训练模型并预测每个样本所属的簇 * **predict(X)**: 预测样本所属的簇 * **score(X[,y])**: 给出了样本距离各簇中心的偏移量的相反数 |

## 模型选择

### StratifiedShuffleSplit

分层随机分割交叉验证器可以将数据分割为训练集和测试集，不过它只提供训练集/测试集数据在原始数据集中的位置索引。由该类生成的交叉验证对象融合了StratifiedKFold和ShuffleSplit两个函数的功能，该对象返回分层随机折，对像通过对每一类保留一定比例的样本生成折。注意：同随机分割一样，分层随机分割不保证所有折都是不同的，即使对于大数据集也不例外。

**原型**

class **sklearn.model\_selection.StratifiedShuffleSplit**(n\_splits=10, test\_size=’default’, train\_size=None, random\_state=None)

**参数**

|  |
| --- |
| * **n\_splits**：整数，默认值为10。重新打乱分割的迭代次数 * **test\_size**：浮点数，None。分割后的测试集大小，默认为浮点数0.1（train\_size没有被设置，否则为训练集大小的补集） * 如果为浮点数，取值范围在0.0到1.0之间，表示分割后的测试集占总数据集的比例； * 如果为整数，表示分割后的测试集含有的绝对样本数； * 如果为None，分割后的测试集大小为训练集大小的补集 * **train\_size**：浮点数，整数或None。默认为None * 如果为浮点数，取值范围在0.0到1.0之间，表示分割后的训练集占总数据集的比例； * 如果为整数，表示分割后的训练集含有的绝对样本数； * 如果为None，分割后的训练集大小为测试集大小的补集 * **random\_state**：一个整数或者一个RandomState实例，或者None * 如果为整数，则它指定随机数生成器的种子 * 如果为RandomState实例，则指定随机数生成器 * 如果为None，则使用默认的随机数生成器np.random |

**方法**

|  |
| --- |
| * **get\_n\_splits(X=None, y=None, groups=None)**：返回打乱迭代次数，所有参数都可省略 * **split(X, y, groups=None)**：返回生成分割后的训练和测试集的索引 * X：原始数据集的数据部分 * y：原始数据集的类别标记部分 * groups：没有用，仅为兼容性保留   注意：随机交叉验证分割起每次调用split方法都可能会返回不同的分割结果，可以通过将参数random\_state设置为一个整数使结果保持不变 |

### train\_test\_split

将矩阵或数组随机拆分成训练和测试数据集。

**原型**

sklearn.model\_selection.**train\_test\_split**(\*arrays, \*\*options)

**参数**

|  |
| --- |
| * **\*arrays**：输入数据。允许的输入类型有list, numpy arrays, scipy-sparse matrices或pandas dataframes * **test\_size**： * 浮点数：（可选的）取值范围0.0到1.0之间，表示测试集占总数据集的比例。如果参数train\_size没有被说明，则默认值为0.25，否则测试集为训练集补集 * 整数：表示测试集所含样本的数量 * None：被设置为train\_size大小的训练集的补集 * **train\_size**：浮点数，整数或None，默认为None * 浮点数：取值范围0.0到1.0之间，表示训练集占总数据集的比例。 * 整数：表示训练集所含样本的数量 * None：被设置为test\_size大小的测试集的补集 * **random\_state**：打乱数据顺序的方法 * **shuffle**：布尔值（可选的）默认值为True。在拆分数据之前是否打乱顺序。如果shuffle为False，那么stratify参数必须为None * **stratify**：类数组或None，默认为None。如果不是None，则数据被拆分为分层形式，使用这些作为类标记 |

**返回**

|  |
| --- |
| * **splitting**：列表，长度=2\*len(arrays)，返回的拆分后的数据 |

## 特征抽取

### CountVectorizer

CountVectorizer类可以构建词袋模型（计数）或one-hot模型

class **sklearn.feature\_extraction.text.CountVectorizer**(input=’content’, encoding=’utf-8’, decode\_error=’strict’, strip\_accents=None, lowercase=True, preprocessor=None, tokenizer=None, stop\_words=None, token\_pattern=’(?u)\b\w\w+\b’, ngram\_range=(1, 1), analyzer=’word’, max\_df=1.0, min\_df=1, max\_features=None, vocabulary=None, binary=False, dtype=<class ‘numpy.int64’>)

**参数**

|  |
| --- |
| * **input**：字符串，可选值{‘filename’, ‘file’, ‘content’}，指定传给fit函数的参数的类型，默认选项为’content’ * ‘filename’：传给fit函数的实参为文件名列表 * ‘file’：传给fit函数的实参为拥有’read’方法的序列项 * ‘content’：传给fit函数的实参为字符串序列或字节串序列 * **encoding**：字符串，默认值为’utf-8’，如果fit函数接收的实参为文件或字节序列，则使用这里指定的编码类型 * **decode\_error**：字符串，可选值{‘strict’, ‘ignore’, ‘replace’}，默认值为None什么也不做 * **strip\_accents**：字符串，可选值{‘ascii’, ‘unicode’, ‘None’}，默认值为None。在预处理过程中去除音调（重音）。’ascii’方法是最快的，但它只适用于拥有直接ASCII映射的字符；’unicode’方法稍慢，但适用于任何字符；’None’默认值什么也不做。 * **analyzer**：字符串，可选值为{‘word’, ‘char’}或可调用对象 * ‘word’：特征由词构成 * ‘char’：特征由ngrams字符构成 * 可调用对象：直接由该函数从语料中抽取出特征序列 * **preprocessor**：可调用对象或None，默认值为None，在分词(tokenizing)和生成ngrams时覆盖预处理步骤 * **tokenizer**：可调用对象或None，默认值为None，在预处理(preprocessing)和生成ngrams时覆盖分词步骤。只有在参数analyzer取值为’word’时，该参数才有作用 * **ngram\_range**：元组(min\_n, max\_n)抽取出ngrams的元个数的下限和上限。所有的符合min\_n<=n<=max\_n数量的ngrams都将被抽取出来 * **stop\_words**：字符串，可选值{‘english’}，列表或None，默认值为None * 如果为字符串，则使用内部支持的字符串指定的语种的停用词表 * 如果为列表，列表中的词语为停用词 * 如果为None，不使用停用词。此时可以借助参数max\_df[0.7,1.0]来根据文档频率自动检测和过滤停用词 * **lowercase**：布尔值，默认值为True。在进行分词之前将所有字符转为小写 * **token\_pattern**：字符串。表示一个词的正则表达式，只有当analyzer为’word’时，该参数才起作用。默认正则表达式将词看成由2个或更多的字母数字构成的串，标点被忽略并且被当作词分隔符 * **max\_df**：浮点数，取值范围[0.0,1.0]或整数，默认值为1.0,当构建词汇表时，词语文档频率高于max\_df，则被过滤。当为整数时，词语文档频次高于max\_df时，则被过滤。当vocabulary不是None时，该参数不起作用 * **min\_df**：浮点数，取值范围[0.0,1.0]或整数，默认为1，该参数除了指下限其他都同max\_df * **max\_features**：整数或None，默认为None。根据term frequence排序后的vocabulary的前max\_features个词作为vocabulary。如果参数vocabulary不是None，则该参数不起作用 * **vocabulary**：dict结果的词典键为词语，值为该词语在文档词矩阵中的索引；也可以是term的可迭代对象 * **binary**：布尔值，默认为False。如果为True，则所有非0词特征都被置为1。这不意味着输出只有0，1两种值，只有tf\_idf中的tf是2gram的 * **dtype**：指定由fit\_transform()或transform()返回的矩阵类型 |

**属性**

|  |
| --- |
| * **vocabulary\_**：词典dict，索引：特征词的映射 * **stop\_words\_**：集合set。被滤掉的词，这些词可能是（1）太多文本中包含该词(max\_df)；（2）太少文本中包含该词(min\_df)；（3）被特征选择截断(max\_features)；该属性只有在没有给定vocabulary参数的时候才有意义。注意：stop\_words\_属性可以变大，并在pickle时增加模型大小。此属性仅用于自省，可以使用delattr安全地删除或在pickle之前设置为None |

**方法**

|  |
| --- |
| * **build\_analyzer()**：返回1个可调用句柄进行预处理和分词 * **build\_preprocessor()**：在分词前，返回1个函数来预处理文本 * **build\_tokenizer()**：返回1个函数来文本分词 * **decode(doc)**：Decode输入到unicode字符串 * **fit(raw\_documents[,y])**：从训练集学习词汇表和idf * **fit\_transform(raw\_documents[,y])**：学习词汇表和idf，返回文档词矩阵 * **get\_feature\_names()**：按文档-词矩阵中词语顺序的词语列表 * **get\_params([deep])**：获取实例的参数 * **get\_stop\_words()**：构建或获取有效停用词列表 * **inverse\_transform(X)**：返回每篇文档中的非0特征词 * **set\_params(\*\*params)**：设置实例的参数 * **transform(raw\_documents, copy=True)**：变换文档到文档词矩阵，此处使用词汇表和文档频率由fit函数或fit\_transform函数学习到 |

### TfidfVectorizer

词袋模型的另一种实现。这种词袋模型的特征还是由词构成，但是每篇文档的各维特征值跟CountVectorizer不一样。在CountVectoizer中，每篇文档的各维特征值是特征词在文本中出现的次数，而TfidfVectorizer中，每篇文档的各维特征值是特征词的tfidf值。区别很明显，除了考虑特征词在文档中出现的频率外，还考虑了词语在文档集中的分布情况（也就是tfidf值）。由于TfidfVectorizer与CountVectorizer的区别不大，因此两个类的参数也几乎相同，这里只列出TfidfVectorizer中独有的参数，其他的参数与CountVectorizer完全一样。

原型

class **sklearn.feature\_extraction.text.TfidfVectorizer**(input=’content’, encoding=’utf-8’, decode\_error=’strict’, strip\_accents=None, lowercase=True, preprocessor=None, tokenizer=None, analyzer=’word’, stop\_words=None, token\_pattern=’(?u)\b\w\w+\b’, ngram\_range=(1, 1), max\_df=1.0, min\_df=1, max\_features=None, vocabulary=None, binary=False, dtype=<class ‘numpy.int64’>, norm=’l2’, use\_idf=True, smooth\_idf=True, sublinear\_tf=False)

**参数**

|  |
| --- |
| * **input**：字符串，可选值{‘filename’, ‘file’, ‘content’}，指定传给fit函数的参数的类型，默认选项为’content’ * ‘filename’：传给fit函数的实参为文件名列表 * ‘file’：传给fit函数的实参为拥有’read’方法的序列项 * ‘content’：传给fit函数的实参为字符串序列或字节串序列 * **encoding**：字符串，默认值为’utf-8’，如果fit函数接收的实参为文件或字节序列，则使用这里指定的编码类型 * **decode\_error**：字符串，可选值{‘strict’, ‘ignore’, ‘replace’}，默认值为None什么也不做 * **strip\_accents**：字符串，可选值{‘ascii’, ‘unicode’, ‘None’}，默认值为None。在预处理过程中去除音调（重音）。’ascii’方法是最快的，但它只适用于拥有直接ASCII映射的字符；’unicode’方法稍慢，但适用于任何字符；’None’默认值什么也不做。 * **analyzer**：字符串，可选值为{‘word’, ‘char’}或可调用对象 * ‘word’：特征由词构成 * ‘char’：特征由ngrams字符构成 * 可调用对象：直接由该函数从语料中抽取出特征序列 * **preprocessor**：可调用对象或None，默认值为None，在分词(tokenizing)和生成ngrams时覆盖预处理步骤 * **tokenizer**：可调用对象或None，默认值为None，在预处理(preprocessing)和生成ngrams时覆盖分词步骤。只有在参数analyzer取值为’word’时，该参数才有作用 * **ngram\_range**：元组(min\_n, max\_n)抽取出ngrams的元个数的下限和上限。所有的符合min\_n<=n<=max\_n数量的ngrams都将被抽取出来 * **stop\_words**：字符串，可选值{‘english’}，列表或None，默认值为None * 如果为字符串，则使用内部支持的字符串指定的语种的停用词表 * 如果为列表，列表中的词语为停用词 * 如果为None，不使用停用词。此时可以借助参数max\_df[0.7,1.0]来根据文档频率自动检测和过滤停用词 * **lowercase**：布尔值，默认值为True。在进行分词之前将所有字符转为小写 * **token\_pattern**：字符串。表示一个词的正则表达式，只有当analyzer为’word’时，该参数才起作用。默认正则表达式将词看成由2个或更多的字母数字构成的串，标点被忽略并且被当作词分隔符 * **max\_df**：浮点数，取值范围[0.0,1.0]或整数，默认值为1.0,当构建词汇表时，词语文档频率高于max\_df，则被过滤。当为整数时，词语文档频次高于max\_df时，则被过滤。当vocabulary不是None时，该参数不起作用 * **min\_df**：浮点数，取值范围[0.0,1.0]或整数，默认为1，该参数除了指下限其他都同max\_df * **max\_features**：整数或None，默认为None。根据term frequence排序后的vocabulary的前max\_features个词作为vocabulary。如果参数vocabulary不是None，则该参数不起作用 * **vocabulary**：dict结果的词典键为词语，值为该词语在文档词矩阵中的索引；也可以是term的可迭代对象 * **binary**：布尔值，默认为False。如果为True，则所有非0词特征都被置为1。这不意味着输出只有0，1两种值，只有tf\_idf中的tf是二值的 * **dtype**：指定由fit\_transform()或transform()返回的矩阵类型 * **norm**：规范化数据的范数，’l1’，’l2’或None * **use\_idf**：布尔值，默认为True。使用逆文档频率重新加权 * **smooth\_idf**：布尔值，默认为True。通过对文档频率加1来平滑idf权值，好像有一篇包含有训练集中所有词种各1次的文档被加到了训练集中 * **sublinear\_tf**：布尔值，默认为False。应用sublinear tf值尺度变化，例如用1+log(tf)取代tf |

**属性**

|  |
| --- |
| * **vocabulary\_**：词典dict，索引：特征词的映射 * **idf\_**：数组，长度为特征数量 * **stop\_words\_**：集合set。被滤掉的词，这些词可能是（1）太多文本中包含该词(max\_df)；（2）太少文本中包含该词(min\_df)；（3）被特征选择截断(max\_features)；该属性只有在没有给定vocabulary参数的时候才有意义。注意：stop\_words\_属性可以变大，并在pickle时增加模型大小。此属性仅用于自省，可以使用delattr安全地删除或在pickle之前设置为None |

**方法**

|  |
| --- |
| * **build\_analyzer()**：返回1个可调用句柄进行预处理和分词 * **build\_preprocessor()**：在分词前，返回1个函数来预处理文本 * **build\_tokenizer()**：返回1个函数来文本分词 * **decode(doc)**：Decode输入到unicode字符串 * **fit(raw\_documents[,y])**：从训练集学习词汇表和idf * **fit\_transform(raw\_documents[,y])**：学习词汇表和idf，返回文档词矩阵 * **get\_feature\_names()**：按文档-词矩阵中词语顺序的词语列表 * **get\_params([deep])**：获取实例的参数 * **get\_stop\_words()**：构建或获取有效停用词列表 * **inverse\_transform(X)**：返回某篇训练文档向量中的非0特征值所对应的特征词列表 * **set\_params(\*\*params)**：设置实例的参数 * **transform(raw\_documents, copy=True)**：变换文档到文档词矩阵，此处使用词汇表和文档频率由fit函数或fit\_transform函数学习到 |

## 模型评价

### confusion\_matrix

**原型**

**sklearn.metrics.confusion\_matrix**(y\_true, y\_pred, labels=None, sample\_weight=None)

**参数**

|  |
| --- |
| * **y\_true**：数组，实例的实际类别序列 * **y\_pred**：数组，实例的预测类别序列 * **labels**：需要统计出的类别名称列表。如果为None则在y\_true或y\_pred中出现过的类别都将排序后作为统计类别 * **sample\_weight**：类数组，shape=样本数量，可选的 |

**返回**

|  |
| --- |
| * **C**：数组，shape=[类别数量，类别数量]。混淆矩阵 |

### classification\_report

**原型**

**sklearn.metrics.classification\_report**(y\_true, y\_pred, labels=None, target\_names=None, sample\_weight=None, digits=2)

**参数**

|  |
| --- |
| * **y\_true**：1维数组或标签指示数组/离散矩阵，样本实际类别值列表 * **y\_pred**：1维数组或标签指示数组/离散矩阵，样本预测类别值列表 * **labels**：数组shape=类别数量，需要在报告中给出的类别名称列表 * **target\_names**：字符串列表，预测类别号对应的类别名称列表 * **sample\_weight**：类数组，shape=样本数，样本权重 * **digits**：整数，分类报告中浮点数保留的小数位数，默认值为2 |

**返回**

|  |
| --- |
| * **report**：字符串，报告内容包括precision、recall、F值、宏平均macro avg、微平均micro avg |

### roc\_curve

这个实现被限制在二元分类任务上。

**原型**

**sklearn.metrics.roc\_curve**(y\_true, y\_score, pos\_label=None, sample\_weight=None, drop\_intermediate=True)

**参数**

|  |
| --- |
| * **y\_true**：数组，shape=样本数量。实例的实际类别。可取值为{0,1}或{-1,1}。如果类别标记不是二元的，则参数pos\_label应该显式给出 * **y\_score**：数组，shpae=样本数量。分类器预测分值 * **pos\_label**：整数或字符串默认为None。说明正类的标记 * **sample\_weight**：类数组，shape=样本数量，可选参数。样本权重 * **drop\_intermediate**：布尔值，可选参数，默认为True。是否放弃一些次要的点，使ROC曲线清晰 |

**返回**

|  |
| --- |
| * **fpr**：数组，每一点的假正率 * **tpr**：数组，每一点的真正率 * **thresholds**：数组。用于计算fpr和tpr的决策函数阈值 |

### auc

**原型**

**sklearn.metrics.auc**(x, y, reorder=False)

**参数**

|  |
| --- |
| * **x**：数组，fpr数组 * **y**：数组，tpr数组 |

**返回**

|  |
| --- |
| * **auc**：浮点数，auc值 |

## matplotlib

### scatter

**地址**

<https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html?highlight=scatter#matplotlib.pyplot.scatter>

<https://www.cnblogs.com/darkknightzh/p/6117528.html>

<https://matplotlib.org/gallery/color/named_colors.html#sphx-glr-gallery-color-named-colors-py>

**原型**

**matplotlib.pyplot.scatter**(x,y,s=None,c=None,marker=None,cmap=None,norm=None,vmin=None,vmax=None,alpha=None,linewidths=None,verts=None,edgecolors=None,\*,data=None,\*\*kwargs)

**参数**

|  |
| --- |
| * **x,y**： 类数组，形状为n行 * **s**：标量或类数组，形状为n行 * **c**：clolor类型，序列，color序列。颜色的标记 * 1个color格式的字符串 * color说明的长度为n的序列 * 映射为色彩的数值序列 * 2维数组，数组的每行表示1个RGB色彩 * **marker**：MarkerStyle类型。绘图标记的风格。参数值可以是MarkerStyle类实例也可以是一种特定标记风格的简写文本 |

**返回**

|  |
| --- |
| * **paths**：PathCollection用于传给legend的参数 |

颜色参数可取值图



标记类型可取值

|  |
| --- |
| '.': 'point',  ',': 'pixel',  'o': 'circle',  'v': 'triangle\_down',  '^': 'triangle\_up',  '<': 'triangle\_left',  '>': 'triangle\_right',  '1': 'tri\_down',  '2': 'tri\_up',  '3': 'tri\_left',  '4': 'tri\_right',  '8': 'octagon',  's': 'square',  'p': 'pentagon',  '\*': 'star',  'h': 'hexagon1',  'H': 'hexagon2',  '+': 'plus',  'x': 'x',  'D': 'diamond',  'd': 'thin\_diamond',  '|': 'vline',  '\_': 'hline',  'P': 'plus\_filled',  'X': 'x\_filled',  0: 'tickleft',  1: 'tickright',  2: 'tickup',  3: 'tickdown',  4: 'caretleft',  5: 'caretright',  6: 'caretup',  7: 'caretdown',  8: 'caretleftbase',  9: 'caretrightbase',  10: 'caretupbase',  11: 'caretdownbase',  'None': 'nothing',  None: 'nothing',  ' ': 'nothing',  '': 'nothing' |

### pyplot

**地址**

<https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html?highlight=pyplot#module-matplotlib.pyplot>

**方法**

|  |
| --- |
| * [**legend**](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.legend.html#matplotlib.pyplot.legend)(\*args, \*\*kwargs)：放置图例 |

# HanLP库

## 词法分析器

## 简繁转换

## 实体识别

## 短语抽取

## 文本分类