# 命令与正则

## NLTK正则相关词法分析器

## grep命令

**grep支持两种类型的正则匹配，一般的正则匹配和扩展的正则匹配。凡是表达式中用到了“+”，“？”，“\d”，“{}”时需要用egrep，其他情况用grep即可。**

### grep正则

|  |
| --- |
| # 匹配行首  **$ ls /etc | grep "^po"**  postfix  # 匹配行尾  **$ ls /etc | grep "conf$"**  asl.conf  autofs.conf  dnsextd.conf  kern\_loader.conf  man.conf  newsyslog.conf  nfs.conf  notify.conf  ntp-restrict.conf  ntp.conf  ntp\_opendirectory.conf  pf.conf  resolv.conf  rtadvd.conf  syslog.conf  # 匹配空行  **$ ls /etc | grep "^$"**  # 匹配某个词语  **$ ls /etc | grep "^ntp\.conf$"**  ntp.conf  # 匹配任意单个字符除了换行符\n  **$ ls /etc | grep "^ss."**  ssh  ssl  # 匹配0个或多个字符  **$ ls /etc | grep "^sss\*"**  ssh  ssl  # 字符集  **$ ls /etc | grep "^apache[0-9]"**  apache2 |

### egrep 正则

|  |
| --- |
| # 匹配数字  **$ ls /etc | egrep "apache\d+"**  apache2  # 匹配1个或多个字符  **$ ls /etc | egrep "^ss+"**  ssh  ssl  # 匹配0个或1个字符  **$ ls /etc | egrep "^ss?"**  security  services  shells  snmp  ssh  ssl  sudo\_lecture  sudoers  sudoers.d  syslog.conf |

### 简单示例

|  |
| --- |
| $ cat demo3.txt  ***No matter what you're look for, our motto is "keep it simple"***  ***Star by entering a basic name or word***  ***If you're looking for a place or product in a specific location,***  ***enter the name along with the town or zip code.***  **$ grep "lo\*king" demo3.txt**  If you're looking for a place or product in a specific location, |

|  |
| --- |
| $ cat demo4.txt  ***Intel:800-820-1100***  ***Abit:800-820-0323***  ***Asus:800-820-6655***  ***Sony:800-810-2228***  ***HP:8008100716***  ***IBM:800-810-1818#***  **$ egrep "800-\d{3}-\d{4}$" demo4.txt**  Intel:800-820-1100  Abit:800-820-0323  Asus:800-820-6655  Sony:800-810-2228 |

|  |
| --- |
| $ cat demo5.txt  ***020-85222213***  ***86754234***  ***800-820-1100***  ***8008100716***  ***abc123***  ***98-3876***  **$ egrep "^\d{3}" demo5.txt**  020-85222213  86754234  800-820-1100  8008100716 |

|  |
| --- |
| $ cat html.txt  ***<div class="kd-appbar"><div id='notify-box'>&nbsp;&nbsp;;<span class="notify">***  ***<span id="notify-text">&nbsp;&nbsp;</span></span></div>***  ***<div class="kd-appname-wrapper">&nbsp;&nbsp;***  ***<div class="goog-inline-block kd-appname" id="app-name">***  ***<div class="goog-inline-block goog-float-menu-button-dropdown">&nbsp;</div></div></div>***  **$ egrep "(&nbsp;){2}" html.txt**  <div class="kd-appbar"><div id='notify-box'>&nbsp;&nbsp;;<span class="notify">  <span id="notify-text">&nbsp;&nbsp;</span></span></div>  <div class="kd-appname-wrapper">&nbsp;&nbsp; |

|  |
| --- |
| $ cat ip.txt  ***202.116.3.2***  ***3.4.2***  ***10.0.0.1***  ***255.255.255.255.0***  ***256.45.2.1***  **$ egrep "(\d{1,3}\.){3}\d{1,3}" ip.txt**  202.116.3.2  10.0.0.1  255.255.255.255.0  256.45.2.1 |

## sed命令

### 基本语法

|  |
| --- |
| * sed [options] [script] ‘[address1 [, address2]] command [argument]’ [inputfiles…] * -n：取消默认输出 |

### 选择文本

|  |
| --- |
| * sed -n ‘[address1 [, address2]] p’ [inputfiles…] |

### 正则定位

正则表达式的语法如下

|  |
| --- |
| * /regexp/ |

常用正则表达式元字符如下所列

|  |
| --- |
| * 一般字符：字符本身就匹配字符本身 * \*：表示前置表达式重复0次或多次 * \+：匹配前置表达式1次或多次 * \?：匹配前置表达式0次或1次 * \{i\}：匹配前置表达式i次 * \{i,j\}：匹配前置表达式i次到j次 * \{i,\}：匹配前置表达式i次以上 * .：匹配任意字符 * ^：匹配行首字符 * $：匹配行尾字符 * []：匹配字符集 * [^]：匹配非字符集 * \n：匹配换行符 |

### 简单示例

|  |
| --- |
| # 匹配20020017的学生  $ cat students.txt  200200110 Abdul  200200164 Abram  200200167 Bartley  200200168 Bennett  200200172 Cecil  200200173 John  200200187 Cat  **$ sed -n '/^20020017/ p' students.txt**  200200172 Cecil  200200173 John |

## awk命令

目前，在绝大部分的Linux发行版中，默认安装的是gawk，即GNU awk。在许多Linux发行版中，/bin/awk命令是/bin/gawk命令的符号链接。

## vi命令

### 配置文件vimrc

|  |  |
| --- | --- |
| 显示文件名 | 设置~/.vimrc，添加set laststatus=2 |
|  |  |

### 快捷键

### 查找命令

|  |  |
| --- | --- |
| 查找斜线/ | /\/ |

## find命令

# scikit-learn

## 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 |

其属性如下

|  |
| --- |
| * 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 |

其方法如下

|  |
| --- |
| * 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://www.cnblogs.com/massquantity/p/8964029.html>

原型

classsklearn.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）

原型为：

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 |

## 数据集

### 有关数据集的工具类

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 |

### 有关文本分类聚类数据集

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

### 有关人脸识别的数据集

fetch\_lfw\_pairs 人脸数据集

fetch\_lfw\_people 人脸数据集

fetch\_olivetti\_faces 人脸数据集

### 有关图像的数据集

load\_sample\_image 图像数据集

load\_sample\_images 图像数据集

load\_digits 手写体数据集

### 有关医学的数据集

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

### 其他数据集

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

## 模型选择

模型选择部分包含有以下几个模块：模块1数据拆分类；模块2数据拆分函数；模块3超参调优类；模块4模型验证类。这里介绍模块1和模块2的两个代表函数。

1. StratifiedShuffleSplit
2. train\_test\_split

### 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。

原型为

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。

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

sklearn的特征抽取工具为后续分类或聚类提供了所需的基本操作。它的主要目的是将文本转换为数值向量。在sklearn特征抽取工具中有两个类和两个子模块，两个类分别为sklearn.feature\_extraction.DictVectorizer类和sklearn.feature\_extraction.FeatureHasher类。两个子模块分别为sklearn.feature\_extraction.image和sklearn.feature\_extraction.text。sklearn.feature\_extraction.DictVectorizer类将“特征名:特征值”映射列表转换为Numpy数组或Scipy.sparse矩阵，以便sklearn的估计器使用。sklearn.feature\_extraction.FeatureHasher类将符号名特征列表转换为Scipy.sparse矩阵，该类是对DictVectorizer和CountVectorizer两个类在内存消耗上的替代方法，例如：在嵌入式设备上运行预测代码。子模块sklearn.feature\_extraction.image汇聚了从图像抽取特征的实用工具（utilities）。子模块sklearn.feature\_extraction.text中汇聚了从文档集构建特征向量的实用工具，当前提供了4种实用工具分别为

1. sklearn.feature\_extraction.text.CountVectorizer
2. sklearn.feature\_extraction.text.HashingVectorizer
3. sklearn.feature\_extraction.text.TfidfTransformer
4. sklearn.feature\_extraction.text.TfidfVectorizer

### TfidfVectorizer

今天我们介绍另外一种词袋模型的sklearn实现，今天介绍的词袋模型特征还是由词组成，但是每篇文本的各维度向量值跟昨天介绍的[baiziyu：sklearn——CountVectorizer](https://zhuanlan.zhihu.com/p/59413389) 不一样，在CountVectorizer中，每个文本的各维度值是特征词在文本中的出现次数，今天介绍的TfidfVectorizer，每个文本的各维度值是特征词的Tfidf值。区别很明显，除了考虑特征词在文本中的出现频率外，还考虑了词语在文档集中的分布情况（也就是idf值）。TfidfVectorizer我们已经在[baiziyu：文本分类示例1——英文新闻文本分类](https://zhuanlan.zhihu.com/p/58180474) 这篇文章中应用过了，大家可以查看示例代码。从上边的介绍不难看出，TfidfVectorizer和CountVectorizer的区别不是很大，两个类的参数、属性以及方法都是差不多的，因此我们只介绍TfidfVectorizer中独有的特性，其他的请参考昨天的文章[baiziyu：sklearn——CountVectorizer](https://zhuanlan.zhihu.com/p/59413389) 。

原型为

classsklearn.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 |

### CountVectorizer

从今天开始将介绍sklearn中有关文本分类和聚类的相关类。CountVectorizer类可以构建词袋模型（计数）或one-hot模型。

classsklearn.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 |

## 模型评价

评价方法分为以下几个模块：分类评价、聚类评价等。

### 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

注意：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

计算auc值

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 |

# 隐马分词

# 最大熵词性标注

# 条件随机场实体识别

# 项目实践

## 商品标题类目预测

电商商品种类一般在千万数量级，将商品按类别进行分类是十分必要的。商品分类后不仅利于人员维护，而且有利于提高检索效率。针对这种候选类别非常多的类目预测需求，寻求一种既准确又高效还便于及时干预的类目预测方法是必要的，因此通常选择朴素贝叶斯方法。该方法比起支持向量机，感知机等2类分类模型，在训练时间以及数据均衡等问题上都表现出优势。2类分类算法在解决多类目预测问题时，在训练阶段使用n-1个2类分类器（n为类目数量），在预测阶段通过投票决定预测结果，这不仅延长了训练时间，也延长了预测时间。同时每个2类分类器的训练数据也是不平衡的，这使得原本均衡的训练数据变得不再平衡。朴素贝叶斯方法的另一个优势是原理清晰明了，易于开发人员使用自己熟悉的高效的编程语言实现训练和预测功能。针对这种千级类目数量，在训练阶段可以引入多线程技术来加快训练时间。同样在进行批量预测时，也可以利用多线程编程技术来提升预测效率。由于原理清楚，在遇到badcase时，可以快速定位到词一级别的问题，便于人工干预修正错误。

对于任何机器学习方法，已标注的大量数据是必需的。经验表明，提高数据质量，往往比换模型更有效。训练数据方面，通常电商商品的类目是已知的，虽然有一些类目下会有一小部分商品的类目标签是错误的，但可以通过关键词过滤技术在一定程度上过滤掉这些杂乱数据。一般情况下，要求数据量和类目数量呈正相关。也就是说，类目数量越多，每个类目下的数据量就要准备得更多。对于类目数量为千一级的情况，每个类目下准备万级数据量大概是合适的。在实践过程中，训练阶段没有进行商品标题去重操作，去重是否有利于提高预测准确率还有待实践证明。从表面来看，如果某些商品标题中的词语，对于本类目没有明显指征作用但对于其他类目反倒有表征作用，那么这样的商品数量较多时，势必会对分类造成干扰。

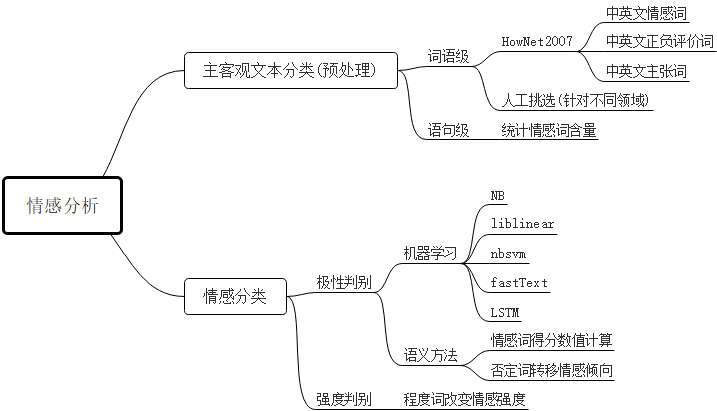
任何针对文本的分类方法，在预处理后所要解决的问题就是文本的数值表示，这一过程可以叫做嵌入表示，特征抽取。即使是时下很流行的word2vec方法，特征词的选择也是必要的。特征抽取方法一般有卡方检测法，互信息法，tf-idf法。他们的核心思想都是利用统计方法，找出与类目关联性最强的词语，滤掉对类目没有指征作用的词语。在商品类目预测实践中，往往标题中的核心物品词就是特征词语。因此前期可以通过大量人工总结得到核心物品词词典。在这里有一个实际问题，也就是含有多个核心物品词的商品标题，往往分类器无法准确预测类目标签。这种商品标题的示例有“沃尔电话电视插座网络电视接口面板开关插座网线电话电视插 86型 香槟金 **电话**电视**插座**”，“佳能 Canon 单反**相机包**200D 1500D 800D 750D 77D 80D M3/M5/M6**相机涤纶**机身附件”。可能的解决方案有两种。但是前期需要人工整理出这些含有多个核心物品词且预测错误的商品标题，构成一个易错集合。在这份数据上使用窗口短语抽取方法而不是常见的连续2gram法抽取固定搭配。窗口短语抽取法的优势在于，不仅可以抽取出连续2gram词，还可以抽取出在窗口内共现频繁的非连续词。把搭配当作一个词语处理，这样应当可以在一定程度上缓解多物品词预测不准的问题。另外一种方案是利用序列标注算法识别核心物品词。但是这种方法对于非连续搭配可能解决不好。

当类目数量很多时，势必会有相似类目出现。化繁为简的方法在这里将得到应用。也就是采用分层的方法。首先将类目按照相似性进行合并，这个过程可以通过人工方法，也可以利用聚类方法实现。此时对于首层分类器，它的类目数量将会大幅度减少，从而必然会提高分类精度。在第2层中，对每个簇训练一个分类器，实现最终的预测。

## dx类目预测

类目系统已知，类目之间互斥，类目数量33个，没有标注数据，只有1000万未标注数据。首先使用kmeans聚类出一定数量的簇，人工将簇内数据放到合适的类目下，构成训练集。

## dx情感分析

情感分析的应用场景很多，下边介绍几种。第1种电子商务方面，利用文本情感分析技术对产品评论观点进行组织和分类。第2种社会舆情分析方面，利用文本情感分析技术，可以更加及时地了解网络民意，使民间智慧与官方智慧良性互动。第3种影视评价方面，文本情感分析技术可以实现影评的自动分类，有利于用户快速浏览正反两方面的评论意见，减少观看影视时的盲目性。 第4种酒店服务评价。酒店可以根据用户的评论，了解他们的现实需求，根据需求提升酒店在相关问题上的服务质量。情感分析分为以下三个分支任务包括主客观文本分类、情感极性判别和情感强度判别。情感分析的任务可以用左图表示：第1部分主客观文本分类，它的主要任务是判断一个句子是否带有主观评价色彩。往往这样的句子中会含有特定类别的词语，因此可以从词语级进行判断。这些词语大致可以分为三类，第1类就是中英文情感词，第2类是正负评价词，第3类是中英文主张词。最后对于一句话，通过统计情感词含量来判断它是否属于主观文本。至此，第1部分的工作结束，这1部分也可以看作是系统的预处理过程，也就是说当输入的文本是客观表述时，不需要后续的处理，直接认为它的情感是中性的。第2部分就是情感分类了。在这里主要使用机器学习方法，对一句话进行正负类别预测，也就是极性判别。常见的适用于情感分类的机器学习方法有朴素贝叶斯，线性SVM，nbsvm，fastText，LSTM。另外一种针对情感分类的方法就是通过语义层面判别，主要是统计情感词数量。除了极性判别以外，还有情感强度计算，也就是通过强度词来衡量一个句子的情感强度。

### 情感词汇资源

在英文词汇资源方面，主要有General Inquirer（GI）和WordNet，其中GI词典（1966年开发）是英文文本情感分析研究中常用的基础资源之一。该词典包含182个词语类目以及11788个英语词汇。其中，有1915个词汇标注了“褒义”属性，2291个词语标注了“贬义”属性。对于一个词汇的多个义项，词典中将之作为不同条目列出，用于区分某个词语在特定义项或词性上体现的不同褒贬属性。GI的构建依赖手工标注，这些手工标注的词语情感倾向信息成为许多相关实验和研究的基础。

可用于情感词词表的构建的中文资源有以下几种：

（1）知网（HowNet）知网是一个以汉语和英语所代表的概念为描述对象，以表示概念与概念之间，以及概念所具有的属性之间的关系为基本内容的常识知识库。知网2007版情感分析用词语集提供了正面评价词3730个，负面评价词3116个，正面情感词836个，负面情感词1254个。这里的评价词是指消费者对产品及其特征发表的“肯定”或“否定”的评价，情感词是指对评论涉及的主题进行各种情感表达。

（2）《学生褒贬义词典》（张伟，2004）该词典收录了含有褒义或贬义的双音词、成语和惯用语共1672条，其中，褒义词728条，贬义词942条，兼具褒贬义的词2条。其中形容词446条，名词257条，动词393条。该词典的优点在于每个词条不仅标注了词性和褒贬色彩（褒义和贬义），还为词条标注了感情特点（赞扬、喜爱、批评、厌恶）和近义词等信息。

（3）《褒义词词典》（史继林，2005）。该词典共收录了5067个词条。

（4）《贬义词词典》（杨玲，2005）。该词典共收录了3495个词条。

获取词典资源，用于主客观评论分类

<https://download.csdn.net/download/kjp330/10106176>

获取语料

|  |  |
| --- | --- |
| 新浪微博开放平台 | <http://open.weibo.com/?sudaref=www.baidu.com&display=0&retcode=6102> |
| SemEval-2016 | <http://alt.qcri.org/semeval2016/task5/index.php?id=data-and-tools> |
| 500万条微博语料 | <https://blog.csdn.net/karamos/article/details/80132231> |
| 中文情感分析语料整理 | <https://download.csdn.net/download/xhyqlbd/10600730> |

### 模糊情感本体构建

1. 构建情感本体的目的

构建模糊情感本体目标是从语义层面明确情感术语，给出术语及其相互关系定义，实现对情感语义的理解（之后就可以利用情感本体实现句子情感分析，而不需要标注的训练集，这样可以在没有训练集的情况下，自动构建情感分析用训练集）。

1. 情感本体模型

模糊情感本体模型FEO=(B,R,E)。其中B(编号，词条，英语，词性，录入者，版本信息)，R(同义词汇集)，E(情感类型，隶属度)。下面依次说明各项意义。

编号：情感词的顺序标记

词条：词语

英语：词语对应的英文单词

词性：相同词语在不同上下文表现不同词性，也就具有了不同的情感类，进而具有不同的情感极性

录入者：便于统计和查证

版本信息：记录情感词汇来源

同义词汇集：体现本体中的关系，同义关系

情感类型：

1. 评价词大类

好和坏两个小类

1. 情感词大类

积极子类

期待、高兴、喜爱3个小类

消极子类

伤心、生气、讨厌3个小类

积极或消极子类

惊讶、焦虑2个小类

隶属度：取值范围0到1，表明评价词或情感词对于某情感类型的隶属度

1. 情感词类别自动标记及隶属度计算

这里有3种自动标注的方法，分别为点互信息法，NGD标准化谷歌距离计算法，基于语义相似度的计算方法。

1. 点互信息

统计语料来源有：微博、论坛、电商、点评的主观性文本。表示词语、共同出现的文本数量，表示出现的文本数量，表示出现的文本数量。

1. NGD标准化谷歌距离

其中，f(x)表示检索x的返回结果数，f(y)表示检索y的返回结果数，f(x,y)表示同时检索x,y返回的结果数，N表示索引的web页面数，NGD越趋于0表示x,y越相似，NGD越大表示x,y越不相似。隶属度=1-NGD(x,y)。

1. 语义相似度

刘群2002，词语语义相似度算法，实现了义原之间语义相似度的计算。史伟的隶属度计算公式如下：

语义分成4部分，第一基本义原，其他基本义原，关系义原，符号义原。，且。

史伟利用语义相似度方法对知网的情感词进行分类以及隶属度计算，得到情感本体2090个，期待170个，高兴395个，喜爱339个，惊讶65个，焦虑271个，悲伤220个，生气201，讨厌429个。

### 程度副词与情感强度

|  |  |
| --- | --- |
| 得分 | 程度词 |
| 1.5 | 最、最为、极、极为、极其、极度、极端 |
| 1.4 | 太、至、至为、顶、过、过于、过分、分外、万分、何等 |
| 1.3 | 很、挺、怪、老、非常、特别、相当、十分、甚、甚为、异常、深为、蛮、满、够、多、多么、殊、何其、尤其、无比、尤为、超、那么 |
| 1.2 | 不甚、不胜、好、好不、颇、颇为、大、大为、真的 |
| 1.1 | 稍稍、稍微、稍许、略微、略为、多少 |
| 0.9 | 较、比较、较为、还 |
| 0.8 | 有点、有些 |

### 否定副词与情感极性

|  |
| --- |
| 并非、不、不对、不再、不曾、不至于、从不、毫不、毫无、决非、绝非、没、没有、尚未、未、未必、未尝、未曾、永不、不大、不是、无 |

### 程度否定组合与情感分析

|  |  |  |  |
| --- | --- | --- | --- |
| 模式 | 模式结构 | 示例 | 计算公式 |
| 模式一 | NA+DA+EW | 不太、不大、不是很、没有那么 |  |
| 模式二 | DA+NA+EW | 太不、很不、有点不、有些不、特别不、非常不 |  |
| 模式三 | DA1+DA2+EW | 非常非常、真的很、真的十分、真的蛮 |  |
| 模式四 | NA+NA+EW | 没有不、不是不 |  |

### 标点符号与情感分析

分别规定“！”和“？”对句子情感类强度的影响为1.2倍和0.5倍。

### 6.3.7 连词与情感分析

|  |  |
| --- | --- |
| 连词类型 | 示例 |
| 转折关系 | 却、尽管、虽然、但、但是、然而、而、偏偏、只是、不过、至于、致、不料、岂知、可是 |
| 递进关系 | 不但、不仅、而且、何况、并、且、更、尤其、况且、甚至 |

规定句子中含有转折关系连词的后半句情感值为前半句情感值取反，含有递进关系连词的后半句情感值为前半句情感值加倍。

在实现上，使用了HanLP提供的情感分析代码。它基于朴素贝叶斯方法，使用卡方进行特征词选择。HanLP关于情感分析的说明文档见

<https://github.com/hankcs/HanLP/wiki/%E6%96%87%E6%9C%AC%E5%88%86%E7%B1%BB%E4%B8%8E%E6%83%85%E6%84%9F%E5%88%86%E6%9E%90>

开源的情感分析器，实际完成的都是情感极性的预测。而在这之前的主、客观语句的分类也是相当重要的一个预处理步骤。当前的研究和实践表明，基于特定领域的情感词典，使用语句包含情感词的数量来判断主、客观性是行之有效的方法之一。当然这里还可以设计一些主、客观打分公式，但限于时间有限在本次实践中并没有进行过多展开。相关研究有叶强2007提出的用于识别汉语主观短语的双词词类搭配和三词词组组合模式。Hownet2007发布了《情感分析用词语集》它是中文主客观情感分析的基本词典。林斌2007通过计算词语和词语组合的互信息量得到主观倾向词语组合75个。

跨领域情感分析是近几年的难点，其实不只是在情感分析上，其他NLP任务比如实体识别，文本分类等都表现出了领域相关性，也就是说基于特定领域的语料训练出的模型在本领域内的表现较好，但是在其他领域的效果就会下降，到目前为止在这方面的研究也没有什么新的进展。本次实践中我们使用了谭的酒店评论语料作为了训练语料，公开的情感词典作为判断主客观文本的依据，测试语料使用了我们自己的dx文本，我们发现预测出的负面评价中未登录词很多，致使一条文本中，仅有1-2个词是训练集中出现的，使得仅仅因为一个负面词就使整个句子的评分很低，我们还能发现，这些负面词单独出现时其实往往不太能表现出很强的负面色彩。因此不得不使用扩充训练语料。

## 基于增量聚类的dx话题发现

### 文本的向量化表示

增量聚类的文本向量化使用doc2vec似乎更合适，原因有（1）无法事先拿到所有需要聚类的文本，词典无法确定。（2）文本向量各维度值不好确定，特别是对于dx这样的短文本来说词语的tf值基本在1到2之间，唯一可以影响词语权值的因素主要是词语的idf值，这样的话筛选得到的词典主要去除了停用词。（3）根据万词千字规律，向量维度太大，将会导致余弦相似度计算量太大，影响速度。这些缺点是doc的优势。词语向量可以使用fasttext的中文词语向量集，也可以使用腾讯AILab开源的800万中文词语向量集<https://ai.tencent.com/ailab/nlp/embedding.html> 。

## 网页去重

## “的”、“地”、“得”误用识别与纠正

首先需要说明的是“的”、“地”、“得”误用识别与信息检索中的错别字纠正是两码事，纠错指的是输入的查询串含有错别字，而不是这里说的用法错误，这里的“的地得”字本身没有错误，错误的方面是指动词前应当使用“地”，形容词、形名词、名词前应该使用“的”，形容词、副词前应该使用“得”。

### 文本自动校对

中文文本自动校对技术的综述文献最新的也是2006年的了，说明十几年来文本自动校对技术没有太多的新方法。或许也可以从另一个角度说明，基于传统方法的文本自动校对技术基本可以满足日常的实际需要。当然一般系统的召回率在70%以上，准确率在40%以下。文本自动校对技术强依赖于词法分析技术，但是对于未登录词，以及一词多义等至今仍然是计算语言学无法完全突破的问题。下面我们对张仰森2006年发表的综述文章《文本自动校对技术研究综述》进行概括摘抄。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 发表人或机构 | 发表时间 | 核心思路 | 难点、缺点、优点 | 文章标题 |
| 张磊（微软） | 2000年 | 用winnow学习字、词和词形、长距离语言特征，使用特征预测混淆集中最可能的正确词语。 | （1）句子转特征  （2）混淆集构建 | Multifeature-based Ap- proach to Automatic Error Detection and Correction of Chinese Text |
| 李建华（哈工大） | 2001年 | 构建句子字、词候选矩阵，利用语言结构特征选择最佳候选字词序列。 | （1）结构特征表示  （2）求取最佳路径  （3）目前只适用于校对拼音输入法文本 | 多特征的中文文本校对算法的研究 |
| 易蓉湘（北师） | 1997年 | 满足校对文本规则时则标记为错误。 | （1）查错能力有限 | 计算机汉语文稿校对系统 |
| 刘挺（哈工大） | 1997年 | 分词、识别未登录词，合并单字串，剩余的不能合并的单字串认为出错。 | （1）无法纠正同音词错误  （2）捆绑规则有限，有可能误判严重 | 中文计算机辅助校对系统原理 |
| 吴岩 | 2001年 | 人机交互。 |  | 中文自动校对系统的研究与实现 |
| 张照煌 | 1994年 | 综合字符集替换+语言模型评分。 | （1）综合近似字集生成  （2）只针对别字错误，对多字、漏字、易位难以发现 | A Pilot Study on Automatic Chinese Spelling Error Correction |
| 于勐（东北大学） | 1998年 | 最长匹配法分词，发现长词错误。提取3gram，计算pre\_word,cur\_word共现频率f1与cur\_word,pro\_word共现频率f2的乘积，小于阈值则错误。对词语进行语法标注，在不可能的语法标注序列字词处作错误标记。 | （1）词语共现频率统计需要大规模不同领域的已标注语料 | 一种混合的中文文本校对方法 |
| 孙才（清华） | 1997年 | 将句子看作字段和词段，计算字段字频、字段转移概率，词段中词间字转移概率，词性转移概率，将转移概率小于阈值的字、词作为查出的错误。 |  | 汉语文本校对字词级查错处理的研究 |
| 邱超捷（北工大） | 1997年 | 构建二字结构工程，引入人名、地名辨识规则，利用词语类间接续关系进行查错。 | （1）对人名、地名误报率低 | 大规模语料库中词语接续对的统计与分析 |
| 郭志立（IBM） | 1997年 | 替换字表结合词典。 | （1）纠错建议仅限于替换字表，没有考虑上下文启发信息  （2）对漏字、多字、易位、多字替换、英文单词纠错能力弱 | 中文校对系统中的修改建议提供算法 |
| 张仰森（山西大学） | 2001年 | 基于似然匹配的纠错建议候选集产生算法。 | （1）对漏字、多字、易位、多字替换等错误类型的纠错能力有了较大提高 | 中文校对系统中纠错知识库的构造及纠错建议的产生算法 |

### 相关统计

基于词表法

如果你在编辑部工作过，那么你一定用过黑马校对软件（<http://www.bjhm.com.cn> ），该软件从出现至今已有20年历史，曾经在一次公司会议上，某领导说曾经也想跟他们合作一下，但是黑马公司就那么几个人，吃着老本，也不想太创新。从对黑马校对评测过程中，猜想它的校对方法就是基于词表的。也就是说这个软件校对出有问题的字一定是错的。因此他的准确率一定是最高的。因此，在实践中的基线系统也采用了这个方法。

问题1:进行“的”、“地”、“得”误用识别纠正的是否有必要？

收集98年和14年各1个月的人民日报语料，对收集到的语料进行初步统计。

|  |  |
| --- | --- |
| 语料字形总数 | 3800万 |
| “的”、“地”、“得”出现总数 | 110万 |
| “的”、“地”、“得”占比 | 110/380=3% |

结论：根据统计结果可以看出“的”、“地”、“得”字使用频度还是不低的，大概每100字的书面新闻报道中就有3个字是“的”、“地”、“得”，因此对书面语进行“的”、“地”、“得”误用识别还是有必要的。

问题2:对由“的”、“地”、“得”构成的2字以上词语进行误用识别是否有必要？或者说是否需要整理包含有“的”、“地”、“得”字的词语？

|  |  |
| --- | --- |
| 由“的”、“地”、“得”构成的2字以上词语总数 | 16万 |
| “的”、“地”、“得”出现总数 | 110万 |
| 占比 | 13/110=11% |

结论：由“的”、“地”、“得”构成的词语占“的”、“地”、“得”出现总次数的11%，说明在使用“的”、“地”、“得”3个字时，还是有一定比例是作为词语的。因此，对由“的”、“地”、“得”构成的词语作误用检测也是有必要的，虽然当前的输入法的智能联想功能基本上可以避免词语中单个字的错误，但是“的”、“地”、“得”构成词语的误用作为“的”、“地”、“得”误用的一个方面也是需要在进行误用识别时给予充分考虑的。

问题3:对于“的”、“地”、“得”单独成词与前后词构成3grams时出现冲突的情况，是否需要考虑到规则中？

冲突的种类有：“的地”、“的得”、“地得”、“的地得”。

|  |  |
| --- | --- |
| “的地”、“的得”、“地得”、“的地得”冲突3gram数 | 228 |
| “的”、“地”、“得”3gram总数 | 95万 |
| 占比 | 0.2% |

结论：对于使用用冲突的3grams可以考虑暂不进行误用识别，原因是他们只占了所有3grams的0.2%，牺牲的这些召回率可以为准确率提高带来帮助，因此可以忽略掉。

问题4:如何设计误用识别的规则？

|  |  |  |
| --- | --- | --- |
| “的”构成的3grams总数 | 91万 | 占比91/95=96% |
| “地”构成的3grams总数 | 3万 | 占比3/95=3% |
| “得”构成的3grams总数 | 1万 | 占比1/95=1% |

结论：即使全部预测为使用“的”字，准确率也可以达到96%。因此规则匹配的顺序设置为首先匹配“得”规则，其次是“地”规则，最后是“的”规则。

以上统计的代码

|  |
| --- |
| *# 抽取出所有的含有"的"，"地"，'得'的3grams* outfile = open(**'的地得3-grams.txt'**, **'wb'**) **with** codecs.open(**'data/dedide/rmrb.txt'**, **'rb'**, **'utf-8'**, **'ignore'**) **as** infile:  **for** line **in** infile:  line = line.strip()  **if** line:  word\_li = list(jieba.cut(line))  word\_li = [**'begin'**] + word\_li + [**'end'**]  **for** pre\_word, cur\_word, pro\_word **in** ngrams(word\_li, 3):  **if** cur\_word == **u'的' or** cur\_word == **u'地' or** cur\_word == **u'得'**:  out\_str = **u'%s\t%s\t%s\n'** % (cur\_word, pre\_word, pro\_word)  outfile.write(out\_str.encode(**'utf-8'**, **'ignore'**))  outfile.close() |

|  |
| --- |
| *# 抽取出所有的含有"的"，"地"，"得"的词语（至少2字），并输出字形总数* character\_cnt = 0 outfile = open(**'的地得word.txt'**, **'wb'**) **with** codecs.open(**'data/dedide/rmrb.txt'**, **'rb'**, **'utf-8'**, **'ignore'**) **as** infile:  **for** line **in** infile:  line = line.strip()  **if** line:  character\_cnt += len(line)  **for** word **in** jieba.cut(line):  **if** len(word) >= 2 **and** (**u'的' in** word **or u'地' in** word **or u'得' in** word):  out\_str = **u'%s\n'** % word  outfile.write(out\_str.encode(**'utf-8'**, **'ignore'**)) print(**'语料中字形总数=%d'** % character\_cnt) |

|  |
| --- |
| *# 抽取冲突* conflict\_dict = dict() **with** codecs.open(**'的地得3-grams.txt'**, **'rb'**, **'utf-8'**, **'ignore'**) **as** infile:  **for** line **in** infile:  line = line.strip()  **if** line:  cur\_word, pre\_word, pro\_word = line.split(**u'\t'**)  key\_phrase = **u'%s\t%s'** % (pre\_word, pro\_word)  conflict\_dict.setdefault(key\_phrase, [0, 0, 0])  **if** cur\_word == **u'的'**:  conflict\_dict[key\_phrase][0] += 1  **elif** cur\_word == **u'地'**:  conflict\_dict[key\_phrase][1] += 1  **elif** cur\_word == **u'得'**:  conflict\_dict[key\_phrase][2] += 1  **else**:  **pass with** open(**'的地得-3grams\_conflict.txt'**, **'wb'**) **as** outfile:  **for** key\_phrase, cnt\_li **in** conflict\_dict.items():  **if** cnt\_li[0] > 0 **and** cnt\_li[1] > 0 **and** cnt\_li[2] == 0:  out\_str = **u'的地\t%s\t%d\t%d\t%d\n'** % (key\_phrase, cnt\_li[0] + cnt\_li[1], cnt\_li[0], cnt\_li[1])  outfile.write(out\_str.encode(**'utf-8'**, **'ignore'**))  **elif** cnt\_li[0] == 0 **and** cnt\_li[1] > 0 **and** cnt\_li[2] > 0:  out\_str = **u'地得\t%s\t%d\t%d\t%d\n'** % (key\_phrase, cnt\_li[1] + cnt\_li[2], cnt\_li[1], cnt\_li[2])  outfile.write(out\_str.encode(**'utf-8'**, **'ignore'**))  **elif** cnt\_li[0] > 0 **and** cnt\_li[1] == 0 **and** cnt\_li[2] >0:  out\_str = **u'的得\t%s\t%d\t%d\t%d\n'** % (key\_phrase, cnt\_li[0] + cnt\_li[2], cnt\_li[0], cnt\_li[2])  outfile.write(out\_str.encode(**'utf-8'**, **'ignore'**))  **elif** cnt\_li[0] > 0 **and** cnt\_li[1] > 0 **and** cnt\_li[2] > 0:  out\_str = **u'的地得\t%s\t%d\t%d\t%d\t%d\n'** % (key\_phrase,cnt\_li[0] + cnt\_li[1] + cnt\_li[2], cnt\_li[0], cnt\_li[1], cnt\_li[2])  outfile.write(out\_str.encode(**'utf-8'**, **'ignore'**)) |

### 6.6.3 字符级错误规则

对于包含“的”、“地”、“得”的词语，根据观察“的地得word\_u.txt”我们可以发现，只有当词语表现出名词性时词语中的“的”、“地”、“得”一般情况下不可以相互替换。

|  |
| --- |
| **from** nltk **import** FreqDist **import** re word\_li = [] **with** codecs.open(**'的地得word.txt'**, **'rb'**, **'utf-8'**, **'ignore'**) **as** infile:  **for** line **in** infile:  line = line.strip()  **if** line:  word = line  word\_li.append(word) fdist = FreqDist(word\_li) out\_li = [] **for** word **in** fdist.keys():  **if u'的' in** word:  word1 = re.sub(**u'的'**, **u'地'**, word)  word2 = re.sub(**u'的'**, **u'得'**, word)  out\_li.append((word1, word, fdist.freq(word)))  out\_li.append((word2, word, fdist.freq(word)))  **elif u'地' in** word:  word1 = re.sub(**u'地'**, **u'的'**, word)  word2 = re.sub(**u'地'**, **u'得'**, word)  out\_li.append((word1, word, fdist.freq(word)))  out\_li.append((word2, word, fdist.freq(word)))  **elif u'得' in** word:  word1 = re.sub(**u'得'**, **u'的'**, word)  word2 = re.sub(**u'得'**, **u'地'**, word)  out\_li.append((word1, word, fdist.freq(word)))  out\_li.append((word2, word, fdist.freq(word)))  **else**:  **continue** out\_li = sorted(out\_li, key=**lambda** x: len(x[0])) **with** open(**'word\_rule.txt'**, **'wb'**) **as** outfile:  **for** wrong\_word, right\_word, freq **in** out\_li:  out\_str = **u'%s\t%s\t%.10f\n'** % (wrong\_word, right\_word, freq)  outfile.write(out\_str.encode(**'utf-8'**, **'ignore'**)) |

### 基于朴素贝叶斯方法

先说明基于“的”、“地”、“得”前后字的NB原理，然后可以根据实际的语法规则设计不同的似然概率，比如“的地得”的前后词或者后词的词性等等。以wi表示“的”，“地”，“得”中的任何一个，w-1表示wi前边的1个字，w+1表示wi后边的1个字，因此我们可以很容易写出条件概率式

我们要求的就是使这个概率最大的wi。显然证据（分母）对于所有wi都是一样的，因此问题转化为求解使分子概率值最大的wi。这里引入条件独立性得到求解公式

其中P(wi)就是“的”、“地”、“得”三个字的使用频率，（+1平滑）。想必介绍到这里你应该知道扩充词或词性的公式了。

# 附录2 HanLP使用

## HanLP简介

HanLP是完全用Java自实现的自然语言处理工具包。特点是完全用Java实现不引入第三方工具包。完全开源。中文的开源工具能做到这么完整的大概只有HanLP。包括了词法分析、句法分析、分类、聚类、关键词抽取等常见NLP应用任务。并且github上问题回答快，作者很是认真的解决大家提出的问题。虽然用Java实现，HanLP也提供了Python接口。

* **github主页地址为**[**https://github.com/hankcs/pyhanlp**](https://github.com/hankcs/pyhanlp) **，**[**https://github.com/hankcs/HanLP**](https://github.com/hankcs/HanLP) **。**
* **问题汇总页地址为**<https://github.com/hankcs/HanLP/issues> 在该网页上可以找到过往大家遇到过的问题。
* Python接口使用的示例代码页地址为<https://github.com/hankcs/pyhanlp/tree/master/tests/demos> 。

PyHanLP的安装命令为

|  |
| --- |
| # 安装命令  $ pip install pyhanlp  # 更新到最新代码包和数据包  $ hanlp update |

## HanLP的分词与词性标注

* 词标注集地址为<http://www.hankcs.com/nlp/part-of-speech-tagging.html#h2-8>
* 示例代码地址为<https://github.com/hankcs/pyhanlp/blob/master/tests/demos/demo_at_first_sight.py>
* 示例

|  |
| --- |
| In [1]: from pyhanlp import \*  In [5]: print(HanLP.segment("你好，欢迎使用HanLP汉语处理包！接下来请从其他Demo中  ...: 体验HanLP丰富的功能~"))  [你好/vl, ，/w, 欢迎/v, 使用/v, HanLP/nx, 汉语/gi, 处理/vn, 包/v, ！/w, 接下来/vl, 请/v, 从/p, 其他/rzv, Demo/nx, 中/f, 体验/v, HanLP/nx, 丰富/a, 的/ude1, 功能/n, ~/nx]  In [11]: for word in word\_li:  ...: print(word.word, word.nature)  ...:  你好 vl  ， w  欢迎 v  使用 v  HanLP nx  汉语 gi  处理 vn  包 v  ！ w  接下来 vl  请 v  从 p  其他 rzv  Demo nx  中 f  体验 v  HanLP nx  丰富 a  的 ude1  功能 n  ~ nx |

关于HanLP.segment的说明。内存要求：120MB以上，标准数据包（35万核心词库+默认用户词典）。HanLP对词典的数据结构进行了长期的优化，哪怕HanLP的词典上百兆也无需担心。HanLP.segment是一个工厂函数，它是对StandardTokenizer的封装。当前StandardTokenizer使用的是viterbi最短路分词。viterbi分词器是目前效率和效果的最佳平衡。该函数的详细代码在

<https://github.com/hankcs/HanLP/blob/master/src/main/java/com/hankcs/hanlp/seg/Viterbi/ViterbiSegment.java> 。分词大致功能有：首先生成词网和词图即可以得到粗分词网，经维特比算法找最短路径和人工干预分词后即可得到粗分结果。之后根据配置可以进行数字识别，人名识别，译名识别，地名识别，机构名识别，如果是索引分词则进行全切分分词，词性标注。

HanLP的com.hankcs.hanlp.tokenizer包中封装了很多开箱即用的分词器，但是不是所有的分词器都能在Python接口中直接使用。这些分词器有BasicTokenizer这是NGram分词器，不识别命名实体，不能使用用户词典。SpeedTokenizer这是最长匹配分词器。NotionalTokenizer这是实词分词器。StandardTokenizer当前效率和效果最佳的分词器。NLPTokenizer更精确的中文分词器。IndexTokenizer适用于信息检索的分词器。

## HanLP人工干预词表的使用

## HanLP基于朴素贝叶斯的新闻文本分类

## HanLP基于朴素贝叶斯的酒店评论情感分类

# 附录3 NLTK库

## TF-IDF计算

classnltk.text.**TextCollection**(source)

该类构建一个词语集合，可以通过文本列表加载，也可以通过包含有一个或多个的语料库（corpus）加载。这些文本列表或语料库应该支持计数，索引，搭配发现。词语的tf值计算公式tf(w)=词语w在文档集中出现次数/文档集中词形总数。idf(w)=log(文档集中文档总数/包含词w的文档数)。tf\_idf(w)=tf(w)\*idf(w)。

用法示例：

|  |
| --- |
| In [2]: from nltk.text import TextCollection  # 定义文档集  In [45]: text\_li = [['汽车','参加','春季'],  ...: ['信鸽','归巢','春季']]  # 构建TextCollection并统计idf值  In [46]: text\_collection = TextCollection(text\_li)  # 遍历文档集中所有出现的词形，打印词形idf值  In [47]: for word in text\_collection:  ...: print(word, "idf=", text\_collection.idf(word))  ...:  汽车 idf= 0.6931471805599453  参加 idf= 0.6931471805599453  春季 idf= 0.0  信鸽 idf= 0.6931471805599453  归巢 idf= 0.6931471805599453  春季 idf= 0.0  # 构建文档集词汇表  In [48]: vocabulary\_set = set([w for text in text\_li for w in text])  In [49]: vocabulary\_set  Out[49]: {'信鸽', '参加', '归巢', '春季', '汽车'}  # 构建文档集词形列表  In [51]: all\_word\_li = [w for text in text\_li for w in text]  In [52]: all\_word\_li  Out[52]: ['汽车', '参加', '春季', '信鸽', '归巢', '春季']  # 计算每个词种相对于整个文档集的tf值  In [54]: for word in vocabulary\_set:  ...: word\_tf = text\_collection.tf(word, all\_word\_li)  ...: print(word, "tf=", word\_tf)  ...:  春季 tf= 0.3333333333333333  信鸽 tf= 0.16666666666666666  汽车 tf= 0.16666666666666666  归巢 tf= 0.16666666666666666  参加 tf= 0.16666666666666666  # 计算每个词种相对于整个文档集的tf-idf值  In [55]: for word in vocabulary\_set:  ...: word\_tf\_idf = text\_collection.tf\_idf(word, all\_word\_li)  ...: print(word, "word\_tf\_idf=", word\_tf\_idf)  ...:  春季 word\_tf\_idf= 0.0  信鸽 word\_tf\_idf= 0.11552453009332421  汽车 word\_tf\_idf= 0.11552453009332421  归巢 word\_tf\_idf= 0.11552453009332421  参加 word\_tf\_idf= 0.11552453009332421 |

## 简单词语统计分布

classnltk.probability.**FreqDist**(samples=None)

用来统计实验结果的频率分布。一个频率分布实例记录了实验的每一种结果的出现次数。例如一个频率分布可以记录一个词种在一篇文档中的出现次数。

参数

|  |
| --- |
| * samples 文档词语列表或者None，默认值为None |

方法

|  |
| --- |
| * B() 返回词种总数 * N() 返回词形总数 * copy() 返回频率分布实例的复本 * freq(sample) 返回样本sample的频率 * hapaxes() 返回只出现1次的词语列表 * max() 返回最频繁的词，如果有多个词频繁程度相同，则返回其中任意一个 * pformat(maxlen=10) 返回maxlen个最频繁词的词语列表 * plot(cumulative=False) 画样本频率分布图 * pprint(maxlen=10, stream=None) 在指定的stream输出流上打印maxlen个词语 * r\_Nr(bins=None) 返回每种频次对应的样本数量的列表，bins用来计算Nr(0)的方法，默认情况下bins=self.B() * tabulate(samples, cumulative=False) 绘制samples列表中指定的词语列表的频率分布表 |

|  |
| --- |
| # 定义文本  In [**1**]: text = """第十三届尤伯杯羽毛球赛今天在名古屋进行了小组第二轮,比赛,中国队  ...: 轻松地以,战胜荷兰队,从而以两战两,胜的战绩稳获,组的出线权,中国队今天出场  ...: 的阵容与,日一样,盘单打比赛,的结果是,唐九红以,和,胜库,讷,黄华以,和,胜范迪  ...: 克,周,雷以,和,胜梅琳克,其中唐九,红第一局比赛从对手手中夺来发球权后,仅用,  ...: 分钟就连,得,分,两盘双打比赛的结果是,姚芬,赖彩勤以,和,胜库讷,范迪克,关渭  ...: 贞,史方静,以,和,胜梅琳克,博埃尔,今天另外,场比赛的结果是,英格兰队,胜丹麦,  ...: 队,印度尼西亚队,胜日本队,南朝鲜队,胜瑞,典队,组,由于中国队和南朝鲜队均是  ...: 两战两胜,荷兰队,和瑞典队均为两负,因此,实际上中国队和南朝鲜队已获,得参加  ...: 第二阶段比赛的资格,组,印度尼西亚队两战两胜,已获出线权,丹麦队,两负,出线无  ...: 望,日本队和英格兰队均是,胜,负,这个,小组另外一个出线名额,日将在这两个队中  ...: 产生,中国队,日将和南朝鲜队交锋,争夺小组第一名"""  In [**2**]: text  Out[**2**]: '第十三届尤伯杯羽毛球赛今天在名古屋进行了小组第二轮,比赛,中国队轻松地以,战胜荷兰队,从而以两战两,胜的战绩稳获,组的出线权,中国队今天出场的阵容与,日一样,盘单打比赛,的结果是,唐九红以,和,胜库,讷,黄华以,和,胜范迪克,周,雷以,和,胜梅琳克,其中唐九,红第一局比赛从对手手中夺来发球权后,仅用,分钟就连,得,分,两盘双打比赛的结果是,姚芬,赖彩勤以,和,胜库讷,范迪克,关渭贞,史方静,以,和,胜梅琳克,博埃尔,今天另外,场比赛的结果是,英格兰队,胜丹麦,队,印度尼西亚队,胜日本队,南朝鲜队,胜瑞,典队,组,由于中国队和南朝鲜队均是两战两胜,荷兰队,和瑞典队均为两负,因此,实际上中国队和南朝鲜队已获,得参加第二阶段比赛的资格,组,印度尼西亚队两战两胜,已获出线权,丹麦队,两负,出线无望,日本队和英格兰队均是,胜,负,这个,小组另外一个出线名额,日将在这两个队中产生,中国队,日将和南朝鲜队交锋,争夺小组第一名'  # 导入HanLP分词包  In [**3**]: **from** **pyhanlp** **import** \*  # 分词与词性标注  In [**4**]: word\_li = HanLP.segment(text)  # 只保留词语  word\_li = [w.word **for** w **in** word\_li]  # 去除逗号  In [**9**]: word\_li = [w **for** w **in** word\_li **if** w != ',']  # 去除单字词  In [**12**]: word\_li = [w **for** w **in** word\_li **if** len(w)>=2]  # 输出分词和预处理后的文本词语列表  In [**13**]: ','.join(word\_li)  Out[**13**]: '十三,尤伯杯,羽毛球赛,今天,名古屋,进行,小组,二轮,比赛,中国队,轻松,战胜,荷兰队,从而,战两,战绩,稳获,出线权,中国队,今天,出场,阵容,一样,单打,比赛,结果,唐九红,华以,范迪克,雷以,梅琳,其中,唐九,第一,比赛,对手,手中,夺来,发球权,分钟,双打,比赛,结果,姚芬,赖彩勤,范迪克,关渭贞,史方,梅琳,埃尔,今天,另外,比赛,结果,英格兰队,丹麦,印度尼西亚队,日本队,南朝鲜队,由于,中国队,南朝鲜队,战两胜,荷兰队,瑞典队,因此,实际上,中国队,南朝鲜队,参加,第二,阶段,比赛,资格,印度尼西亚队,战两胜,出线权,丹麦队,出线,无望,日本队,英格兰队,这个,小组,另外,一个,出线,名额,产生,中国队,南朝鲜队,交锋,争夺,小组第一'  # 导入统计分布类  In [**14**]: **from** **nltk** **import** FreqDist  # 统计输入词语列表  In [**15**]: fdist = FreqDist(word\_li)  # 返回词种总数  In [**16**]: fdist.B()  Out[**16**]: 67  # 返回词形总数  In [**17**]: fdist.N()  Out[**17**]: 94  # 返回指定词语的频次  In [**18**]: fdist['比赛']  Out[**18**]: 6  # 返回指定词语的频率  In [**19**]: fdist.freq('比赛')  Out[**19**]: 0.06382978723404255  # 返回只出现1次的词语列表  In [**32**]: ' '.join(fdist.hapaxes())  Out[**32**]: '二轮 单打 华以 阵容 稳获 史方 姚芬 名额 出场 争夺 实际上 参加 资格 第二 埃尔 小组第一 一个 十三 从而 对手 赖彩勤 发球权 交锋 战两 瑞典队 手中 唐九红 羽毛球赛 战胜 尤伯杯 雷以 丹麦队 产生 由于 轻松 关渭贞 无望 丹麦 双打 其中 阶段 进行 名古屋 一样 第一 夺来 分钟 因此 战绩 这个 唐九'  # 返回频率最高的词语  In [**21**]: fdist.max()  Out[**21**]: '比赛'  # 返回按频次降序排列的词典  In [**22**]: fdist.pformat(maxlen=10)  Out[**22**]: "FreqDist({'比赛': 6, '中国队': 5, '南朝鲜队': 4, '结果': 3, '今天': 3, '梅琳': 2, '英格兰队': 2, '范迪克': 2, '日本队': 2, '出线权': 2, ...})"  # 返回每种频次的样本数量  In [**24**]: fdist.r\_Nr()  Out[**24**]: defaultdict(int, {0: 0, 1: 51, 2: 11, 3: 2, 4: 1, 5: 1, 6: 1}) |

# 附录4 概率统计

## 随机事件

在实际社会和语言研究中，有的现象在同一条件下，并不总是出现相同的结果，这些现象称为随机现象。比如语言研究中，动词后面出现的词类，不一定就是名词，这就是**随机现象**。随机现象最简单的结果，称为基本结果，也是统计学中抽样的基本单元，称为**样本点**。比如语言研究中动词后面出现的词类，如名词、助词等都是基本结果。随机现象所有的基本结果构成的全体，称为**样本空间**或**基本空间**。语言研究中动词后面出现的词类如名词、助词等也构成一个基本空间。**事件**是样本空间的子集。在试验的结果中，随机变量X取得某一数值X=x，取得不大于实数x的值X<=x，取得区间(x1,x2)内的值x1<X<x2都是随机事件。比如语言研究中动词后面出现的名词也是一个事件，其由1种结果，即名词组成。当样本空间的子集为它自身的时候，这个子集称为**必然事件**。比如汉语语句结束的时候有语调。当样本空间的子集为空集时，称为**不可能事件**。比如汉语音节中两个辅音相连。

## 随机变量

随机变量是在试验结果中能取得不同数值的量，它的数值是随着随机试验结果而定的。由于试验的结果是随机的，所以它的取值具有随机性。定义：如果对于样本空间中的每一个样本点w，变量X都有一个确定的实数值与之对应，则变量X是样本点w的实函数，记作X=X(w)。我们称这样的变量X为随机变量。

|  |
| --- |
| 随机变量就是取值不确定的变量，但取值并非随机取，而是以一定概率取不同的值，每一种取值都对应为样本空间中的一个样本点。 |

有时试验的结果是与数量直接相关的，有时虽然试验的结果与数量无直接关系，但是也可以引入随机变量，并用随机变量取不同值的数值表示试验的结果。

随机事件的结果用数值表示时，即为**随机变量**。比如，一个文本中的句长，其值是与数值相对应的，称为数量型随机变量。另一类是其值与非数值相对应，无法计数，如作家的性别，有男、女两种性质，称为品质型随机变量，当然品质型随机变量依然可以用数值来表示。**离散型随机变量**是指随机变量的取值是有限个或可以列举的，如词长，其只能是整数，而且是可以列举的。**连续型随机变量**其值是某一区间内的所有的值的随机变量。如平均词长，其取值是平均词长可能的取值范围内的任何值。

## 概率

**概率**，又称为可能性，是概率论的基本概念，是对随机事件发生可能性的度量。**先验概率**是指无需经过实际观测即可求得的概率。如老舍《老张的哲学》一书中，总词数为60315，名词数为9959，则《老张的哲学》中一个词语是名词的概率为9959/60315=0.17。**后验概率**，又称统计概率，是指在加入已知信息后进行修正的概率。比如下边的翻译例子：

原文C：党指挥枪是我党的行动指南

候选译文E1：It is a guide to action which ensures that the military always obeys the command of the party.

候选译文E2：It is to ensure the troops forever hearing the activity guidebook that party direct.

如果从双语语料库中统计得出P(E1|C)>P(E2|C)，那么我们认为与E2比较，E1更可能是C的翻译。

## 独立与贝叶斯定理

**独立**指的是P(AB)=P(A)P(B)。比如两个不同的作者的作品中名词使用的概率，由于两个各自独立创作，因此，两者使用名词的情况必然是独立的。用下边的例子说明**贝叶斯定理**。

例：英文中有一种名为寄生间隙的句法结构，其在100K个句子中平均只出现一次。语言学家Joe构造了一种复杂的模式匹配器来识别这种句法结构。当通过匹配判断某个句子内含有一个寄生间隙时，判断正确的可能性0.95，判断错误的概率0.005。求识别出一个寄生间隙的情况下，确实是一个寄生间隙的概率。

设事件A：识别某个句子含有寄生间隙

事件B1：句子确实含有寄生间隙，P(B1)=1/100K

事件B2：句子确实不含寄生间隙，P(B2)=1-1/100K

P(A|B1)=0.95，P(A|B2)=0.005

## 概率分布与频率分布

### 概率分布（理论分布）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X | x1 | x2 | … | xn | … |
| p(xi) | p(x1) | p(x2) | … | p(xn) | … |

### 频率分布（统计分布、经验分布）

在实际观察或测量离散随机变量时，得到的是频率分布。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X | x1 | x2 | … | xl |
| fn(xi) | fn(x1) | fn(x2) | … | fn(xl) |

其中fn(xi)表示随机变量X的观测值xi出现的频率，n为观测总数，设mi为观测值xi出现的次数（称为频数），则，可按公式计算fn(xi)=mi/n。

如果观测次数n很大，则随机变量取得某一数值的频率接近相应的概率，也即统计分布大致与理论分布符合。

## 二维随机变量联合分布与边缘分布

除一维随机变量外，我们往往还需要同时考虑两个、三个或更多个随机变量构成的随机变量组，它们的值分别由两个、三个或更多个数来确定，这样的随机变量分别叫做二维、三维或多维随机变量。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| X | Y | | | | | |
| y1 | y2 | … | yn | … |  |
| x1 | p(x1,y1) | p(x1,y2) | … | p(x1,yn) | … |  |
| x2 | p(x2,y1) | p(x2,y2) | … | p(x2,yn) | … |  |
| … | … | … | … | … | … | … |
| xm | p(xm,y1) | p(xm,y2) | … | p(xm,yn) | … |  |
|  |  |  | … |  | … | 1 |

分布中每一列表示随机变量Y的一种取值，它与我们在机器学习中看到的二维表从本质上就是不同的，不要混淆，机器学习中的二维表的每一列表示的是一个随机变量，每一行表示的是一次观察得到的各随机变量的取值。

## 最大似然估计法

设总体X是离散随机变量，概率函数为，其中是未知参数。从总体X中抽取样本X1,X2,…,Xn，如果得到的样本观测值为x1,x2,…,xn，则表明随机事件{X1=x1,X2=x2,…,Xn=xn}发生了。因为随机变量X1,X2,…,Xn相互**独立**，并且与总体X有相**同**的概率**分布**函数，所以随机事件发生的概率P{X1=x1,X2=x2,…,Xn=xn}=。

设

函数称为**似然函数**，对于已取定的x1,x2,…,xn，它是未知参数的函数。

**最大似然估计法的直观想法是：如果抽样的结果得到样本观测值x1,x2,…,xn，则我们应当这样选取参数的值，使这组样本观测值出现的可能性最大，也就是使似然函数达到最大值，从而求得参数的估计值**。求解最大值的方法就是导数求极值的方法。

## 假设检验

在语言研究中，我们常常会碰到需要比较两个对象间是否有显著差异的问题。比如《红楼梦》的前八十回和后四十回在使用虚词上有没有明显不同。通过比较，可以推断二者之间是否存在真正的差异，而不是偶然的差异。这种差异性的检验，在统计学上，称为显著性检验，而这个检验推断的过程，称为**假设检验**。假设检验是统计推断中一个重要的问题。它对未知的总体分布形式或总体的未知参数做出一定的假设，然后构造适合的统计量并根据样本信息进行计算，在设定的显著水平或置信度上判断假设是否成立。

假设检验又可以分为参数假设检验和非参数假设检验。**参数假设检验**是指在总体分布已知的情况下，对未知的参数进行检验。而**非参数假设检验**，是指总体的分布未知，只能够对其分布函数类型或者某型特性进行检验。非参数假设检验主要用于总体分布未知或者总体的分布已知但并不适合运用参数假设检验的方法的场合。非参数假设检验一般不依赖于总体的分布，而是从数据本身获取所需的信息。

**卡方检验**是一种重要的非参数假设检验。主要是比较两个及两个以上的样本率（构成比）以及两个分类变量的关联性分析。其根本思想就是在于比较理论频数和实际频数的吻合程度或拟合优度问题。理论频数与实际频数差别越大，卡方值就越大，其计算的基本公式为

其中O表示实际频数，E表示理论频数。

**两总体独立性检验**。在语言研究中，常常会遇到一个对象可以同时由两个属性来刻画的情形，换言之，从总体中抽取出的样本可以由两个特征对其进行分类。比如某一词语的使用情况，既可以按照不同文体进行分类，也可以按照不同作者进行分类等等。因此，就会遇到一个问题，这两种分类之间是否有联系，也即是说，我们用于考察对象的两个特征之间是否相互独立。要检验两个特征是否独立可以使用卡方检验。

通过一个简单的例子来看假设检验问题，即判断假设的结论是否成立或成立的概率有多高。假设，在一个城市随机采样到程序员和性别的关系的数据：

|  |  |  |  |
| --- | --- | --- | --- |
| 性别/职业 | 程序员 | 非程序员 | 小计 |
| 男 | 12 | 230 | 242 |
| 女 | 2 | 245 | 247 |
| 小计 | 14 | 475 | 489 |

假设，我们的结论是程序员和性别无关，这个假设称为原假设。问：通过我们随机采样观测到的数据，原假设是否成立，或者说原假设成立的概率有多高？

卡方检验是检测假设成立与否的一个常用的工具。它的计算公式是：，其中，卡方检验的值记为，是观测值，是期望值。针对我们的例子，如果原假设成立，则程序员职业和性别无关，那么我们期望的男性程序员数量应该为(14/489)\*242=6.928个，女程序员数量应该为(14/489)\*247=7.072，同理可得到我们的**期望值**如下：

|  |  |  |
| --- | --- | --- |
| 性别/职业 | 程序员 | 非程序员 |
| 男 | 6.928 | 235.072 |
| 女 | 7.072 | 239.928 |

根据卡方检验的公式，可以算出卡方值为：

算出卡方值后，怎么判断原假设成立的概率是多少呢？这里还涉及到自由度和卡方分布的概念。简单地讲，自由度是(r-1)\*(c-1)，其中r是行数，c是列数，针对我们的问题，其自由度为1。卡方分布是指，若n个相互独立的随机变量均服从均匀分布，则这n个随机变量的平方和构成一新的随机变量，其分布规律称为卡方分布。卡方分布的密度函数和自由度相关，知道了自由度和目标概率，我们就能求出卡方值。

针对我们的问题，可以查表得到，自由度为1的卡方分布，在99%处的卡方值为6.63。我们计算出来的卡方值为6.148。由于6.148<6.63，故有99%的把握可以接受原假设。

在sklearn里，计算卡方值的函数是sklearn.feature\_selection.chi2()。除了卡方检验外，还有F值检验法，也可以用来评估特征与目标值的相关性。SelectKBest默认使用的就是F值检验算法，在sklearn里使用sklearn.feature\_selection.f\_classif来计算F值。

python示例

|  |
| --- |
| In [**1**]: **import** **numpy** **as** **np**  In [**2**]: **from** **scipy.stats** **import** chi2\_contingency  In [**3**]: d = np.array([[12, 230],  ...: [2, 245]])  In [**4**]: chi2\_contingency(d)  Out[**4**]:  (6.147863876868535,  0.01315709240127498,  1,  array([[ 6.92842536, 235.07157464],  [ 7.07157464, 239.92842536]]))  # 输出内容依次为卡方值，概率值，自由度，期望列联表 |

# 附录5 信息论

## 信息量

信息论奠基人香农(Shannon)认为"信息是用来消除随机不确定性的东西"。也就是说衡量信息量大小就看这个信息消除不确定性的程度。“太阳从东方升起了”这条信息没有减少不确定性。因为太阳肯定从东面升起。这句话信息量为0“吐鲁番下中雨”(吐鲁番年平均降水量日仅6天)这条信息比较有价值。按照统计来看吐鲁番明天不下雨的概率为98%(1-6/360)，对于吐鲁番不下雨这件事，首先它是随机不确定的，这条信息直接否定了发生概率为98%的事件。即消除不确定性的程度很大，所以这条信息的信息量比较大。信息量的表示：。信息量描述的是某一个符号的不确定性。

## 信息熵

信息量度量的是一个具体事件发生所带来的信息，而**熵**考虑的不是某一单个符号发生的不确定性，是要**考虑**这个信源**所有发生情况的平均不确定性**。若信源符号有n种U1,U2,…Un对应概率为p1,p2,…,pn且各种符号的出现彼此独立。这时，信源的平均不确定性应当为单个符号不确定性的统计平均值(E)，可称为信息熵。，单位(比特)。当某一符号的概率为零时，在熵公式中无意义，为此规定这时的也为零。当信源X中只含有一个符号x时，必定有p(x)=1，此时信源熵H(x)为零。信息熵描述的是所有可能符号的平均不确定性。

## 信息增益

从通信系统来讲，信息增益可以描述信宿收到消息后获得的关于信源不确定性减少的量。

## 互信息

描述两个事件关系的密切程度。语言学中常用来给搭配打分。

|  |  |  |
| --- | --- | --- |
| 共现概率与单独概率关系 | 互信息值 | 事件关系 |
| 共现概率>单独概率 | >0 | 密切 |
| 共现概率单独概率 | 0 | 独立 |
| 共现概率<单独概率 | <0 | 互斥 |

实践中，在一个大小为N（词语数量）的语料库中，若w与w1相距j个词处出现的次数为，w、w1单独出现的次数分别为r(w)，r(w1)，则互信息为。

互信息在实际应用中可以做文本分类时的特征词、词组抽取，可以做商品参数堆积检查，可以电商近义词抽取。下边以实际的例子进行说明。

例：对于两组候选搭配，“能力”与“弱”，“能力”与“强”进行考察。对于候选搭配“能力”与“弱”在规模为的语料库中检索得出：，，；，，。所以，。

同样，。

从两组候选搭配的强度来看，“能力”“弱”比“能力”“大”更像一个搭配。并且虽然“能力”“弱”仅共现9次，而“能力”“大”共现54次，但是“弱”仅单独出现177次，而“大”则单独出现了19913次，结果算下来，反而大于。

# 附录6 核心期刊名录

**自动化技术、计算机技术**

|  |  |  |
| --- | --- | --- |
| 期刊名称 | 复合影响因子 | 综合影响因子 |
| 计算机学报 | 5.179 | 3.003 |
| 自动化学报 | 3.976 | 2.549 |
| 软件学报 | 3.803 | 2.302 |
| 计算机研究与发展 | 2.788 | 1.887 |
| 机器人 | 2.293 | 1.397 |
| 控制与决策 | 2.225 | 1.395 |
| 计算机集成制造系统 | 1.935 | 1.099 |
| 控制理论与应用 | 1.822 | 1.065 |
| 国土资源遥感 | 1.763 | 1.167 |
| 传感技术学报 | 1.707 | 1.269 |
| 遥感技术与应用 | 1.587 | 1.107 |
| 中国图像图形学报 | 1.488 | 1.025 |
| 计算机应用研究 | 1.466 | 0.849 |
| 计算机应用 | 1.450 | 0.941 |
| 计算机辅助设计与图形学学报 | 1.435 | 0.945 |
| 信息与控制 | 1.424 | 1.061 |
| 模式识别与人工智能 | 1.411 | 0.871 |
| 智能系统学报 | 1.350 | 0.806 |
| 计算机科学与探索 | 1.235 | 0.759 |
| 计算机科学 | 1.165 | 0.706 |
| 中文信息学报 | 1.121 | 0.650 |
| 计算机工程与应用 | 1.106 | 0.665 |
| 计算机工程 | 1.061 | 0.677 |
| 计算机工程与科学 | 0.981 | 0.607 |
| 小型微型计算机系统 | 0.962 | 0.616 |
| 计算机工程与设计 | 0.915 | 0.562 |
| 遥感信息 | 0.891 | 0.556 |
| 控制工程 | 0.882 | 0.552 |
| 计算机仿真 | 0.854 | 0.658 |
| 系统仿真学报 | 0.757 | 0.491 |
| 计算机应用与软件 | 0.706 | 0.402 |
| 微电子学与计算机 | 0.664 | 0.420 |