文本分类技术基础及实践

2019年6月15日

目录

[文本分类技术基础及实践 0](#_Toc17037799)

[目录 1](#_Toc17037800)

[基础知识篇 1](#_Toc17037801)

[思维导图 2](#_Toc17037802)

[第一章 文本向量化表示 3](#_Toc17037803)

[1.1 词袋模型和文本表示方法 3](#_Toc17037804)

[1.1.1 One-Hot表示法 4](#_Toc17037805)

[1.1.2 TF表示法 4](#_Toc17037806)

[1.1.3 TF-IDF表示法 5](#_Toc17037807)

[1.2 词嵌入技术 6](#_Toc17037808)

[1.2.1 GloVe 7](#_Toc17037809)

[1.2.2 Word2Vec 8](#_Toc17037810)

[1.3 常见疑问解答 8](#_Toc17037811)

[1.2.1 任何分类模型都需要文本向量化表示吗？ 8](#_Toc17037812)

[1.2.2 向量化表示与特征选择的关系是什么？ 8](#_Toc17037813)

[1.2.3 词袋模型与词嵌入模型的对比 8](#_Toc17037814)

[1.4 sklearn实践 9](#_Toc17037815)

[第二章 文本分类技术 11](#_Toc17037816)

[2.1 分类算法 11](#_Toc17037817)

[2.1.1 多项式朴素贝叶斯 11](#_Toc17037818)

[2.1.2 贝努利朴素贝叶斯 13](#_Toc17037819)

[2.1.3 多项式与贝努利的比较 14](#_Toc17037820)

[2.1.1 支持向量机 15](#_Toc17037821)

[2.1.2 线性分类器 21](#_Toc17037822)

[2.2 多类目预测 23](#_Toc17037823)

[2.2.1 单标签分类 23](#_Toc17037824)

[2.2.2 多标签分类 23](#_Toc17037825)

[2.3 模型评价 24](#_Toc17037826)

[2.3.1 混淆矩阵 24](#_Toc17037827)

[2.3.2 精确率、准确率、召回率、F值、宏平均与微平均 25](#_Toc17037828)

[2.3.3 ROC曲线与AUC值 28](#_Toc17037829)

[2.4 模型选择 30](#_Toc17037830)

[2.4.1 交叉验证 30](#_Toc17037831)

[2.4.2 模型选择图 31](#_Toc17037832)

[2.4.3 模型选择表 31](#_Toc17037833)

[2.5 sklearn实践 32](#_Toc17037834)

[2.5.1 英文新闻文本分类 35](#_Toc17037835)

[2.5.2 英文影评情感分类 36](#_Toc17037836)

[2.5.3 英文垃圾邮件分类 42](#_Toc17037837)

[第三章 特征工程 44](#_Toc17037838)

[3.1 移除低方差特征 44](#_Toc17037839)

[3.2 单变量特征选择 45](#_Toc17037840)

[3.2.1 互信息 45](#_Toc17037841)

[3.2.2 卡方统计量 47](#_Toc17037842)

[3.2.3 频率 50](#_Toc17037843)

[3.2.4 信息增益 52](#_Toc17037844)

[3.3 递归特征消除 52](#_Toc17037845)

[3.4 基于分类模型的特征选择 52](#_Toc17037846)

[3.4.1 基本朴素贝叶斯模型的特征选择 52](#_Toc17037847)

[3.4.2 基于线性SVM的特征选择 53](#_Toc17037848)

[3.5 不同特征选择方法的比较 53](#_Toc17037849)

[3.6 多类问题特征选择 53](#_Toc17037850)

[3.7 Sklearn实践 54](#_Toc17037851)

[第四章 关键词抽取 56](#_Toc17037852)

[4.1 类目关键词抽取 56](#_Toc17037853)

[4.1.1 关键词抽取-TFIDF 56](#_Toc17037854)

[4.1.2 聚类后TF-IDF抽取簇关键词 63](#_Toc17037855)

[4.1.3 基于互信息和左右信息熵的短语抽取 68](#_Toc17037856)

[第五章 文本聚类技术 69](#_Toc17037857)

[5.1 数据规范化 70](#_Toc17037858)

[5.1.1 中心化变换 70](#_Toc17037859)

[5.1.2 极差正规化变换 70](#_Toc17037860)

[5.1.3 极差标准化变换 70](#_Toc17037861)

[5.1.4 标准化变换 71](#_Toc17037862)

[5.1.5 向量归一化 71](#_Toc17037863)

[5.1.6 矩阵归一化 71](#_Toc17037864)

[5.2 聚类算法 72](#_Toc17037865)

[5.2.1 K-Means 72](#_Toc17037866)

[5.2.2 MiniBatches 73](#_Toc17037867)

[5.2.3 Single-Pass 73](#_Toc17037868)

[5.3 简单示例 74](#_Toc17037869)

[5.3.1 K-Means示例 74](#_Toc17037870)

[5.3.2 Single-Pass示例 75](#_Toc17037871)

[第六章 神经网络 78](#_Toc17037872)

[6.1 CNN 78](#_Toc17037873)

[6.2.1 基本概念 78](#_Toc17037874)

[6.2.2 文本分类示例 79](#_Toc17037875)

[6.2 RNN 85](#_Toc17037876)

[6.3 LSTM 85](#_Toc17037877)

[6.4 Bi-LSTM 85](#_Toc17037878)

[6.5 激活函数 85](#_Toc17037879)

[6.6 初始化方法 86](#_Toc17037880)

[6.6.1 Uniform 86](#_Toc17037881)

[6.7 损失函数 86](#_Toc17037882)

[6.3.1 交叉熵 86](#_Toc17037883)

[6.3.2 KL散度 86](#_Toc17037884)

[6.3.3 对称KL散度 86](#_Toc17037885)

[6.8 正则化 86](#_Toc17037886)

[6.4.1 L1正则化 86](#_Toc17037887)

[6.4.2 L2正则化 86](#_Toc17037888)

[6.4.3 L2/L1正则原理 86](#_Toc17037889)

[6.4.4 Dropout 86](#_Toc17037890)

[6.9 优化算法 86](#_Toc17037891)

[6.5.1 SGD 86](#_Toc17037892)

[6.5.2 Momentum 86](#_Toc17037893)

[6.5.3 Adagrad 86](#_Toc17037894)

[基础实践篇 88](#_Toc17037895)

[第七章 分类训练语料构建 89](#_Toc17037896)

[7.1 语料构建导图和流程图 89](#_Toc17037897)

[7.1.1 语料构建导图 89](#_Toc17037898)

[7.1.2 语料构建机器端流程图 90](#_Toc17037899)

[7.1.3 语料构建人工端流程图 91](#_Toc17037900)

[7.1.4 机器端每日例行任务 91](#_Toc17037901)

[7.2 类目体系构建 91](#_Toc17037902)

[7.2.1 用户给定类目体系 91](#_Toc17037903)

[7.2.2 特定领域的公开类目体系 91](#_Toc17037904)

[7.2.3 语料聚类后人工制订 92](#_Toc17037905)

[7.3 类目关键词（组）和关联词（组）抽取 92](#_Toc17037906)

[7.3.1 人工列举 92](#_Toc17037907)

[7.3.2 人工从语料挑选 92](#_Toc17037908)

[7.3.3 语料聚类后抽取簇关键词 92](#_Toc17037909)

[7.3.4 扩展已有关键词 92](#_Toc17037910)

[7.3.5 词语频率法 95](#_Toc17037911)

[7.4 类目和文本向量化表示 97](#_Toc17037912)

[7.4.1 类目向量化表示 97](#_Toc17037913)

[7.4.2 文本向量化表示 99](#_Toc17037914)

[7.5 分类模型词汇表（特征词表） 99](#_Toc17037915)

[7.6 待标文本类目预测 99](#_Toc17037916)

[7.6.1 文本与类目相似度方法 99](#_Toc17037917)

[7.6.2 分类模型预测法 99](#_Toc17037918)

[7.7 半监督文本分类调研 99](#_Toc17037919)

[7.7.1 什么是半监督文本分类？ 99](#_Toc17037920)

[7.7.2 可参考的文献 100](#_Toc17037921)

[7.7.3 可参考的代码 104](#_Toc17037922)

[7.8 标注工具功能及标注方法 104](#_Toc17037923)

[7.8.1 标注工具调研 104](#_Toc17037924)

[7.8.2 标注工具应具备的功能 105](#_Toc17037925)

[7.8.3 标注方法 110](#_Toc17037926)

[第八章 文本情感分析 111](#_Toc17037927)

[8.1 短文本情感分析概述 111](#_Toc17037928)

[8.2 情感分析语料和词典资源 111](#_Toc17037929)

[8.2.1 情感词典资源 111](#_Toc17037930)

[8.2.2 语料库资源 112](#_Toc17037931)

[8.3 PyHanLP搭建短文本情感分析系统 113](#_Toc17037932)

[8.3.1 HanLP中有关情感分析的三个类 113](#_Toc17037933)

[8.3.2 训练朴素贝叶斯分类器 114](#_Toc17037934)

[8.3.3 朴素贝叶斯分类器预测 114](#_Toc17037935)

[8.4 情感分析标注 115](#_Toc17037936)

[8.4.1 情感类别 115](#_Toc17037937)

[8.4.2 情感极性 116](#_Toc17037938)

[8.4.3 语料标注 116](#_Toc17037939)

[8.4.4 词典构建 117](#_Toc17037940)

[8.4.5 标注工具 120](#_Toc17037941)

[8.4.6 标注工具应具备功能 124](#_Toc17037942)

[8.5 标注方法 124](#_Toc17037943)

[8.5.1 主张短语显示 124](#_Toc17037944)

[8.6 标注示例 124](#_Toc17037945)

[8.7 否定副词对情感极性的影响 124](#_Toc17037946)

[8.8 表转折的连词对情感极性的影响 125](#_Toc17037947)

[8.9 疑问句的情感分析 125](#_Toc17037948)

[8.10 反讽表达 125](#_Toc17037949)

[8.11 造因-情感-反应框架 125](#_Toc17037950)

[第九章 实践经验 125](#_Toc17037951)

[9.1 实践经验 125](#_Toc17037952)

[经验1 特征选择保留词语数量不能太少 126](#_Toc17037953)

[经验2 特征选择算法不是万能的 127](#_Toc17037954)

[经验3 朴素贝叶斯预测概率时防止溢出 127](#_Toc17037955)

[经验4 类目关键词构造方法 127](#_Toc17037956)

[9.2 实用脚本 128](#_Toc17037957)

[9.2.1 文本处理实用脚本 128](#_Toc17037958)

[9.2.2 文本处理Shell命令 128](#_Toc17037959)

[9.3 数据收藏 128](#_Toc17037960)

[9.3.1 Boson数据集 128](#_Toc17037961)

[9.4 常用各语言代码 128](#_Toc17037962)

[9.4.1 Java入口参数使用 128](#_Toc17037963)

[9.4.2 Java从终端读取字符串 129](#_Toc17037964)

[附录A sklearn库 129](#_Toc17037965)

[A.1 数据集 129](#_Toc17037966)

[A.1.1 数据集工具类 129](#_Toc17037967)

[A.1.2 文本数据集 132](#_Toc17037968)

[A.2 特征抽取 132](#_Toc17037969)

[A.2.1 CountVectorizer 132](#_Toc17037970)

[A.2.2 TfidfVectorizer 134](#_Toc17037971)

[A.3 特征选择 136](#_Toc17037972)

[A.3.1 VarianceThreshold 136](#_Toc17037973)

[A.3.2 chi2 138](#_Toc17037974)

[A.3.3 SelectKBest 138](#_Toc17037975)

[A.3.4 SelectFromModel 138](#_Toc17037976)

[4.1 朴素贝叶斯 139](#_Toc17037977)

[4.2.1 MultinomialNB 139](#_Toc17037978)

[4.2.2 BernoulliNB 140](#_Toc17037979)

[4.2 支持向量机 141](#_Toc17037980)

[4.3.1 LinearSVC 141](#_Toc17037981)

[4.3 线性模型 142](#_Toc17037982)

[4.4.1 SGDClassifier 142](#_Toc17037983)

[4.4 K-Means聚类 144](#_Toc17037984)

[4.5.1 Kmeans 144](#_Toc17037985)

[4.5 模型选择 145](#_Toc17037986)

[4.6.1 StratifiedShuffleSplit 145](#_Toc17037987)

[4.6.2 train\_test\_split 146](#_Toc17037988)

[4.9 模型评价 147](#_Toc17037989)

[4.9.1 confusion\_matrix 147](#_Toc17037990)

[4.9.2 classification\_report 147](#_Toc17037991)

[4.9.3 roc\_curve 147](#_Toc17037992)

[4.9.4 auc 148](#_Toc17037993)

[4.9 matplotlib 148](#_Toc17037994)

[4.9.1 scatter 148](#_Toc17037995)

[4.9.2 pyplot 150](#_Toc17037996)

[附录B HanLP库 151](#_Toc17037997)

[B.1 词法分析器 151](#_Toc17037998)

[B.1.1 实词分词器NotionalTokenizer 151](#_Toc17037999)

[B.1.2 定制用户词表 151](#_Toc17038000)

[B.1.3 定制停用词词表 152](#_Toc17038001)

[B.2 实体识别 153](#_Toc17038002)

[B.3 短语抽取 153](#_Toc17038003)

[B.4 文本分类 153](#_Toc17038004)

[B.4.1 分类模块的分词器 153](#_Toc17038005)

[B.4.2 文本分类主要模块和方法调用关系 154](#_Toc17038006)

[B.4.3 新闻文本分类示例 155](#_Toc17038007)

[B.4.4 酒店评论情感分类示例 156](#_Toc17038008)

[B.4.5 线程安全性 157](#_Toc17038009)

[B.4.6 性能指标 157](#_Toc17038010)

[B.4.7 PyHanLP卡方特征选择有问题 157](#_Toc17038011)

[B.4.8 将取对数后的预测类目概率值序列去掉对数并归一化 159](#_Toc17038012)

[附录C fastText库 160](#_Toc17038013)

[附录C.1 fastText简介 160](#_Toc17038014)

[附录D Numpy库 161](#_Toc17038015)

[D.1 loadtxt 161](#_Toc17038016)

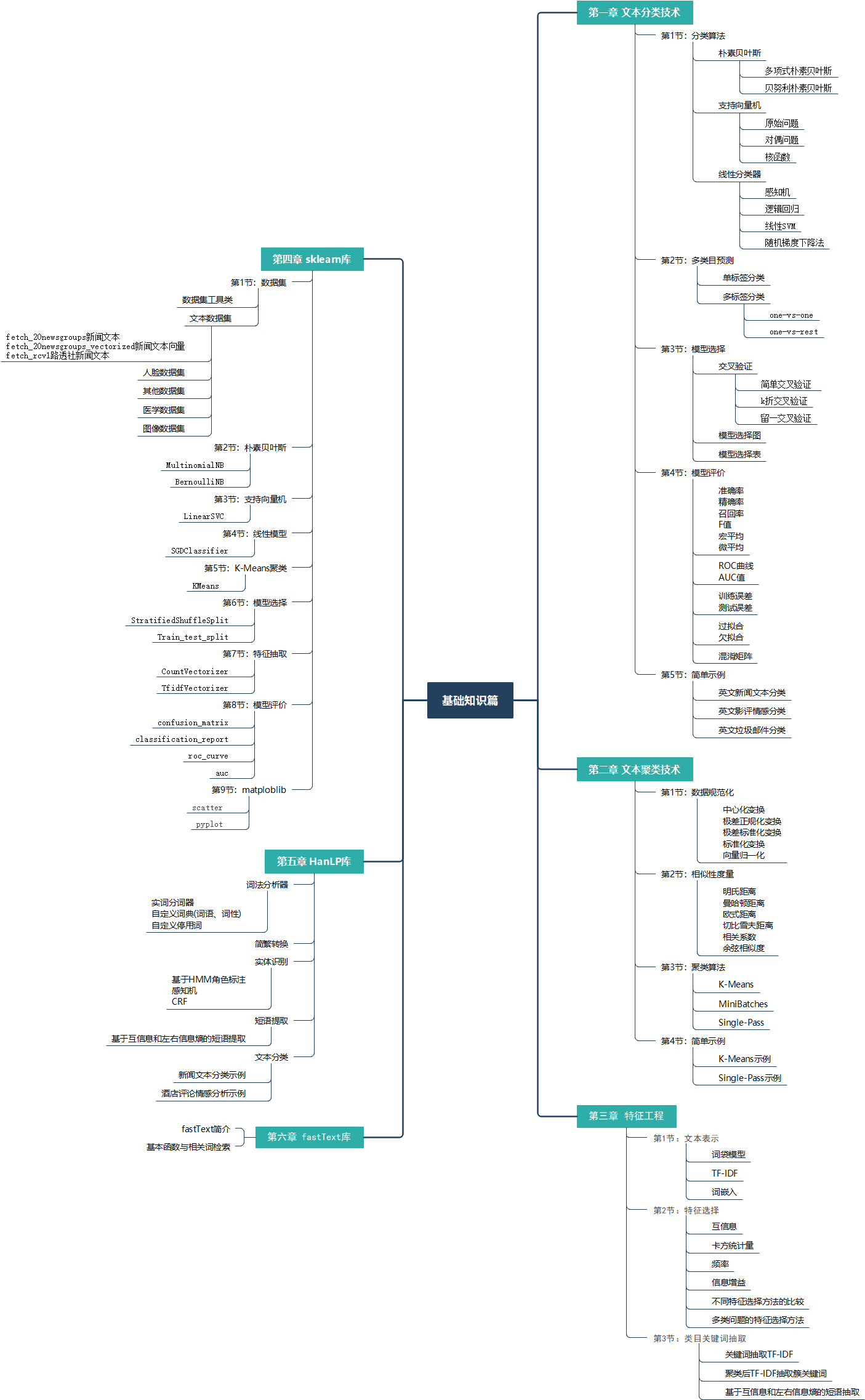
[D.2 mean 161](#_Toc17038017)

[D.3 var 161](#_Toc17038018)

[附录E Keras库 161](#_Toc17038019)

基础知识篇

# 思维导图

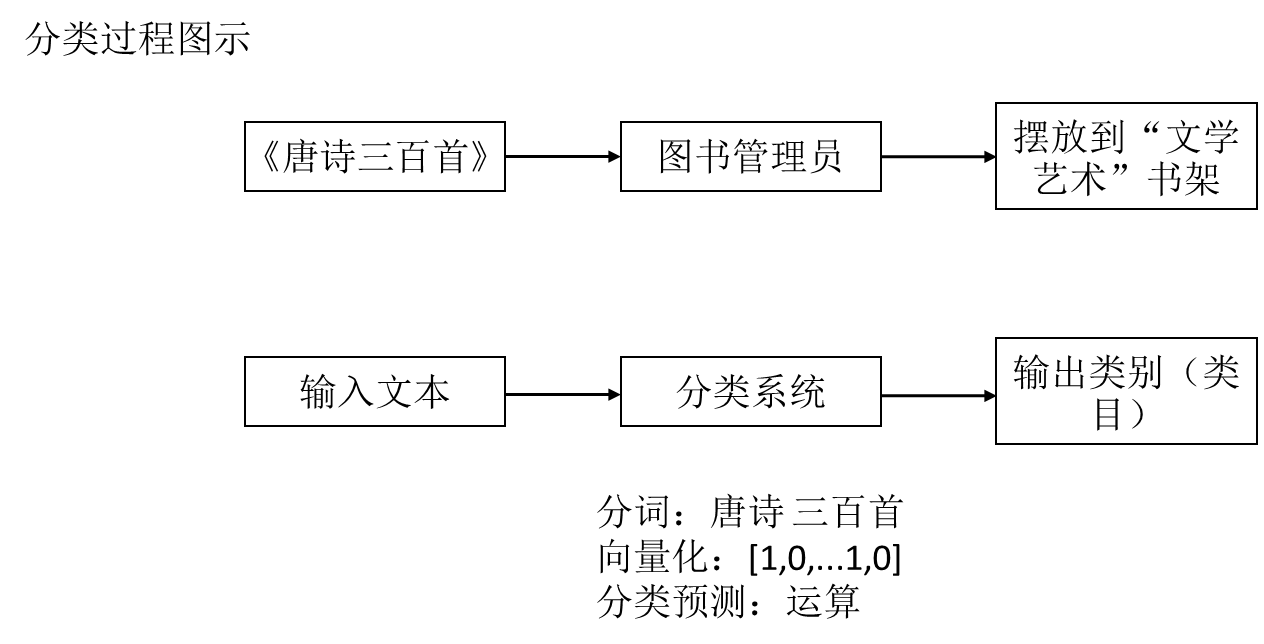


# 文本向量化表示

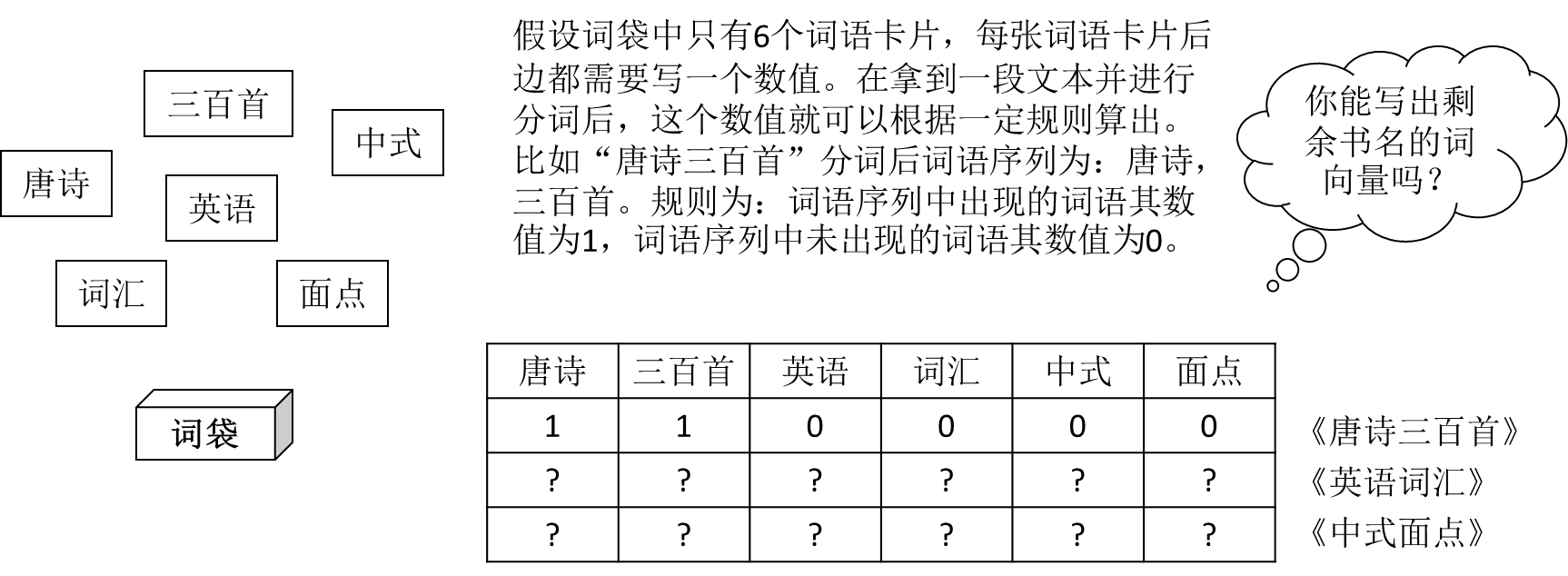
学习词袋模型要抓住两个重点，第一个重点是词袋是一张由训练语料得到的词汇表（词典），第二个重点是在给出一篇文本后，通过计算词汇表中每个词语的数值来表示文本。本章没有详述词汇表的获取，因为只要拿到训练语料，机器分词后词汇表自然就得到了。本章主要介绍如何计算词语数值，三种计算规则分别称为One-Hot表示法、TF表示法、TF-IDF表示法。本章的1.1节介绍One-Hot表示法、TF表示法、TF-IDF表示法，接着在1.2.1节说明哪些分类模型在使用前需要对文本进行向量化表示。1.2.2节说明向量化表示与特征选择的关系。1.2.3节对比说明了词袋模型与词嵌入模型的优缺点。最后在1.3节介绍了Sklearn实践部分。

## 词袋模型和文本表示方法

从书店图书管理员谈起。假设书店有3排书架，分别摆放“文学艺术”、“教育考试”、“烹饪美食”3种主题的书籍，现在新到了3本书分别是《唐诗三百首》、《英语词汇》《中式面点》，你是一名图书管理员，要怎样将这些书摆放到合适的书架上呢？实际上你摆放图书的过程就是分类的过程。如下图所示：



文本向量化表示就是用数值向量来表示文本的语义。我们人类在读一段文本后立刻就能明白它要表达的内容，如何让机器也能拥有这样的能力呢？文本分类领域使用了信息检索领域的词袋模型，词袋模型在部分保留文本语义的前提下对文本进行向量化表示。表示过程是怎样的呢？看下边的图：



### One-Hot表示法

One-Hot表示法的数值计算规则为：词语序列中出现的词语其数值为1，词语序列中未出现的词语其数值为0。用数学式子表达为：

例1 已知有下边的几篇英文文本，请用词袋模型One-Hot法向量化表示每篇文本。

|  |  |
| --- | --- |
| 文档ID | 文档词列表 |
| 1 | Chinese Beijing Chinese |
| 2 | Chinese Chinese Shanghai |
| 3 | Chinese Macao |
| 4 | Tokyo Japan Chinese |

解：1. 构建词袋。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| beijing | chinese | japan | macao | shanghai | tokyo |

2. 对于每一篇文本，按照One-Hot表示法，计算词袋中各词语的数值，得到该篇文本的向量。

Chinese Beijing Chinese的词向量为：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| beijing | chinese | japan | macao | shanghai | tokyo |
| 1 | 1 | 0 | 0 | 0 | 0 |

Chinese Chinese Shanghai的词向量为：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| beijing | chinese | japan | macao | shanghai | tokyo |
| 0 | 1 | 0 | 0 | 1 | 0 |

Chinese Macao的词向量为：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| beijing | chinese | japan | macao | shanghai | tokyo |
| 0 | 1 | 0 | 1 | 0 | 0 |

Tokyo Japan Chinese的词向量为：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| beijing | chinese | japan | macao | shanghai | tokyo |
| 0 | 1 | 1 | 0 | 0 | 1 |

从以上介绍可以看到，词袋模型的One-Hot表示法考虑了都有哪些词在文本中出现，用出现的词语来表示文本的语义。

### TF表示法

TF表示法的数值计算规则为：词语序列中出现的词语其数值为词语在所在文本中的频次，词语序列中未出现的词语其数值为0。用数学式子表达为：

其中，表示词语j，表示词语j在所在文本出现的次数。

例2 已知有下边的几篇英文文本，请用词袋模型TF法向量化表示每篇文本。

|  |  |
| --- | --- |
| 文档ID | 文档词列表 |
| 1 | Chinese Beijing Chinese |
| 2 | Chinese Chinese Shanghai |
| 3 | Chinese Macao |
| 4 | Tokyo Japan Chinese |

解：1. 构建词袋。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| beijing | chinese | japan | macao | shanghai | tokyo |

2. 对于每一篇文本，按照One-Hot表示法，计算词袋中各词语的数值，得到该篇文本的向量。

Chinese Beijing Chinese的词向量为：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| beijing | chinese | japan | macao | shanghai | tokyo |
| 1 | 2 | 0 | 0 | 0 | 0 |

Chinese Chinese Shanghai的词向量为：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| beijing | chinese | japan | macao | shanghai | tokyo |
| 0 | 2 | 0 | 0 | 1 | 0 |

Chinese Macao的词向量为：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| beijing | chinese | japan | macao | shanghai | tokyo |
| 0 | 1 | 0 | 1 | 0 | 0 |

Tokyo Japan Chinese的词向量为：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| beijing | chinese | japan | macao | shanghai | tokyo |
| 0 | 1 | 1 | 0 | 0 | 1 |

从以上介绍可以看到，词袋模型的TF表示法除了考虑都有哪些词在文本中出现外，还考虑了词语出现的频次，用出现词语的频次来突出文本主题进而表示文本的语义。

### TF-IDF表示法

TF-IDF表示法的数值计算规则为：词语序列中出现的词语其数值为词语在所在文本中的频次乘以词语的逆文档频率，词语序列中未出现的词语其数值为0。用数学式子表达为：

非平滑版：

平滑版：

例3 已知有下边的几篇英文文本，请用词袋模型TF法向量化表示每篇文本。

|  |  |
| --- | --- |
| 文档ID | 文档词列表 |
| 1 | Chinese Beijing Chinese |
| 2 | Chinese Chinese Shanghai |
| 3 | Chinese Macao |
| 4 | Tokyo Japan Chinese |

解：1. 构建词袋并计算词语IDF值

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | beijing | chinese | japan | macao | shanghai | tokyo |
| 非平滑 |  |  |  |  |  |  |
| 平滑 |  |  |  |  |  |  |

IDF值计算过程列下：

，

，

，

，

，

，

2. 对于每一篇文本，按照One-Hot表示法，计算词袋中各词语的数值，得到该篇文本的向量。

Chinese Beijing Chinese的词向量为：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | beijing | chinese | japan | macao | shanghai | tokyo |
| 非平滑 | **1\*2.386=2.386** | **2\*1=2** | 0 | 0 | 0 | 0 |
| 平滑 | **1\*1.916=1.916** | **2\*1=2** | 0 | 0 | 0 | 0 |

Chinese Chinese Shanghai的词向量为：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | beijing | chinese | japan | macao | shanghai | tokyo |
| 非平滑 | 0 | 2\*1=2 | 0 | 0 | **1\*2.386=2.386** | 0 |
| 平滑 | 0 | 2\*1=2 | 0 | 0 | **1\*1.916=1.916** | 0 |

Chinese Macao的词向量为：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | beijing | chinese | japan | macao | shanghai | tokyo |
| 非平滑 | 0 | 1\*1=1 | 0 | **1\*2.386=2.386** | 0 | 0 |
| 平滑 | 0 | 1\*1=1 | 0 | **1\*1.916=1.916** | 0 | 0 |

Tokyo Japan Chinese的词向量为：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | beijing | chinese | japan | macao | shanghai | tokyo |
| 非平滑 | 0 | 1\*1=1 | **1\*2.386=2.386** | 0 | 0 | **1\*2.386=2.386** |
| 平滑 | 0 | 1\*1=1 | **1\*1.916=1.916** | 0 | 0 | **1\*1.916=1.916** |

从以上介绍可以看到，词袋模型的TF-IDF表示法除了考虑都有哪些词在文本中出现外，还考虑了词语出现的频次以及词语在所有文本上的频率倒数。用出现词语的频次来突出文本主题，用出现词语的逆文档频率来突出文档的独特性，进而表示文本的语义。

## 词嵌入技术

词嵌入通过降维技术表征文本语料库中单词共现信息。词嵌入方法包括神经网络（通常所说的”word2vec”技术）和矩阵分解。

### GloVe

GloVe（Global Vectors for Word Representation）预训练词向量是斯坦福的开源项目，是一种基于贡献矩阵分解的词向量。GloVe全局词表示向量。它可以无监督地学习词向量表示。它通过训练词共现统计矩阵得到模型。它的训练方式使其可以展现词向量空间的线性子结构。

GloVe的官网地址为<https://nlp.stanford.edu/projects/glove/> ，

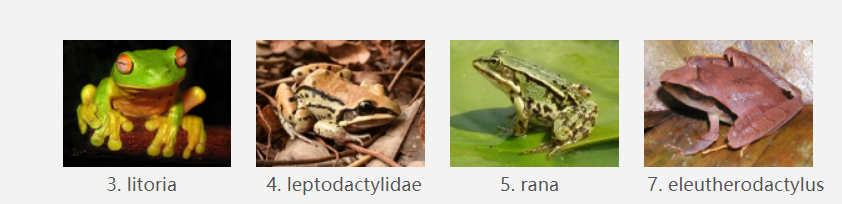
Github地址为<https://github.com/stanfordnlp/GloVe>。

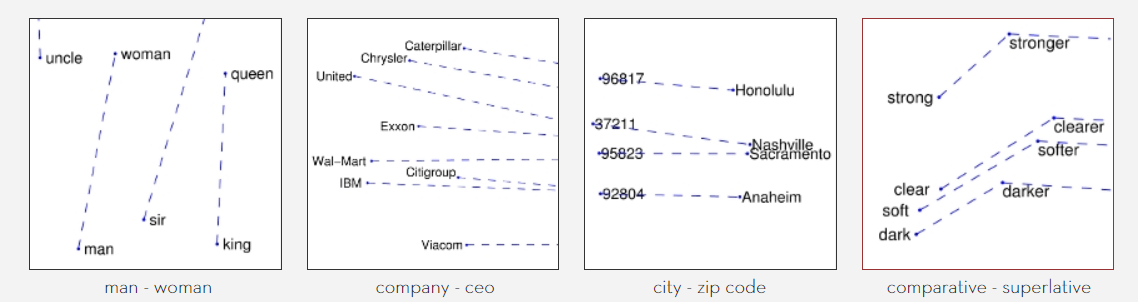
论文地址为<https://nlp.stanford.edu/pubs/glove.pdf>。

GloVe在**2014年**发部第一个版本v1.0，在**2015年**发布了第二个版本v1.2。

#### GloVe具有两个显著的特点

（1）可以查找词语最近邻，如下图所示，所有蛙类词语的相似度测度函数值很大。

（2）学习到线性子结构，如下图所示，具有相似关系的不同近义词词对之间的差向量是平行的，也就是使向量差能尽可能多的捕获近义词对周围词的意义。（个人理解）

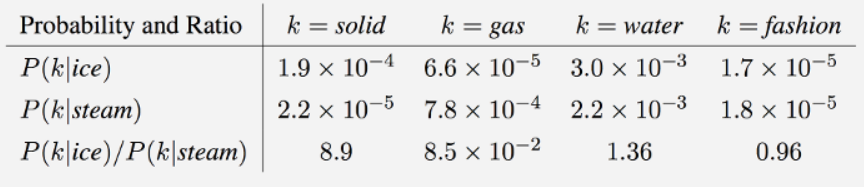


#### GloVe的训练代码和预处理代码

Glove模型要求输入为词共现计数矩阵，因此在输入语料后，预处理代码首先统计词语共现计数矩阵，之后将词共现矩阵输入到Glove核心训练代码，核心训练代码进行训练。两个代码之间是相互隔离的。

#### GloVe基本思想

Glove本质上为以加权最小二乘为目标的对数双线性模型。该模型的直观感觉是：一个词语与另外两个不同的词语共现的概率之比可以编码该词的意义。看下图



正如我们所料想的那样，ice与solid共现的频率比与steam共现的频率高，而steam与gas共现的频率比与solid共现的频率高。这两个词都与它们共有的属性water经常同时出现，而与不相关的单词fashion很少同时出现。在概率比中，water和fashion这样的非区别性词汇的比值大小适中，而能反映ice特性的词语solid对应的比值非常大，能反映steam特性的词语gas所对应的比值非常小。这样，概率之比就编码了一些与热力学抽象概念相关的粗糙形式的意义。

### Word2Vec

## 常见疑问解答

### 任何分类模型都需要文本向量化表示吗？

不是的。朴素贝叶斯模型不需要将文本向量化表示。因为朴素贝叶斯模型记录的是词语的条件概率值，只要对输入各词语的条件概率值进行计算即可得到预测数值。线性分类模型需要对文本进行向量化表示。因为线性模型记录的是线性表达式系数，必须输入一个数值向量才能计算得到预测数值。

### 向量化表示与特征选择的关系是什么？

文本的向量化表示是用数值来表达文本语义的，这一过程中会引入失真，也就是失去文本的主题。虽然One-Hot、TF、TF-IDF一步步减少失真突出文本主题，但是依然不够，因此可以借助特征选择方法筛选出对于区分类别主题有帮助的词语。

### 词袋模型与词嵌入模型的对比

词袋模型使用一组词语序列来表示一段文本，这一组词语序列就是词袋模型，也可以叫词汇表。词汇表中每个词语之间的顺序是任意的，但是一旦词汇表确定后词语之间的前后顺序就不能变化了。由于词语之间的顺序任意，所以词袋模型忽略了文本的语法和语序要素。词袋模的One-Hot表示法、TF表示法、TF-IDF表示法的数值计算规则都没有考虑词语之间的共现关系。比如“的”字的后边只能接名词性词语，“地”字的后边只能接动词性词语。显然词袋模型无法表示词语之间的共现关系，也就是说词袋模型认为一个词出现的可能性与其他词出现的可能性无关，词语的出现是相互独立的。词袋模型最大的缺陷是向量的维度高，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。由于词袋模型表示的文本向量的每个维度都代表一个词语，因此可以用聚类后簇中心向量的具有较大值的维度对应的词语来作为簇的关键词。词嵌入模型是浅层神经网络的副产品，在用浅层神经网络做文本分类时，发现在得到分类结果的同时，输入矩阵刚好可以用来表示词语，由于词语是用上下文来表达的，因此在一定程度上反映了词语的语义，但这并不是机器真正明白了词语的意义，而只是相似的词语之间向量相似度大而已。它的优点：1.不需要人工参与可以得到，大厂推出了各种词向量库，词汇量一般都是百万级。2.可以表达一个词语的向量，这是前几年办不到的。3. 文本的表示维度降下来了，一般只需100到300维之间。缺点也很明显：1.强烈依赖分词特别是未登录词识别技术，也就是说新出现的词没有对应向量。2.不是真正明白词语意思，对一词多义无能为力，只能表达最通用的语义。3.有些词在语义上没有近义关系，但因为经常出现在相同的上下文中，向量的相似度也很大。有些概念抽象的词（特别时文本类别名称词），按照相似度检索到的近义词之间不具有语义相关性。4. 每个维度不再对应一个特征词语，后续排查分类badcase时不方便。

## sklearn实践

Sklearn的特征抽取模块可以从原始数据中抽取特征。目前该模块提供了图像和文本特征抽取类。文本的特征抽取类可以从原始文本中抽取出词语特征，特征数据格式满足所有机器学习算法对输入数据格式的要求。请注意特征抽取与特征选择的区别。特征抽取是将文本数据转换成适合机器学习的数值特征而特征选择是一种应用于数值特征的机器学习技术。

例4 请用sklearn对下边的文本进行词袋模型向量化表示。

|  |  |
| --- | --- |
| 文档ID | 文档词列表 |
| 1 | Chinese Beijing Chinese |
| 2 | Chinese Chinese Shanghai |
| 3 | Chinese Macao |
| 4 | Tokyo Japan Chinese |

解：

|  |
| --- |
| In [1]: texts = ['Chinese Bejing Chinese',  ...: 'Chinese Chinese Shanghai',  ...: 'Chinese Macao',  ...: 'Tokyo Japan Chinese']  In [2]: from sklearn.feature\_extraction.text import CountVectorizer  # One-Hot表示法  In [3]: cv = CountVectorizer(binary=True)  In [4]: document\_vec = cv.fit\_transform(texts)  # 词袋  In [5]: cv.get\_feature\_names()  Out[5]: ['bejing', 'chinese', 'japan', 'macao', 'shanghai', 'tokyo']  In [6]: document\_vec.toarray()  Out[6]:  array([[1, 1, 0, 0, 0, 0],  [0, 1, 0, 0, 1, 0],  [0, 1, 0, 1, 0, 0],  [0, 1, 1, 0, 0, 1]], dtype=int64)  # TF表示法  In [7]: cv = CountVectorizer()  In [8]: document\_vec = cv.fit\_transform(texts)  In [9]: document\_vec.toarray()  Out[9]:  array([[1, 2, 0, 0, 0, 0],  [0, 2, 0, 0, 1, 0],  [0, 1, 0, 1, 0, 0],  [0, 1, 1, 0, 0, 1]], dtype=int64)  # TF-IDF表示法（未平滑）  In [10]: from sklearn.feature\_extraction.text import TfidfVectorizer  In [11]: tv = TfidfVectorizer(use\_idf=True, smooth\_idf=False, norm=None)  In [13]: tv\_fit = tv.fit\_transform(texts)  In [13]: tv.get\_feature\_names()  Out[13]: ['bejing', 'chinese', 'japan', 'macao', 'shanghai', 'tokyo']  In [14]: tv\_fit.toarray()  Out[14]:  array([[2.38629436, 2. , 0. , 0. , 0. ,0. ],  [0. , 2. , 0. , 0. , 2.38629436,0. ],  [0. , 1. , 0. , 2.38629436, 0. ,0. ],  [0. , 1. , 2.38629436, 0. , 0. ,2.38629436]])  # TF-IDF表示法（平滑）  In [15]: tv = TfidfVectorizer(use\_idf=True, smooth\_idf=True, norm=None)  In [16]: tv\_fit = tv.fit\_transform(texts)  In [17]: tv\_fit.toarray()  Out[17]:  array([[1.91629073, 2. , 0. , 0. , 0. ,0. ],  [0. , 2. , 0. , 0. , 1.91629073,0. ],  [0. , 1. , 0. , 1.91629073, 0. ,0. ],  [0. , 1. , 1.91629073, 0. , 0. ,1.91629073]]) |

# 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。

朴素贝叶斯具有多项式朴素贝叶斯和贝努利朴素贝叶斯两种形式。本小节介绍多项式朴素贝叶斯，下一小节介绍贝努利朴素贝叶斯。

#### 训练阶段

（1）先验概率计算

（2）条件概率

其中，c表示某个类别，表示某个词语。

|  |
| --- |
| 心得：  条件概率表示的是词在类别c中出现的概率。  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）加入平滑

考虑到预测文本时会出现未登录词，使用下面的平滑方法来计算条件概率。平滑的基本思想是：适当减小登陆词的条件概率，把减小的概率分给未登录词。

拉普拉斯平滑（加1平滑）

加1平滑采用均匀分布作为先验分布，即先假定每个词项在每个类别中出现一次，然后再计算登陆词的条件概率，这样一来，**未登陆词的条件概率值为 1/词汇表长度**。

#### 预测阶段

预测阶段需要计算待预测文本属于每个类目的概率，将概率值最大的类目作为待预测文本的类目。用数学公式表示如下：

由公式式可以看出，预测阶段需要求和的项数为：**待预测文本所含词语个数+1**。

例1 给定下边的训练和测试语料，判断待预测文本是否属于China类。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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 | ? |

**训练阶段**：

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

（2）条件概率

类别c=China文档集中词语总数为8个，分别为：

Chinese Beijing Chinese Chinese Chinese Shanghai Chinese Macao

类别c!=China文档集中词语总数为3个，分别为：

Tokyo Japan Chinese

|  |  |  |  |
| --- | --- | --- | --- |
| 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**。

**预测阶段：**

（1）待预测文档词语列表为：

Chinese Chinese Chinese Tokyo Japn

（2）计算文本属于China和不属于China的概率

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。

### 贝努利朴素贝叶斯

#### 训练阶段

（1）先验概率

（2）条件概率

（3）拉普拉斯平滑（加1平滑）

未登录词的条件概率值为 **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2 给定下边的训练和测试语料，判断待预测文本是否属于China类。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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 | ? |

**训练阶段：**

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

（2）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**。

**预测阶段：**

（1）待预测文档词语集合

Chinese Tokyo Japn

（2）计算文本属于China和不属于China的概率

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

|  |
| --- |
| 心得：  采用贝努利模型预测长文本时，往往会因为某个词在文档中出现一次而将文本预测为某一类，比如例子中的“China”这个词。从贝努利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。 |

### 多项式与贝努利的比较

|  |  |  |
| --- | --- | --- |
|  | 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

学习支持向量机首先要明确它是一个线性模型，所谓线性指的是系统的方程为N元一次方程，如果数据特征数量为2个，也就是用两个词语来表示文本，则系统方程为二元一次方程。学习过初中数学就会知道我们求解的都是二元一次方程组，如果只有一个二元一次方程是有无穷多解的。如果我们绘出二元一次方程的曲线，则它应当为平面上的一条直线，直线的角度由斜率（也就是w）决定，直线的位置由方程的截距（也就是b）决定，至此我们给出支持向量机的系统表达式wx+b=0。w为系数向量，b为截距值。再来看分类问题，怎样把一个直线方程应用到分类问题呢？变等式为不等式，也就是说wx+b>0时，认为实例x向量属于正类，wx+b<0时，认为实例x向量属于负类。大于0和小于0所表述的是一个数值范围，怎样对应到两类的类别标签呢？需要引入一个函数，这个函数输入为负数时输出一种值，输入为正数时输出一种值，显然sign符号函数可以满足这种要求，因此支持向量机的决策函数为sign(wx+b)。模型预测过程在下边图示：

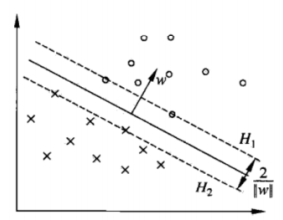
支持向量机分类法就是要在训练集中找到具有**最大类间隔**的决策平面。这也是它与感知机的区别。

由于支持向量机算法只能预测输出两种类别状态，因而对于多类分类问题通常需要建立多个支持向量机分类器。

此外，将分界面定在最大类间隔的中间，对于许多情况来说不是最优的。

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。

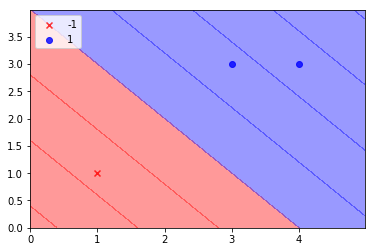
**对偶问题可用SMO算法求解**

**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

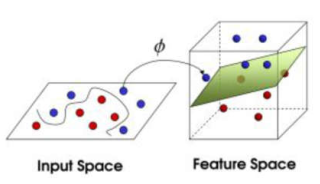
s.t.

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。

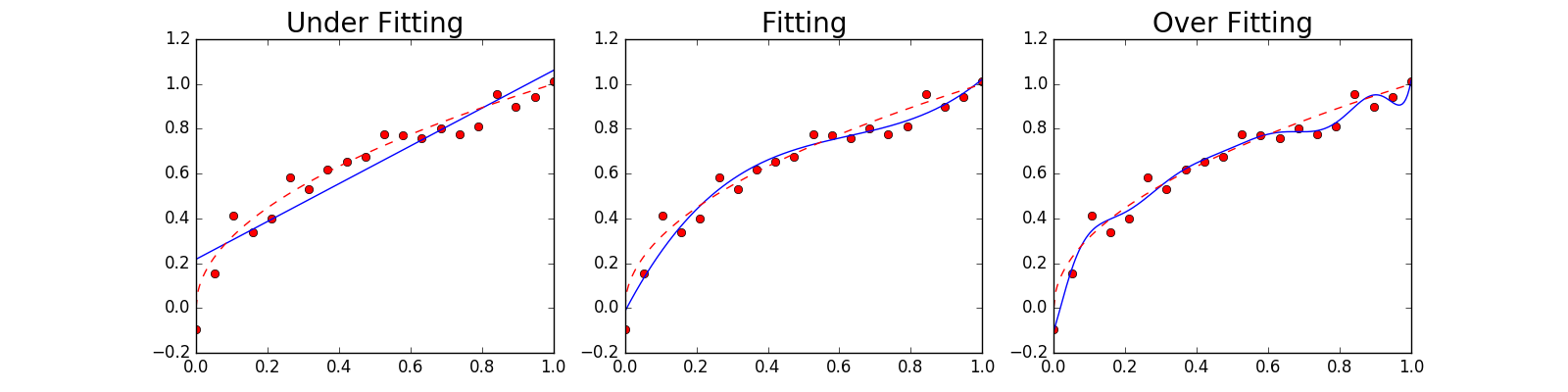


从图中可以看到原来的实例点在一个2维空间上，显然在这里只能使用一条曲线将两类实例点分离到曲线两侧。那么怎么才能用一个线性超平面将两类实例点分开呢？答案是使用非线性函数，将2维空间中的实例点映射到3维空间，也就是图中的feature space，这时就可以在3维空间内找到一个平面，将两种实例分离到平面的两侧。需要说明的是，当把数据映射到高维空间后，每一个实例点都需要更多的数值来表示，这样在计算对偶公式中的内积时会使计算量增大，这是不能忍受的，怎样解决这个问题呢？根据mercer定理，针对某一种可以找到一个核函数K，用这个核函数K在低维空间中计算出某两个实例的值，这个值刚好就是这两个实例在高维空间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

#### 过拟合与欠拟合

**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。

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() |

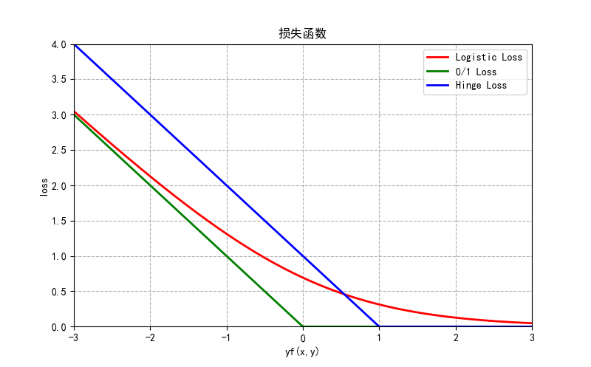
### 线性分类器

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

## 模型评价

### 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

**sklearn实践**

这里在iris数据集上训练一个线性SVM分类器，并用混淆矩阵来评价。分别输出了未经规范化和经过规范化的混淆矩阵，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。

|  |
| --- |
| # 导入鸢尾花数据集  from sklearn import datasets  iris = datasets.load\_iris()  # 鸢尾花数据矩阵  X = iris.data  # 各条数据的类别索引label  y = iris.target  # 各label对应的类别名称序列  class\_names = iris.target\_names  # 分解为训练集和测试集  from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)  # 使用linear SVC分类  from sklearn import svm  classifier = svm.SVC(kernel='linear', C=0.01)  y\_pred = classifier.fit(X\_train, y\_train).predict(X\_test)  **# 计算混淆矩阵**  **from sklearn.metrics import confusion\_matrix**  **cm = confusion\_matrix(y\_true=y\_test, y\_pred=y\_pred)**  **# 混淆矩阵打印函数**  **def print\_confusion\_matrix(cm, classes, normalize=False):**  **if normalize:**  **cm = cm.astype('float')/cm.astype('float')/cm.sum(axis=1)[:, np.newaxis]**  **print("%-20s" % "", end='')**  **for i in range(len(classes)):**  **print("%-20s" % classes[i], end='')**  **print()**  **for i in range(cm.shape[0]):**  **print("%-20s" % classes[i], end='')**  **for j in range(cm.shape[1]):**  **print("%-20.2f" % cm[i,j], end='')**  **print()**  print\_confusion\_matrix(cm, class\_names, normalize=False) |

输出结果

|  |
| --- |
| setosa versicolor virginica  setosa 13.00 0.00 0.00  versicolor 0.00 10.00 6.00  virginica 0.00 0.00 9.00 |

### 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正确预测的覆盖程度。

#### 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。AUC是一个比精确率更好的评价不平衡分类问题的性能指标，它利用真正率和假正率来考虑类别数据的不平衡性，它的一个缺点是无法扩展到多值分类器。

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。在评价不平衡分类问题时，通常将数据少的类别作为正类，数据多的类别作为负类。

|  |
| --- |
| 我的心得：实际上ROC曲线和AUC值，就是综合考虑了正类上的召回率和负类上的召回率。曲线越靠左上说明分类器在正负类别上的召回率都比较高。 |

首先给出二类分类的混淆矩阵

|  |  |  |
| --- | --- | --- |
|  | 实际为正 | 实际为负 |
| 预测为正 | 真阳性TP | 假阴性 FN |
| 预测为负 | 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。

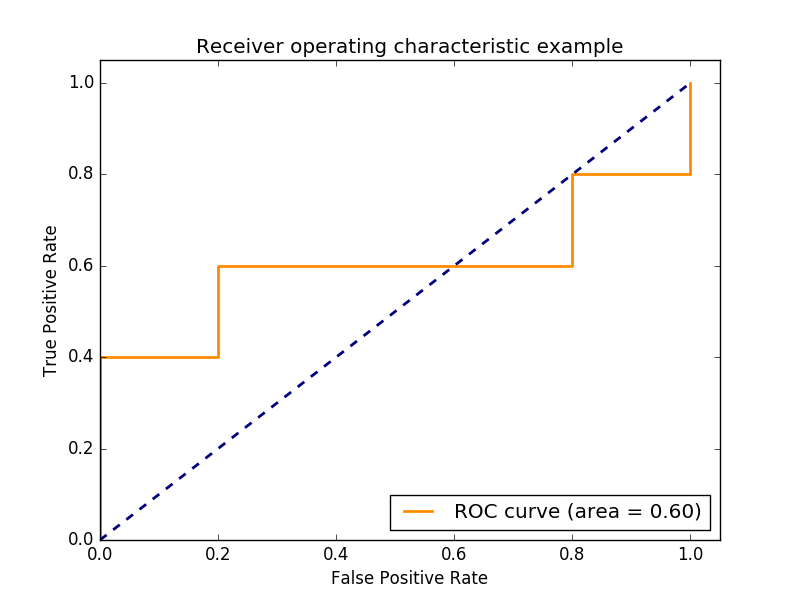
1. 让模型对每一个实例进行预测，记录正类得分，并按得分将实例升序排列。
2. 从排序列表中按顺序选择第1个得分最小的记录，从该记录开始到列表结束的所有记录都被指定为正类，其他实例指定为负类，计算混淆矩阵并计算TPR，FPR。此时，TPR=FPR=1。
3. 从排序列表中选择下1个记录，从该记录开始到列表结束的所有记录都被指定为正类，其他实例指定为负类，计算混淆矩阵并计算TPR，FPR
4. 重复步骤（3），直到列表中所有实例都被选择过。
5. 以FPR为横轴，TPR为纵轴，描点绘制ROC曲线。

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>

## 模型选择

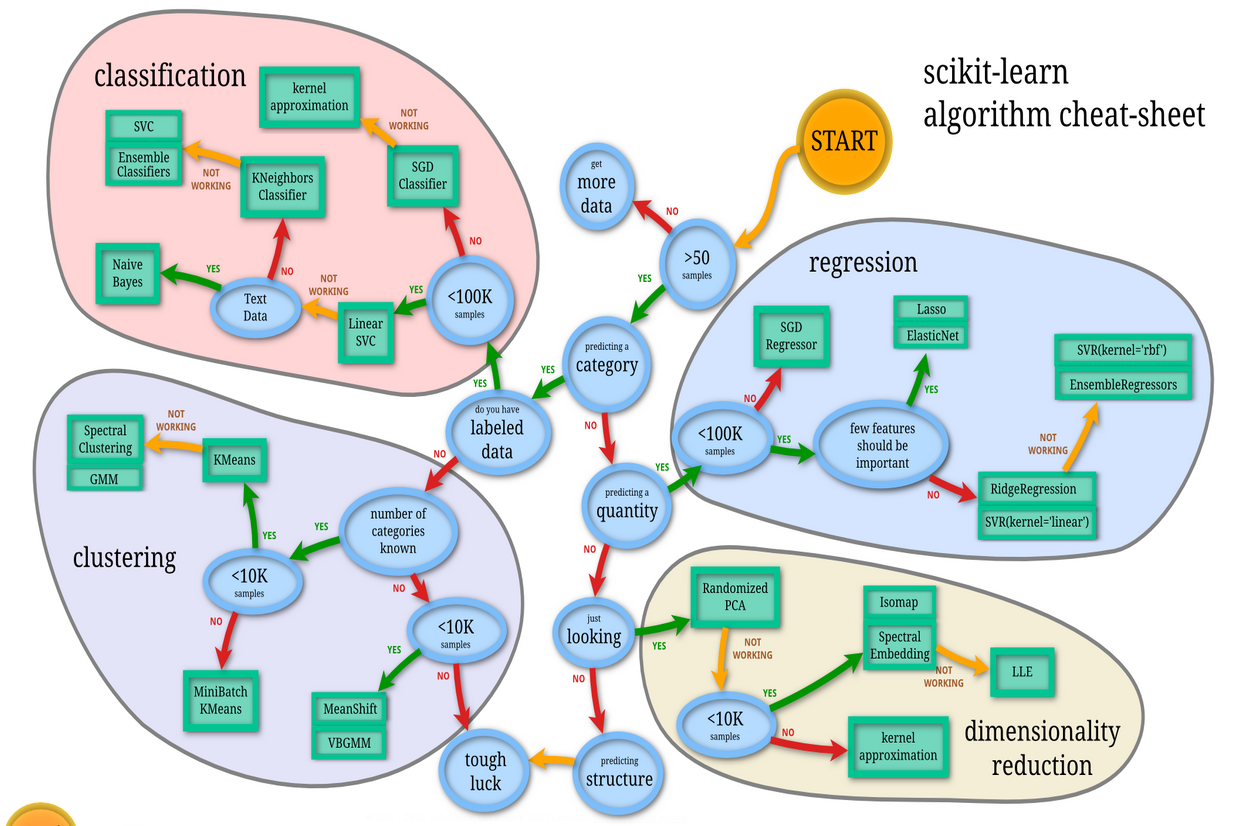
### 交叉验证

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 |

## sklearn实践

例：请用sklearn重新计算例1中待预测文本属于给类目的概率。

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。如果要查看每个词语在不同类目条件下的概率值可以查看属性feature\_log\_prob\_ 。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!=China)的过程。

（1）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'] |

（2）文本向量化表示

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

（3）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和多项式系数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!=China) = 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 |

**显然截距表示的是非China类别的先验概率，系数表示的是词语在非China类别条件下的概率。**

（4）表示测试集

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

（5）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。

（6）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]]) |

例2：请用sklearn重新计算例1中待预测文本属于给类目的概率。

|  |
| --- |
| # 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]]) |

与我们在例2中计算的结果是一致的。

### 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)))** |

### 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) |

# 特征工程

## 移除低方差特征

低方差特征指的是特征词在各个实例中的取值基本相同，这样的特征对于区分各实例的不同没有帮助，因此可以移除。看下边的示例：

例1 已知某一数据集如下，移除方差值小于0.16的特征。

解：分别计算3个特征的总体方差，公式为：

平均数：

总体方差：

计算结果为：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 第1列特征 | 第2列特征 | 第3列特征 |
| **总体方差** | **0.166667** | **0.266667** | **0.3** |

因此，移除第1列特征，保留后两列特征。

移除低方差特征来进行特征选择方法的优点是不需要类别信息。

## 单变量特征选择

单变量特征选择通过单变量统计检验方法选择最优特征。基于单变量统计检验进行特征选择的方法，常作为模型训练开始前的预处理步骤。sklearn通过下边类的对象来进行特征选择：

* SelectKBest 保留top-k特征。
* SelectPercentile 保留特征得分值与最大特征得分值之比满足阈值的特征。

上边的两个对象在实例化时需要传入计算特征得分的函数方法，对于分类问题常用的方法有：

* chi2 卡方。
* f\_classif F检验。
* mutual\_info\_classif 互信息。

注意：F检验只能评价两个随机变量的线性相关性。互信息则可以捕获任何种类的统计相关性。两种计算得分的方法都需要大量标注样本作为条件。

### 互信息

从通信角度补充一些信息论的基础知识。信息论是在**信息**可以度量的基础上，研究**有效地**和**可靠地**传递信息的科学。1941年~1944年香农用概率论的方法研究通信系统，揭示了通信系统传递的对象就是信息，并对**信息**给以科学**的定量描述**，提出了信息**熵**的概念。指出通信系统的中心问题是在噪声下如何**有效**且**可靠**地传送信息以及实现这一目标的主要方法是**信道编码**。这一成果于1948年以《A mathematical theory of communication》为题发表。信源编码的研究落后于信道编码，香农1959年在文章《Coding theorems for a discrete source with a fidelity criterion》中系统地提出了信息率失真理论，它是**数据压缩**的数学基础，为各种**信源编码**的研究奠定了基础。显然信源编码是为了尽可能减少数据的大小，提高通信系统的**有效性**。各种压缩软件所使用的方法就是信源编码。

1. 自信息量

自信息量反应的是某一条消息发生的不确定性，在通信系统中用来描述信源发出**一种符号**的不确定性。这种**不确定性**是和**概率**有关的，而且不确定性程度越大（出现概率越小），则收到这种符号所得到信息量越大，也就是说这种符号所具有的**信息量**越大。考虑到上边的这些特性得到下边自信息量的定义式：

其中表示的特性是：不确定性程度越大，消息发生的概率越小。

1. 离散信源熵

首先解释一下什么是离散信源，它就是能它发出一个**符号集合**的机器，可以把它理解为一个装满了球的盒子，球的颜色种类是有限可列的，那么从盒子里摸一个球出来就是发出了一个符号。自信息量描述发出一个符号收到后得到的信息量，那么怎么度量一个信源所能提供的信息量呢，或者说怎么度量一个信源的不确定性呢。当然很容易想到求每种符号自信息量的平均值。当然这里的每个样本值是自信息量而不是一个数值，因此需要在自信息量前乘以一个符号出现概率的权值，于是得到**描述一个离散信源不确定性的度量定义式**（熵的定义）：

从上边的定义实际上熵就是各符号信息量的数学期望。

熵的应用（1）训练决策树时，对于每一层节点的属性选择，利用了分裂前后数据集的不确定性变化，也就是说一个数据集经过某个属性分裂后所能发出的类别的不确定性。当然越向下生长希望数据集发出类别的不确定性越小。（2）最大熵原理中的熵。实际上我们描述信源时就是用的它的概率分布，那么最大熵就是找到使信源熵最大的概率分布参数。用信源这种物理事务来说概率分布是不是会感觉更容易摸得着。

事件自信量与信源熵的相同点和不同点

两者都是用来度量事件发生不确定性的测度。不同的是事件自信息量度量单个事件的不确定性大小，而信源熵度量一个事件集合不确定性的大小。

1. 互信息
2. 熵的性质

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)，由这两个公式即可得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。词项与类别的互信息定义式为：

用MLE估计概率值后，得到互信息公式为：

其中，表示的是词项（）且类别（）的文档数，有下边四种不同情况：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2对文本分类训练语料中的每个类目进行特征选择。

部分Python代码如下

|  |
| --- |
| 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 |

选择结果

|  |  |  |
| --- | --- | --- |
| Medical  治疗 0.079  患者 0.048  农民 0.036  市场 0.035  经济 0.034  疗效 0.034  病人 0.033  医院 0.030  医疗 0.029  农村 0.026 | Sports  经济 0.157  运动员 0.117  女子 0.090  部门 0.083  效益 0.077  规模 0.076  体育 0.075  方式 0.074  投资 0.072  国家 0.071 | Agriculture  农业 0.594  农民 0.274  经济 0.272  农村 0.233  市场 0.187  粮食 0.181  土地 0.181  产量 0.154  面积 0.148  价格 0.147 |
| Education  学生 0.075  学校 0.074  教师 0.066  农业 0.064  国家教委 0.044  农民 0.035  教育 0.034  中小学 0.034  市场 0.031  经济 0.026 | Electronics  电子 0.042  集成电路 0.032  半导体 0.023  电子产品 0.021  刘桂莲 0.019  计算机 0.017  经济 0.017  电子部 0.015  电脑 0.012  存在 0.012 | Communication  通信 0.060  通信网 0.037  邮电 0.033  电话 0.022  光缆 0.019  用户 0.019  电信 0.015  卫星通信 0.015  进一步 0.011  交换机 0.011 |

“运动”类目中只有2篇文本含有“经济”这个词，说明互信息特征选择法容易选择出罕见词。

### 卡方统计量

卡方特征选择法的数学原理是卡方检验。卡方检验是一种常用的假设检验方法，它的无效假设H0是：两个随机变量相互独立。如果两个随机变量的卡方值为0，那么观察频数与期望频数没有差别，假设成立，于是两个随机变量相互独立；如果两个随机变量的卡方值大于0，那么观察频数与期望频数有差别，假设不成立，于是两个随机变量以一定概率存在关系（1-p值）

在词项特征选择中，两个随机变量分别为**词项的出现与否和类别的出现与否**，卡方的定义式为：

其中，表示词项出现与否，表示类别出现与否（文本实例是否属于类别），表示训练语料中词项取值为x且类别取值为y的文本数（观察次数），表示词项取值为x且类别取值为y的期望文本数（期望次数）。期望的计算式为：

其中，表示文本总数，表示词项取值为的文本数，表示类别取值为的文本数。

需要注意的是

（1）定义中两个随机变量是词项的出现与否以及类别的出现与否，所以在使用sklearn对文本词项进行特征选择时，我们应该使用的是**One-Hot表示法**，而不是任何其他的向量化方法。

（2）自由度指的是不受限制的变量个数，自由度=(行数-1)\*（列数-1），这里类别也作为一列参与计算。之所以引入自由度的概念是因为通过卡方值和自由度查表可以得到 p值，即原假设成立，两个变量间独立的概率。

在实际计算中我们经常用四格相依表来求解卡方值，四格相依表如下所示：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 文本属于类别c | 文本不属于类别c | 合计 |
| 文本中出现t |  |  |  |
| 文本中未出现t |  |  |  |
| 合计 |  |  |  |

四格相依表中的元素值表示满足行和列所述条件的的文本数。有了四格相依表，计算公式如下：

式子中N11表示包含词项t且属于类别c的文本数，N10表示包含词项t且不属于类别c的文本数，N00表示不包含词项t且不属于类别c的文本数，N01表示不包含词项t且属于类别c的文本数。上边的公式可以用下边的Python代码计算：

|  |
| --- |
| 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)  if fenmu == 0:  return 0  return fenzi\*1.0/fenmu |

需要注意的是：卡方计算式中分母不能为0，如果分母为0，则一定存在取值全相同的词项列，也就是说词项在每篇文本中都出现，此时该词项的卡方值为0。

值度量的是期望值E和观察值N的偏离程度。值大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。值的取值范围为。

**例3** 假如有下边训练语料，请分别计算词项“beijing”、“chinese”、“japan”与类别的互信息值。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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 |

解：首先计算出训练集的向量化矩阵并添加类别列

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| beijing | chinese | japan | macao | shanghai | tokyo | label |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 |

由上表计算各词语与类别的四格相依表，并按照卡方公式计算卡方值。

beijing与类别的卡方值计算

|  |  |  |  |
| --- | --- | --- | --- |
|  | china | ~china | 合计 |
| beijing | 1 | 0 | 1() |
| ~beijing | 2 | 0 | 3() |
| 合计 | 3() | 1() | 4 |

再用定义式计算一遍：

chinese与类别的卡方值计算

|  |  |  |  |
| --- | --- | --- | --- |
|  | china | ~china | 合计 |
| chinese | 3 | 1 | 4 |
| ~chinese | 0 | 0 | 0 |
| 合计 | 3 | 1 | 4 |

再用定义式计算一遍：

japan与类别的卡方值计算

|  |  |  |  |
| --- | --- | --- | --- |
|  | china | ~china | 合计 |
| japan | 0 | 1 | 1 |
| ~japan | 3 | 0 | 3 |
| 合计 | 3 | 1 | 4 |

再用定义式计算一遍：

从该例题我们可以看出，由于卡方计算时只考虑了词语在多少篇文本中出现，于是它忽略了词语在一篇文档中的频次，也就是说卡方法会忽略词语在文本中的频次TF值。在例子中“chinese”在文本1和文本2中都出现了2次，说明“chinese”对于类别“china”还是有指征作用的，但是“chinese”的卡方值却为0，很容易在特征选择时被剔除掉。我们也应该看到，对于短文本来说在一个文本中高频词是很少的，因此这种问题可以忽略。

### 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4对文本分类训练语料中的每个类目进行特征选择。

部分Python实现代码如下：

|  |
| --- |
| 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 |

选择结果

|  |  |  |
| --- | --- | --- |
| Medical  治疗 0.510  研究 0.431  专家 0.333  中国 0.294  医疗 0.294  患者 0.275  医院 0.275  组织 0.255  达到 0.255  发现 0.255 | Sports  中国 0.483  北京 0.422  运动员 0.340  冠军 0.333  选手 0.320  获得 0.306  中国队 0.293  体育 0.279  女子 0.272  成绩 0.265 | Agriculture  农业 0.952  经济 0.780  研究 0.679  农民 0.670  水平 0.665  农村 0.663  我国 0.658  市场 0.644  国家 0.641  要求 0.593 |
| Education  教育 0.695  学校 0.644  工作 0.542  学生 0.492  培养 0.441  教师 0.424  学习 0.424  加强 0.390  国家 0.373  北京 0.339 | Electronics  电子 0.630  市场 0.444  公司 0.407  计算机 0.407  报道 0.370  中国 0.370  我国 0.333  投资 0.333  国家 0.333  集成电路 0.296 | Communication  通信 0.720  我国 0.520  中国 0.440  邮电 0.400  通信网 0.400  用户 0.400  北京 0.360  公司 0.360  电话 0.360  邮电部 0.320 |

从结果可以看到词语文档频率的长尾现象，所谓**长尾**指的是频率的分布画在图上后像一条立起来的眼镜蛇，只有头部抬起，后边拖着一条长长的尾巴。如下图：

例2至例4的实践过程中，我们分批多次向停用词表中添加了停用词，才得到目前的选择结果。但是在实际应用中光靠添加停用词有时可能还是略显麻烦，也可以从另一个角度来做，即先确定一张词汇表，分词后只保留词汇表中登陆的词语。

### 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。

## 递归特征消除

## 基于分类模型的特征选择

### 基本朴素贝叶斯模型的特征选择

以二分类多项式朴素贝叶斯模型为例说明特征选择的原理。通过前边朴素贝叶斯sklearn实践部分我们可以看到，对于二分类问题来说，sklearn中只训练一个线性分类器f(x)=wx+b，这里显然w是一个一维的向量，它是分类器的参数，x是待预测文本的TF法向量化表示，b表示该分类器所预测的两个类别中的一个类别的出现概率。显然**w各维表示的词语的似然概率p(w|c)，p(w|c)的值越大说明类别c产生这个词语的可能越大，也就是说这个词语更能表征这个类别。**通过上边的分析，我们不难想到，可以通过设一定的系数值阈值来实现特征选择，对于系数值大于设定阈值的词项予以保留，对于系数值小于设定阈值的词项予以去除，实现了特征选择的目的。

例5 已知有下边的训练文本，试用朴素贝叶斯模型特征选择法筛选特征，阈值设定为所有系数的均值。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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 |

解：本题为二分类问题，因此我们训练一个多项式朴素贝叶斯模型。词汇表以及每个词语的似然概率列于下：

|  |  |
| --- | --- |
| 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 |

系数均值=(-1.504-2.197-2.197-2.197-1.504-1.504)/6=-1.850，所有大于等于该阈值的特征词予以保留，于是选出的特征词有chinese、tokyo、japan。

### 基于线性SVM的特征选择

## 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。

## 多类问题特征选择

前边我们介绍了互信息、卡方、信息增益等特征选择方法在二类分类问题中的计算，本节介绍这些特征选择方法在多类分类问题中的计算。看下表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 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看作一个二类分类问题，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最终的效用指标。

## Sklearn实践

例 假定我们有一个布尔型的数据矩阵，请用sklearn的VarianceThreshold移除80%以上的样本值为True或False的特征。数据矩阵如下：

解：

|  |
| --- |
| In [1]: from sklearn.feature\_selection import VarianceThreshold  In [2]: import numpy as np  In [3]: X = np.array([[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1, 1]])  In [4]: X  Out[4]:  array([[0, 0, 1],  [0, 1, 0],  [1, 0, 0],  [0, 1, 1],  [0, 1, 0],  [0, 1, 1]])  # 实例化方差特征选择法实例  In [5]: sel = VarianceThreshold(threshold=(.8 \* (1 - .8)))  # 计算各属性方差并进行特征选择  In [6]: sel.fit\_transform(X)  Out[6]:  array([[0, 1],  [1, 0],  [0, 0],  [1, 1],  [1, 0],  [1, 1]]) |

例 假如有下边训练语料，请使用sklearn分别计算词项“beijing”、“chinese”、“japan”与类别的互信息值。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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 |

解：

sklearn的chi2只计算四格相依表中的某两项，观测次数的计算方法是通过构建矩阵后，执行得到观测值向量的，举个例子如果属性列，类别列为，则

，

期望值的计算方法是对按行求和得到feature\_count向量，对按行求均值得到类别概率向量class\_prob，用计算对应于上边观测值的期望值。对于上边例子的期望值计算过程如下：对属性列按行求和得到，对按行求均值得到，计算期望值。于是得到。

|  |
| --- |
| In [1]: from sklearn.feature\_selection import chi2  # 定义文本  In [2]: texts = ['Chinese Bejing Chinese',  ...: 'Chinese Chinese Shanghai',  ...: 'Chinese Macao',  ...: 'Tokyo Japan Chinese']  ...:  # 定义类别序列  In [3]: y = [1, 1, 1, 0]  # One-Hot法表示文本  In [4]: from sklearn.feature\_extraction.text import CountVectorizer  In [5]: cv = CountVectorizer(binary=True)  In [6]: cv\_fit = cv.fit\_transform(texts)  # 打印特征词名称以及向量化后的文本矩阵  In [7]: cv.get\_feature\_names()  Out[7]: ['bejing', 'chinese', 'japan', 'macao', 'shanghai', 'tokyo']  In [8]: cv\_fit.toarray()  Out[8]:  array([[1, 1, 0, 0, 0, 0],  [0, 1, 0, 0, 1, 0],  [0, 1, 0, 1, 0, 0],  [0, 1, 1, 0, 0, 1]], dtype=int64)  # 打印类别序列  In [9]: y  Out[9]: [1, 1, 1, 0]  # 计算各特征词列的chi2值，返回卡方值和p值  In [10]: chi2(cv\_fit.toarray(), y)  Out[10]:  (array([0.33333333, 0. , 3. , 0.33333333, 0.33333333,  3. ]),  array([0.56370286, 1. , 0.08326452, 0.56370286, 0.56370286,  0.08326452])) |

# 关键词抽取

## 类目关键词抽取

类目关键词抽取最直接的方法是找到相应类目的术语词典。也可以用网络抓取技术抓取相应类目的文本用本节介绍的方法抽取类目关键词。

### 关键词抽取-TFIDF

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。

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 """ 关键词抽取tf-idf法 用法：python 类目关键词抽取tf.py 文件名 每个类目最大关键词数量 要求：python3，sklearn，PyHanLP 说明：输入文件中每一行存储一个类目的所有文本。 程序会统计每个词项的tf-idf值，这里的idf指的逆类目频率， 并输出每个类目的按tf-idf值降序的topx个词语，x由第2个参数决定默认为10 """* **import** codecs **from** pyhanlp **import** \* **from** sklearn.feature\_extraction.text **import** TfidfVectorizer **from** nltk **import** ngrams  *# 加载实词分词器 参考https://github.com/hankcs/pyhanlp/blob/master/tests/demos/demo\_notional\_tokenizer.py* Term = JClass(**"com.hankcs.hanlp.seg.common.Term"**) NotionalTokenizer = JClass(**"com.hankcs.hanlp.tokenizer.NotionalTokenizer"**)   *# 通用预处理（训练语料和预测语料通用）* **def** preprocess(text):  *# 如果抽取词语则n\_grams=1，如果抽取2grams则n\_grams=2* n\_grams = 2  *# 全部字母转小写* text =text.lower()  word\_li = []   *# NotionalTokenizer.segment中有去除停用词的操作* **for** term **in** NotionalTokenizer.segment(text):  word = str(term.word)  pos = str(term.nature)  *# 去掉时间词* **if** pos == **u't'**:  **continue** word\_li.append(word)   *# 去掉单字词（这样的词的出现有可能是因为分词系统未登录词导致的）* **if** n\_grams == 1 **and** len(word) == 1:  **continue** word\_li.append(word)   *# 如果只是分词则直接返回word\_li即可* **if** n\_grams == 2:  ngrams2\_li = [**u'\_'**.join(w) **for** w **in** ngrams(word\_li, 2)]  word\_li = ngrams2\_li  **return** word\_li   **def** extract\_keyword(text\_li, topx=10):  *"""  用tf-idf法抽取每个类目的关键词* **:param** *text\_li: 类目文本类表，每个元素表示一个类目的所有文本串* **:param** *topx: 每个类目抽取出的关键词数量* **:return***: 返回每个类目的关键词序列  """* tv = TfidfVectorizer(analyzer=preprocess)  tv\_fit = tv.fit\_transform(text\_li)  vsm = tv\_fit.toarray()  category\_keywords\_li = []  **for** i **in** range(vsm.shape[0]):  sorted\_keyword = sorted(zip(tv.get\_feature\_names(), vsm[i]), key=**lambda** x:x[1], reverse=**True**)  category\_keywords = [w[0] **for** w **in** sorted\_keyword[:topx]]  category\_keywords\_li.append(category\_keywords)  **return** category\_keywords\_li   **def** main():  input\_file\_name = sys.argv[1]  **if** len(sys.argv) == 3:  topx = int(sys.argv[2])  **else**:  topx = 10  **with** codecs.open(input\_file\_name, **'rb'**, **'utf-8'**, **'igonre'**) **as** infile:  text\_li = infile.readlines()  category\_keywords\_li = extract\_keyword(text\_li, topx)  print(category\_keywords\_li)   **if** \_\_name\_\_ == **"\_\_main\_\_"**:  main() |

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 |

### 聚类后TF-IDF抽取簇关键词

|  |
| --- |
| *#coding:utf-8 """ 关键词抽取tf-idf法 要求：python3，NLTK，PyHanLP，fastText 输入：默认一篇文本占一行 """* **from** time **import** time **from** fastText **import** load\_model **import** numpy **as** np **from** pyhanlp **import** \* **import** sys **import** codecs **from** py.类目关键词抽取\_tfidf **import** extract\_keyword  *# 实词分词器实例* Term = **None** NotionalTokenizer = **None** *# fastText模型* fasttext\_model = **None** *# 存储所有文本向量的矩阵* text\_vec = **None** *# 存储每篇文本的主题序号* topic\_serial = **None** *# 当前拥有的主题数量* topic\_cnt = **None** *# 每个主题中的文本数量* topic\_cnt\_dict = **None** *# 预处理后的文本列表* preprocessed\_data = **None** *# 系统初始化* **def** init():  **global** fasttext\_model  **global** text\_vec  **global** topic\_serial  **global** topic\_cnt  **global** topic\_cnt\_dict  **global** Term  **global** NotionalTokenizer  **global** preprocessed\_data   *# 读取fastText词语向量矩阵* fasttext\_model = read\_fasttext\_data(**'../dictionary/cc.zh.300.bin'**)  *# 初始化文本向量矩阵* text\_vec = np.array([])  text\_vec.resize((0, fasttext\_model.get\_dimension()))  *# 初始化文本话题编号序列* topic\_serial = []  *# 初始化话题数量* topic\_cnt = 0  *# 初始化每个主题中的文本数量变量* topic\_cnt\_dict = dict()  *# 加载实词分词器 参考https://github.com/hankcs/pyhanlp/blob/master/tests/demos/demo\_notional\_tokenizer.py* Term = JClass(**"com.hankcs.hanlp.seg.common.Term"**)  NotionalTokenizer = JClass(**"com.hankcs.hanlp.tokenizer.NotionalTokenizer"**)  *# 初始化文本列表* preprocessed\_data = []   *# 通用预处理（训练语料和预测语料通用）* **def** preprocess(text):  *# 全部字母转小写* text =text.lower()  word\_li = []   *# NotionalTokenizer.segment中有去除停用词的操作* **for** term **in** NotionalTokenizer.segment(text):  word = str(term.word)  pos = str(term.nature)  *# 去掉时间词* **if** pos == **u't'**:  **continue** *# 去掉单字词（这样的词的出现有可能是因为分词系统未登录词导致的）* **if** len(word) == 1:  **continue** word\_li.append(word)   **return** word\_li   *# 读取fastText词语向量矩阵* **def** read\_fasttext\_data(file\_path):  t0 = time()  fasttext\_model = load\_model(file\_path)  t1 = time()  print(**"加载fastText向量库时间%.2fs"** % (t1-t0))  **return** fasttext\_model   *# 计算句子的单位向量* **def** compute\_sentence\_vector(word\_li):  **global** fasttext\_model   *# 初始化句子向量* sen\_vec = np.array([])  sen\_vec.resize((1, fasttext\_model.get\_dimension()))  *# 在fastText中登陆的词语列表* has\_vec\_word\_li = []  **for** word **in** word\_li:  *# 词语有向量值* **if** fasttext\_model.get\_word\_id(word) != -1:  has\_vec\_word\_li.append(word)  word\_vec = fasttext\_model.get\_word\_vector(word)  sen\_vec += word\_vec  **if** len(has\_vec\_word\_li) != 0:  sen\_vec /= len(has\_vec\_word\_li)  *# 单位化句子向量* sen\_vec /= np.linalg.norm(sen\_vec)  **return** sen\_vec, has\_vec\_word\_li   *# SinglePass文本聚类* **def** single\_pass(sen\_vec, sim\_threshold=0.6, max\_text\_number=100):  **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 = np.vstack([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.append(topic\_cnt)  *# 初始化话题内文本数量* topic\_cnt\_dict[topic\_cnt] = 1  **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(**"最相似文本的话题编号"**, topic\_ser, **"相似度值"**, max\_value)  *# 添加文本向量* text\_vec = np.vstack([text\_vec, sen\_vec])  *# 分配话题编号(相似度值大于等于sim\_threshold，且话题内文本数量小于等于max\_text\_number）* **if** max\_value >= sim\_threshold **and** topic\_cnt\_dict[topic\_ser] <= max\_text\_number:  *# 将文本聚合到该最大相似度的话题中* topic\_serial.append(topic\_ser)  *# 话题内文本数量+1* topic\_cnt\_dict[topic\_ser] += 1  **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)  *# 初始化话题内文本数量* topic\_cnt\_dict[topic\_cnt] = 1   **def** main():  **global** preprocessed\_data  **global** topic\_serial   *# 输入文件名* file\_name = sys.argv[1]  *# 资源初始化* init()  *# 读文本并进行增量聚类* **with** codecs.open(file\_name, **'rb'**, **'utf-8'**) **as** infile:  **for** line **in** infile:  line = line.strip()  **if** line:  word\_li = preprocess(line, 1)  sen\_vec, has\_vec\_word\_li = compute\_sentence\_vector(word\_li)  **if** has\_vec\_word\_li:  preprocessed\_data.append(**u' '**.join(word\_li))  single\_pass(sen\_vec)  *# 输出聚类结果* cluster\_text\_li = []  outfile\_name = file\_name.split(**u'/'**)[-1]  outfile\_name = **u'Cluster\_%s'** % outfile\_name  **with** open(outfile\_name, **'wb'**) **as** outfile:  sorted\_topic\_cnt\_li = sorted(topic\_cnt\_dict.items(), key=**lambda** x:x[1], reverse=**True**)  **for** out\_topic\_ser, text\_cnt **in** sorted\_topic\_cnt\_li:  cluster\_text = **u''  if** text\_cnt >= 5:  **for** topic\_ser, text **in** zip(topic\_serial, preprocessed\_data):  **if** topic\_ser == out\_topic\_ser:  out\_str = **u'%d\t%s\n'** % (topic\_ser, text)  outfile.write(out\_str.encode(**'utf-8'**, **'ignore'**))  cluster\_text += **u'%s '** % text  **if** cluster\_text:  cluster\_text\_li.append(cluster\_text)  *# 对每个簇抽取关键词* category\_keywords\_li = extract\_keyword(cluster\_text\_li)  **for** key\_word\_li **in** category\_keywords\_li:  print(**u','**.join(key\_word\_li))   **if** \_\_name\_\_ == **"\_\_main\_\_"**:  main() |

### 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] |

# 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

为了帮助避免对度量单位选择的依赖性，数据应该规范化或标准化，使其各特征值域落入较小的共同区间，如[-1,1]或[0.0,1.0]。规范化会赋予所有属性相等的权重。对于涉及神经网络的分类算法或基于距离度量的分类（如最近邻分类）和聚类，规范化特别有用。比如数据规范化可以加快神经网络后向传播算法的收敛速度，可以避免在基于距离度量的分类聚类算法中具有较大值域属性的权重大于具有较小值域属性的权重。

### 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

极差正规化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值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，，。

当特征j的实际最小值和最大值未知，或离群点左右了最小-最大规范化时，该方法是有用的。标准化变换可能将原来的数据改变很多同时还必须保留规范化参数以便将来的数据可以用一致的方式规范化。

### 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 |

## 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。

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。

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

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。

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() |

# 神经网络

## CNN

### 基本概念

卷积神经网络并不是一个新的概念，甚至在20世纪90年代就已经被广泛使用，但深度学习卷土重来的第一功臣非卷积神经网络莫属，原因之一就是卷积神经网络非常适合计算机视觉应用的模型。

卷积在工程和数学上都有很多应用——在统计学中，加权的滑动平均是一种卷积；在概率论中，两个统计独立的变量x和y求和的概率密度函数是x和y的概率密度函数的卷积；在信号系统中，任意一个线性系统的输出都可以通过将输入信号与系统函数（系统的冲击响应）做卷积获得。

在卷积网络的术语中，卷积的第一个参数通常表示输入（Input），第二个参数表示核（Kernel），输出被称作特征映射（Feature Map）。离散时间系统的响应y(n)可以由激励x(n)和单位样值响应h(n)的卷积得到，公式为：

许多机器学习库将互相关的实现称为卷积，公式为：

### 文本分类示例

#### 数据准备

使用Stanford的Glove进行文本向量化表示，词向量维度为100维，单词数量为40万。数据集使用news20groups，该数据共有20个类别，数据信息如下：

训练集：11314 篇文本。

|  |  |  |
| --- | --- | --- |
| 序号 | 类别 | 数量 |
| 1 | alt.atheism | 480 |
| 2 | comp.graphics | 584 |
| 3 | comp.os.ms-windows.misc | 591 |
| 4 | comp.sys.ibm.pc.hardware | 590 |
| 5 | comp.sys.mac.hardware | 578 |
| 6 | comp.windows.x | 593 |
| 7 | misc.forsale | 585 |
| 8 | rec.autos | 594 |
| 9 | rec.motorcycles | 598 |
| 10 | rec.sport.baseball | 597 |
| 11 | rec.sport.hockey | 600 |
| 12 | sci.crypt | 595 |
| 13 | sci.electronics | 591 |
| 14 | sci.med | 594 |
| 15 | sci.space | 593 |
| 16 | soc.religion.christian | 599 |
| 17 | talk.politics.guns | 546 |
| 18 | talk.politics.mideast | 564 |
| 19 | talk.politics.misc | 465 |
| 20 | talk.religion.misc | 377 |

测试集：7532篇文本。

|  |  |  |
| --- | --- | --- |
| 序号 | 类别 | 数量 |
| 1 | alt.atheism | 319 |
| 2 | comp.graphics | 389 |
| 3 | comp.os.ms-windows.misc | 394 |
| 4 | comp.sys.ibm.pc.hardware | 392 |
| 5 | comp.sys.mac.hardware | 385 |
| 6 | comp.windows.x | 395 |
| 7 | misc.forsale | 390 |
| 8 | rec.autos | 396 |
| 9 | rec.motorcycles | 398 |
| 10 | rec.sport.baseball | 397 |
| 11 | rec.sport.hockey | 399 |
| 12 | sci.crypt | 396 |
| 13 | sci.electronics | 393 |
| 14 | sci.med | 396 |
| 15 | sci.space | 394 |
| 16 | soc.religion.christian | 398 |
| 17 | talk.politics.guns | 364 |
| 18 | talk.politics.mideast | 376 |
| 19 | talk.politics.misc | 310 |
| 20 | talk.religion.misc | 251 |

统计代码的github地址为

#### 实验步骤

1. 将所有的新闻样本转化为词索引序列，即为每一个词依次分配一个整数ID。遍历所有的新闻文本，我们只保留最常见的20000个词，而且，每个新闻文本最多只保留1000个单词。

2. 生成一个词向量矩阵。第i列表示词索引为i的单词的词向量。

3. 将词向量矩阵载入Keras **Embedding层**，设置该层的权重不可再训练（即在之后的网络训练过程中，使用GloVe预训练词向量，词向量不再改变）。

4. Keras Embedding层之后使用**1D的卷积层**进行预测模型的构建，并用一个**softmax全连接层**输出新闻类别判断。

#### 实验代码

##### 数据准备

###### （1）导入GloVe词向量词典

我们使用了GloVe向量作为词嵌入层的初始集合。因此，这里导入GloVe词向量词典。

|  |
| --- |
| import numpy as np  # GloVe向量词典  embeddings\_index = dict()  # 读入向量  with open('../data/glove.6B/glove.6B.100d.txt', encoding='utf-8', errors='ignore') as infile:  for line in infile:  line = line.strip()  if line:  items = line.split()  word = items[0]  coefs = np.array(items[1:], dtype='float32')  embeddings\_index[word] = coefs  print('readed %d word vectors' % len(embeddings\_index)) |

###### （2）导入训练和测试语料

这里我们使用sklearn的load\_files方法来加载分类数据。

|  |
| --- |
| from sklearn.datasets import load\_files  train\_data = load\_files(container\_path='../data/20news-bydate/20news-bydate-train', # 分类语料根目录  categories=None, # 需要加载的类目名称列表  encoding="gbk", decode\_error="ignore") # 文件编码  print('readed %d train texts' % len(train\_data.data))  test\_data = load\_files(container\_path='../data/20news-bydate/20news-bydate-test', # 分类语料根目录  categories=None, # 需要加载的类目名称列表  encoding="gbk", decode\_error="ignore") # 文件编码  print('readed %d test texts' % len(test\_data.data)) |

###### （3）将文本数据转换称神经网络所需的2维张量

由于是多分类问题，这里我们对类别做了one-hot编码。

|  |
| --- |
| from keras.preprocessing.text import Tokenizer  from keras.preprocessing.sequence import pad\_sequences  from keras.utils import to\_categorical  # 构建“分词器”,设置在从语料获得词典时收录的最大词语数量  MAX\_NB\_WORDS = 20000  tokenizer = Tokenizer(num\_words=MAX\_NB\_WORDS)  # 构建词典  tokenizer.fit\_on\_texts(train\_data.data)  # 获得词典中单词索引  word\_index = tokenizer.word\_index  # 打印词典收录词语数量  print('dictionary len = %d' % len(word\_index))  # 将训练文本集中的每篇文本转换为词索引序列  train\_sequences = tokenizer.texts\_to\_sequences(train\_data.data)  # 将测试文本集中的每篇文本转换为词索引序列  test\_sequences = tokenizer.texts\_to\_sequences(test\_data.data)  # 使每篇文本的词索引序列长度一致  MAX\_SEQUENCE\_LENGTH=1000  x\_train = pad\_sequences(train\_sequences, maxlen=MAX\_SEQUENCE\_LENGTH)  x\_test = pad\_sequences(test\_sequences, maxlen=MAX\_SEQUENCE\_LENGTH)  # 获得每篇文本对应的类别  y\_train = to\_categorical(train\_data.target)  y\_test = to\_categorical(test\_data.target)  # 求类别索引对应类别名称  labels\_index = dict(enumerate(train\_data.target\_names))  # 输出数据形状  print("x\_train shape = ", x\_train.shape)  print("y\_train shape = ", y\_train.shape)  print("x\_val shape=", x\_test.shape)  print("y\_val shape=", y\_test.shape) |

###### （4）用GloVe词向量词典将构建的词典转换称2维张量矩阵

|  |
| --- |
| # 当前词典中的词语数量,之所以这样恐怕是因为Tokenizer类旧版中参数num\_words不起作用  num\_words = min(MAX\_NB\_WORDS, len(word\_index))  # 初始化嵌入层矩阵  EMBEDDING\_DIM = 100 # 嵌入层输出尺寸  embedding\_matrix = np.zeros((num\_words, EMBEDDING\_DIM))  # 遍历由训练集构建的词典中的每个词语  for word, i in word\_index.items():  # 如果词语的索引也就是位置超过了设置的词典收录词语数，则丢弃  if i >= num\_words:  continue  # 读取词语向量,如果词语未被GloVe收录，则其向量为0向量  if word in embeddings\_index:  embedding\_matrix[i] = embeddings\_index[word] |

##### 设计网络结构

###### （1）用词典向量矩阵初始化Embedding层

|  |
| --- |
| from keras.models import Sequential  from keras.layers.embeddings import Embedding  model = Sequential()  model.add(Embedding(input\_dim=num\_words, # 输入词向量矩阵的最大词语数量，也就是词典最大索引数+1，词典索引从0开始  output\_dim=EMBEDDING\_DIM, # 将输入向量矩阵转换为3维矩阵后，第3维的元素个数  weights=[embedding\_matrix], # 用词典向量矩阵初始化嵌入层参数  input\_length=MAX\_SEQUENCE\_LENGTH, # 嵌入层接受的输入文本集的每一行文本的单词序列的长度  trainable=False)) # 嵌入层不再可训练 |

###### （2）卷积抽取特征

该段结构可以重复多次。

|  |
| --- |
| # 卷积层  from keras.layers import Conv1D  model.add(Conv1D(filters=256, # 卷积滤子输出的维度，要求整数  kernel\_size=5, # 卷积核的空域或时域窗长度。要求是整数或整数的列表，或者是元组。如果是单一整数，则应用于所有适用的维度  activation='relu',# 激活函数  ))  # 池化层  from keras.layers import MaxPooling1D  model.add(MaxPooling1D(pool\_size=5))  # 丢弃层  from keras.layers import Dropout  model.add(Dropout(0.4)) |

###### （3）平坦层

|  |
| --- |
| # 平坦层  from keras.layers import Flatten  model.add(Flatten()) |

###### （4）全连接层

预测输出

|  |
| --- |
| from keras.layers import Dense  model.add(Dense(128, activation='relu'))  model.add(Dense(len(labels\_index), activation='softmax')) |

##### 设置模型

|  |
| --- |
| model.compile(loss='binary\_crossentropy',  optimizer='Adagrad',  metrics=['accuracy'])  model.summary() |

##### 训练

|  |
| --- |
| train\_history = model.fit(x\_train, y\_train,  batch\_size=25, # 1个训练周期中，mini-batch gradient descent中每一批次100项数据  epochs=10, # 整个数据集将被周期地训练10次，取最优的一次  verbose=2, # 显示训练过程  validation\_split=0.2) # 20%验证数据 |

##### 评测

|  |
| --- |
| model.evaluate(x\_test, y\_test, verbose=1) |

##### 保存模型

|  |
| --- |
| model.save\_weights('./CNN20\_model') |

## RNN

## LSTM

## Bi-LSTM

## 激活函数

## 初始化方法

### Uniform

## 损失函数

### 交叉熵

### KL散度

### 对称KL散度

## 正则化

### L1正则化

### L2正则化

### L2/L1正则原理

### Dropout

## 优化算法

### SGD

### Momentum

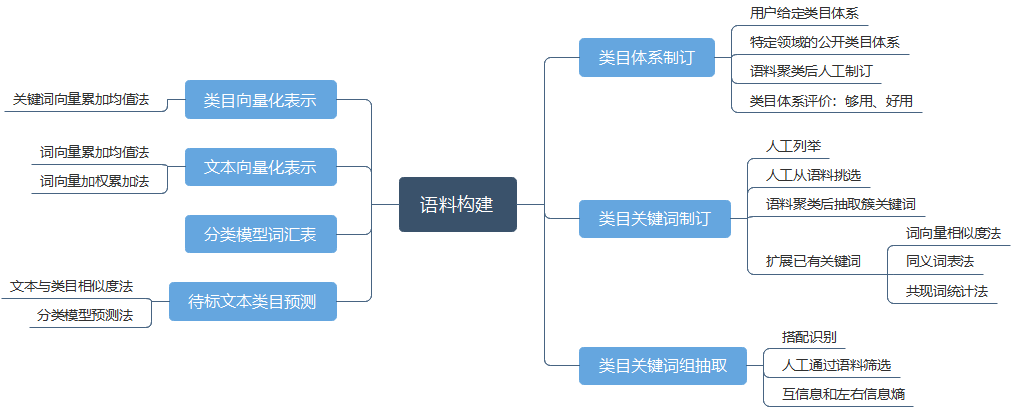
### Adagrad

基础实践篇

# 分类训练语料构建

## 语料构建导图和流程图

### 语料构建导图



### 语料构建机器端流程图



### 语料构建人工端流程图

### 机器端每日例行任务

## 类目体系构建

### 用户给定类目体系

### 特定领域的公开类目体系

《GBT20093-2006中文新闻信息分类与代码》是新闻文本的类目体系标准。该标准规定了中文新闻信息分类的原则、方法、体系和类目代码。**标准适用于通讯社、报社、广播电台、电视台、杂志社、网络媒体，以及各种资讯机构对中文新闻信息进行分类、检索、标识等方面的处理与交换业务。**标准制定了两种类型的表，分别为**主类表**和**复分表**，主类表是新闻信息分类的主要依据，复分表是根据新闻信息内容中出现的共性内容单独编列构成的表，又称**附表或辅助表**。主类表由三个表组成，分别为**一级类表**、**简表**和**详表**。其中一级类表是一级类目组成的一览表，又称基本大类表。简表是一级、二级类目组成的一览表，又称基本类目表。详表是全部类目组成的一览表，又称主表。标准由**三大部类**构成：政治部类，代码01~05；经济部类，代码11~22，文化部类，代码为31~39，详情可以看标准的表1“中文新闻信息分类一级类表”。标准的详表中最多分为**五个层级**类目，其中**一级类目23个**，**二级类目293个**。标准的一级类表如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 部类 | 代码 | 一级类目名称 | 说明 |
| 政治 | 01 | 政治 |  |
| 02 | 法律、司法 |  |
| 03 | 外交、国际关系 |  |
| 04 | 军事 |  |
| 05 | 社会、劳动、灾害事故 |  |
| 经济 | 11 | 经济 |  |
| 12 | 财政、金融 |  |
| 13 | 基本建设、建筑业、房地产 |  |
| 14 | 农业、农村 |  |
| 15 | 矿业、工业 |  |
| 16 | 能源、水务、水利 |  |
| 17 | 信息产业 |  |
| 18 | 交通运输、邮政、物流 |  |
| 19 | 商业、外贸、海关 |  |
| 21 | 服务业、旅游业 |  |
| 22 | 环境、气象 |  |
| 文化 | 31 | 教育 |  |
| 33 | 科学技术 |  |
| 35 | 文化、休闲娱乐 |  |
| 36 | 文学、艺术 |  |
| 37 | 传媒业 |  |
| 38 | 医药、卫生 |  |
| 39 | 体育 |  |

### 语料聚类后人工制订

一般情况下舆情文本的类目体系可以根据自己的训练数据制定。类目体系一般为二级结构，一级类目通常包括经济、政治、文教、民生、信息、工业、能源、农业等，二级类目可以根据自己的训练数据再详细划分。

## 类目关键词（组）和关联词（组）抽取

### 人工列举

### 人工从语料挑选

### 语料聚类后抽取簇关键词

### 扩展已有关键词

#### 词向量相似度法

利用fastText词嵌入库模型，对类目关键词进行扩展。扩展代码如下：

|  |
| --- |
| *#coding:utf-8 """ 类目关键词扩展（该文件也可返回任意词语的相关词语列表） 用法：python 类目关键词扩展.py 输入文件名 要求：python3，NLTK，PyHanLP，fastText 输入文件格式：每行一个类目关键词 输出文件名：扩展词\_输入文件名 输出文件格式 输入词语\t扩展词1 扩展词2 ... """* **import** codecs **import** numpy **as** np **from** fastText **import** load\_model **from** time **import** time **import** sys   **def** read\_fasttext\_data(file\_path):  *"""  读取fastText词语向量模型* **:param** *file\_path: fastText词语向量模型文件名* **:return***: fastText模型，单位化后的fastText矩阵  """* t0 = time()  fasttext\_model = load\_model(file\_path)  t1 = time()  print(**"加载fastText向量库时间%.2fs"** % (t1-t0))  *# 对fasttext向量矩阵的每个词向量（行向量）单位化* t0 = time()  fasttext\_vec = fasttext\_model.get\_input\_matrix()[:len(fasttext\_model.get\_words())]  fasttext\_vec = (fasttext\_vec.T/np.linalg.norm(fasttext\_vec, axis=1)).T  t1 = time()  print(**"fastText向量单位化时间%.2fs"** % (t1 -t0))  print(**"fasttext词汇表长度="**, len(fasttext\_model.get\_words()))  print(**"fasttext词语向量矩阵形状"**, fasttext\_vec.shape)  **return** fasttext\_model, fasttext\_vec   **def** get\_related\_words(fasttext\_model, fasttext\_vec, word, num=10):  *"""  通过fastText词语向量库返回近义词（需要注意这里排序函数没有进行优化，  因此会对fastText词汇表中的所有词语按相似度大小进行一次排序）* **:param** *fasttext\_model: 加载到内存中的fastText词向向量模型* **:param** *fasttext\_vec: 单位化后的fastText词向向量矩阵* **:param** *word: 被检索的词语* **:param** *num: 返回的最多的相关词数量* **:return***:相关词及相似度值列表  """* **if** fasttext\_model.get\_word\_id(word) != -1:  word\_vec = fasttext\_vec[fasttext\_model.get\_word\_id(word)]  **else**:  print(**"warning: fastText中没有收录词语：%s"** % word )  **return None** sim\_vec = np.dot(word\_vec, fasttext\_vec.T)  sorted\_sim\_vec = sorted(zip(np.arange(len(sim\_vec)), sim\_vec), key=**lambda** x: x[1], reverse=**True**)  print(**"获取词语'%s'的相关词"** % word)  related\_word\_li = []  fasttext\_words = fasttext\_model.get\_words()  **for** i **in** np.arange(1, min(num, len(sorted\_sim\_vec))+1): *# 注意有越界风险* related\_word\_li.append((fasttext\_words[sorted\_sim\_vec[i][0]], sorted\_sim\_vec[i][1]))  **return** related\_word\_li   **def** main():  file\_name = sys.argv[1]  outfile\_name = **u'RelatedWords\_%s'** % file\_name  outfile = open(outfile\_name, **'wb'**)  *# 读取fastText词语向量模型，注意模型文件路径及名称需要根据实际修改* fasttext\_model, fasttext\_vec = read\_fasttext\_data(**"../dictionary/cc.zh.300.bin"**)  **with** codecs.open(file\_name, **'rb'**, **'utf-8'**, **'ignore'**) **as** infile:  **for** line **in** infile:  line = line.strip()  **if** line:  word = line  related\_words\_li = get\_related\_words(fasttext\_model, fasttext\_vec, word, 10)  out\_str = **u'cluster\_related\_words\t%s %s\n'** % (word, **' '**.join([**u'%s'** % (w[0]) **for** w **in** related\_words\_li]))  outfile.write(out\_str.encode(**'utf-8'**, **'ignore'**))  outfile.close()  print(**"program finished"**)  **if** \_\_name\_\_ == **"\_\_main\_\_"**:  main() |

该段代码可以用于任何词语的同义词（近义词）扩展。测试输出结果展示如下表：

|  |
| --- |
| cluster\_related\_words 经济 社会 經濟 发展 宏观 产业 财政 经济体 农业 县域 贸易 cluster\_related\_words 政治 政冶 人物 思想 外交 宗教 政治家 非政治 政客 政治学 军事 cluster\_related\_words 文教 師荒 溝通漢華 頓華僑 崇德光慧 尊懷 中學語 橡塑料 會執行長 科教 功文 cluster\_related\_words 民生 民计 600016 汐止線 之多艰 加银家盈 利溥 站源 国计 葛旋 消費新聞 cluster\_related\_words 工业 化工业 制造业 轻工业 农业 重工业 产业 园区 工業 工业化 纺织 cluster\_related\_words 信息 旌展 上讯 数据 信息一 資訊 此信息 信息源 信息三 轩马 讯息 cluster\_related\_words 农业 林业 畜牧业 农牧业 种植业 农机 农林 农村 农产品 渔业 农民 cluster\_related\_words 有害 危害 有毒 有害于 有益 有害物 对人体 不利 无害 有益的 检疫性 cluster\_related\_words 股票 股市 炒股 A股 垃圾股 股民 新股 市值 个股 股价 炒股票 cluster\_related\_words 直升机 直升飞机 飞机 直升機 米-171 米-8 停机坪 救援机 NH-90 Z-9 旋翼机 cluster\_related\_words 感冒 液竄 腸胃型 重感冒 咳嗽 流感 流鼻涕 状像 生病 流鼻水 咽喉炎 cluster\_related\_words 自行车 山地车 摩托车 单车 学骑 电动车 有桩 脚踏车 滑板车 车篮 骑车 cluster\_related\_words 电话 打电话 打来 发信息 听电话 坐机 打给 拨通 打接 挂电话 拨错 |

从结果可以看到，训练语料对于词嵌入词语模型库的影响，即有的时候还是需要根据自己的语料训练自己的词语向量，同时注意到越是类别名称词由于其本身的抽象概括特性（也就是在多种上下文中出现）使其扩展出的近义词之间不一定具有明显的相关性。

#### 同义词表法

#### 共现词统计法

### 词语频率法

|  |
| --- |
| *#coding:utf-8 """ 关键词抽取tf法 用法：python 关键词抽取tf.py 文件名 要求：python3，NLTK，PyHanLP 说明：文件中存储要抽取关键词的所有文本，程序会统计出每个词语的频率， 并按照频率降序输出词语及其对应频率值 """* **import** codecs **from** pyhanlp **import** \* **import** re **from** nltk.probability **import** FreqDist 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   **if** \_\_name\_\_ == **"\_\_main\_\_"**:  file\_name = sys.argv[1]  file\_word\_li = []  **with** codecs.open(file\_name, **'rb'**, **'utf-8'**, **'ignore'**) **as** infile:  **for** line **in** infile:  line = line.strip()  **if** line:  word\_li = preprocess(line)  file\_word\_li.extend(word\_li)  fdist = FreqDist(file\_word\_li)  **for** word, freq **in** sorted(fdist.items(), key=**lambda** x: x[1], reverse=**True**):  print(word, freq) |

## 类目和文本向量化表示

### 类目向量化表示

类目向量通过类目关键词向量加权平均求得。公式如下：

这里给出一个类目向量表示及可视化的代码

|  |
| --- |
| *#coding:utf-8 """ 类目向量表示及可视化 """* **import** codecs **from** time **import** time **from** fastText **import** load\_model **import** numpy **as** np **from** sklearn.decomposition **import** TruncatedSVD **from** sklearn.preprocessing **import** Normalizer **from** sklearn.pipeline **import** make\_pipeline **import** matplotlib.pyplot **as** plt **from** matplotlib.font\_manager **import** FontProperties font = FontProperties(fname=**'C:\Windows\Fonts\simkai.ttf'**)  cnames\_li = [**'b'**, **'g'**, **'r'**, **'c'**, **'m'**, **'y'**, **'k'**]\*7   *# 读取fastText词语向量矩阵* **def** read\_fasttext\_data(file\_path):  t0 = time()  fasttext\_model = load\_model(file\_path)  t1 = time()  print(**"加载fastText向量库时间%.2fs"** % (t1-t0))  **return** fasttext\_model   *# 构建类目向量并降维可视化* **def** build\_class\_vector(fasttext\_model):  *# 构建类目向量矩阵* class\_vec\_array = np.array([])  class\_vec\_array.resize((0, fasttext\_model.get\_dimension()))  label\_li = []  **with** codecs.open(**'../data/class\_keywords\_level1.txt'**, **'rb'**, **'utf-8'**, **'ignore'**) **as** infile:  **for** line **in** infile:  line = line.strip()  **if** line:  class\_label, key\_words\_str = line.split(**u'\t'**)  label\_li.append(class\_label)  key\_words\_li = key\_words\_str.split()  class\_vec = np.array([])  class\_vec.resize((1, fasttext\_model.get\_dimension()))  word\_cnt = 0  **for** word **in** key\_words\_li:  *# 词语有向量值* **if** fasttext\_model.get\_word\_id(word) != -1:  word\_vec = fasttext\_model.get\_word\_vector(word)  class\_vec += word\_vec  word\_cnt += 1  class\_vec /= word\_cnt  class\_vec /= np.linalg.norm(class\_vec)  class\_vec\_array = np.row\_stack((class\_vec\_array, class\_vec))  print(**"label\_li len = "**, len(label\_li))  print(**"class\_vec\_array shape = "**, class\_vec\_array.shape)   *# 类目向量矩阵降维* print(**"Performing dimensionality reduction using LSA"**)  t0 = time()  *# Vectorizer results are normalized, which makes KMeans behave as  # spherical k-means for better results. Since LSA/SVD results are  # not normalized, we have to redo the normalization.* svd = TruncatedSVD(2)  normalizer = Normalizer(copy=**False**)  lsa = make\_pipeline(svd, normalizer)  class\_vec\_array = lsa.fit\_transform(class\_vec\_array)  print(**"done in %fs"** % (time() - t0))  explained\_variance = svd.explained\_variance\_ratio\_.sum()  print(**"Explained variance of the SVD step: {}%"**.format(  int(explained\_variance \* 100)))  print(**"class\_vec\_array shape = "**, class\_vec\_array.shape)   *# 类目向量可视化显示* fig, ax = plt.subplots()  **for** i **in** np.arange(class\_vec\_array.shape[0]):  ax.scatter(class\_vec\_array[i, 0], class\_vec\_array[i, 1], c=cnames\_li[i], marker=**"\*"**)  ax.legend(labels=label\_li, prop=font)  plt.show()   **if** \_\_name\_\_ == **"\_\_main\_\_"**:  fasttext\_model = read\_fasttext\_data(**'../dictionary/cc.zh.300.bin'**)  build\_class\_vector(fasttext\_model) |

### 文本向量化表示

文本向量通过文本词语向量加权平均求得。公式如下：

## 分类模型词汇表（特征词表）

## 待标文本类目预测

### 文本与类目相似度方法

计算每个类目的向量

计算待标注文本向量

计算文本向量与各类目向量的相似度（余弦相似度）

对类目名称按相似度值降序排列，取top1作为自动标注结果

### 分类模型预测法

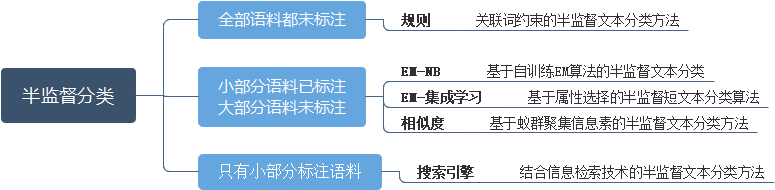
## 半监督文本分类调研

### 什么是半监督文本分类？

给定一些标注类别的熟语料和很多未标注类别的生语料，利用常见的机器学习方法（朴素贝叶斯、线性支持向量机、集成学习等）将生语料自动标注为熟语料进而训练出更具泛化能力的文本分类模型的过程。当前常见的半监督学习算法有：半监督EM算法、协同训练（co-training）算法、直推式支持向量机。

### 可参考的文献

#### 文献导图



#### 第1篇文献 张博峰-2007年-基于自训练EM算法的半监督文本分类-国防科技大学学报

这篇文章详细说明了EM-NB这种半监督文本分类方法，同时做了一些改进。这里首先说明EM-NB的基本原理。EM就是期望最大化算法。操作步骤就是两步，第一步是E步骤，第二步是M步骤。

**E步骤：**利用现有的中间分类器和下边的式子估计未标注样本集中的每个样本属于每个类别的概率。

稍稍解释一下上边的式子，表示文本在当前的中间分类器下属于类别的概率，显然分子是朴素贝叶斯公式的分子，即先验概率乘以似然概率，只不过这里是在中间分类器的条件下的概率（如果实在觉着别扭可以捂住那个）。显然表示在当前的中间分类器下类别出现的概率，表示文本中词项总数，表示文本中第k个词项，表示在类别和中间分类器的条件下词项出现的概率。表示类别总数，显然分母是为了使计算出的概率值在0-1之间所做的归一化操作。表示未标注文本集中文本总数，连乘可以通过取对数变为连加。

**M步骤：**将E 步骤估计出类别的未标注样本加入到训练集中（**此时训练集由已标注文本集和有了属于各类别概率的未标注文本集组成**）利用下边的两个式子训练中间分类器。

稍微解释一下上边的两个式子，第1个式子：是对似然概率的估计，表示训练集中文本总数，表示词项在文本中出现的次数，表示文本属于类别的概率，表示词汇表（特征词表）的词项总数，显然分母是为了使概率在0-1之间所作的归一化操作。第2个式子：表示类别的先验概率的估计，表示训练集中文本总数，表示文本属于类别的概率，表示类别总数，显然分母依然是为了使概率值在0-1之间所做的归一化操作。

重复E步骤和M步骤直到M步骤估计得到的各分类器参数值不再变化。

需要说明的是，在E步骤和M步骤迭代开始前，先通过已标注数据计算出每个类别的和每个类别下词汇表中各词项出现的概率值。

以上是对EM-NB的算法的说明，下边介绍这篇文献所作的改进。在每次EM迭代过程中，将分类器最肯定的未标注样本移入已标注样本集，也就是说在以后的M步骤时，未标注文本集中所含文本的数量会减少，加快收敛速度。

**EM-NB的四个假设**：

1. 混合器模型
2. 混合器分量与类目一一映射
3. 文档中的词独立于上下文生成
4. 文档长度对所有类均匀分布

**朴素贝叶斯模型**被称为**生成模型**的原因：

从分类的反向角度理解，一篇文档是由各个类目词语（成分）的类目出现概率作为权重线性生成的。

#### 第2篇文献 贾志洋-2012年-结合信息检索技术的半监督文本分类方法-苏州大学学报（自然科学版）

这篇文章所描述的系统需要有一些已标注语料，之后系统利用搜索引擎扩展语料并完成标注，最后得到泛化能力较好的分类器。所涉及的内容有：类目关键词的构建、查询序列的构建、更新训练集并训练模型。

**类目关键词的构建**：对已标注的少量数据，用互信息法抽取类目类目关键词，这里加入了**关键词过滤规则**：

**规则1：关键词在所在类目数据中的频次应大于类目数据中所有词项的平均词频**

**规则2：关键词互信息值应大于0**

**查询序列的构建**：查询序列为由2个类目关键词组合构成的元组序列和由3个类目关键词组合构成的元组序列。**查询元组与任意类别的相关度**由下式给出

为词项在类别c数据中的频次，是词项与类别c的互信息。

假设针对某个类别需要下载N个网页，该类别有M个查询，则针对该类目的每个查询所需下载网页数量按下式给出

小解：上边的式子说明，针对某个查询元组下载网页的数量由该查询元组与类别的相关度决定，相关度越高下载网页数量越多。

**更新训练集并训练模型**：

1. 用现有标注文本训练初始分类器。
2. 预测下载的网页集合E的类别标记。
3. 针对每个类目，保留满足下边条件的网页文本：预测的标记与该网页文本对应的查询元组所对应的类目一致。在保留的文本中随机选择m个构成文本集。
4. 将所有类目的新样本集加入到训练集中
5. 迭代（1）-（4）步骤次，为设置的阈值。

|  |
| --- |
| 心得：文章的亮点是查询串的构建以及查询串与类目的相关度度量，利用了搜索引擎检索和返回结果的相关性扩充了语料。可能的问题是搜索引擎返回的网页内容通常是长文本，如果是短文本分类的话，这样的语料是否合适有待考虑。 |

#### 第3篇文献 蔡月红-2010-基于属性选择的半监督短文本分类算法-计算机应用

这篇文章介绍了基于ReliefF评估和独立性度量的特征词选择，分类模型采用**集成学习方法**。采用属性选择和集成学习的用途为（1）克服朴素贝叶斯条件独立性假设（2）克服EM算法对于分类器参数初始化值敏感。下边介绍**算法**：

输入：已标注语料集L，未标注语料集U，个体子模型个数N

输出：分类结果

初始化：

for i=1:N

1. 运用bootstrap取样在L上产生训练子集
2. 利用特征选择从训练子集中选择最优特征子集，从而得到最优训练子集
3. 在最优训练子集上训练个体贝叶斯子模型

end

循环：

E-step：使用当前分类器组，以多数投票策略分类未标注语料集U并计算未标注样本集中各样本最大后验概率

M-step：将估计出类别的样本加入到训练集中，更新分类器组中各分类器参数

重复E步骤和M步骤，直到收敛或参数变化小于某一给定的阈值为止。

#### 第4篇文献 韩红旗-2010-关联词约束的半监督文本分类方法-计算机工程与应用

这篇文章有下边三个要点：（1）从完全未标注文档集出发（2）利用类关联词对对文档进行划分形成初始分类器（3）利用类关联词确定的分类约束条件，采用EM方法迭代地学习一个分类器，最后给文档标以合适类别。

**关联词的获取**：文章中并没有提出从没有类别标记语料中抽取类目关键词的方法。

**利用关联词对文档集划分**：

（1）包含一个类关联词

（2）包含多个类关联词

（3）没有类关联词

**分类约束条件**：

（1）包含一个类关联词，则句子以概率1属于类别cj

（2）包含多个类关联词，则句子以等概率属于多个类别c1,c2,…

（3）不含类关联词，则句子以等概率属于所有类别

有了分类约束，在标注时按照分类约束规则规定给出候选类目，并用分类器给出概率值。

可以直接以下边的矩阵形式存储每个句子属于各类别的概率

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 类别1 | 类别2 | … | 类别|C| |
| 文本1 |  |  |  |  |
| 文本2 |  |  |  |  |
| … |  |  |  |  |
| 文本|D| |  |  |  |  |

|  |
| --- |
| 心得：可以通过分类约束条件，提高含有两个关键词句子的标注准确率。但是，这样的关键词抽取是个难点，必要时可能还是需要人工标注。 |

#### 第5篇文献 杜芳华-2014-基于蚁群聚集信息素的半监督文本分类算法-北京工业大学学报

这篇文章不是用的分类模型对未标注样本做分类，而是利用了相似度的方法来对文本做标注，实际上它很有增量聚类的味道。为了叙述方便直接从计算式入手进行总结。系统首先需要一小部分标注语料。针对输入的一篇文本有两个量需要计算，下边详述

**候选种群判定：**这里的候选种群在文本分类中指针对一篇文本可能被标注的类别。按下式判定

满足 的种群可以作为候选

其中，，K为类别总数。

小解：表示文本j与类别k的相似性，分母取的是文本与所有类别的相似度最大值，显然这个判定规则指的是只有相似度与最大相似度比值大于一定值的类别才可以作为候选类别。极端情况如果设置，那么就只有最相似的那个类别才可以作为候选类别。

需要说明的是显然这是一个递推式，t时刻的值与t-1时刻的值有关，初始时所有类别的

小解：表示类别总数，表示类别k中的一个文本，是类别名称集合，是需要标注类别的文本，表示待标注文本j与类别中的文本i的相似性。从这个式子可以看到，对于一篇文本与一个类别之间的相似性是用这篇文本与类别中的每篇文本的相似性来度量的。

这里的计算的是两篇文本的余弦相似度。

**是否需要对文本做标注**：

满足的文本可以对其进行标注

小解：表示的是按文本与类别相似度降序后前top-k个类别的数量（可以理解为相似度值相同的top-k个类别），需要标注文本的候选类别数量，K为类别总数，显然这里的意图是如果一篇文本的候选类别数量太多，则不对他进行标注。需要说明的是文本的最后的类别只有一个，如果候选类别数量多于一个则随机选择一个。

**算法流程：**

输入：待标注的文本

输出：扩展后的训练集

for {

if() then {

记录该类别为候选类别

}

if() {

按照记录的候选类别序列，把文本放到相应类目下

}

|  |
| --- |
| 心得：这篇文献的优势在于它计算文本与类目下所有文本的相似度而不是文本与类目向量的相似度。其次，它对被标注文本是否可以被准确标注做了数值上的衡量，也就是文献中提到的“是否需要对文本标注”。 |

### 可参考的代码

|  |  |  |
| --- | --- | --- |
| 算法名称 | 地址 | 编程语言 |
| tsvm | <https://github.com/CalculatedContent/tsvm> | Perl |
| tsvm | <https://github.com/esengie/TSVM-on-Python> | Python |
| em-nb | <https://github.com/pl366/Expectation-maximization-Naive-bayes> |  |
| em-nb | <https://github.com/sruddell09/expectationMaximization> |  |

## 标注工具功能及标注方法

### 标注工具调研

以下将调研到的标注工具总结为下表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 工具名称 | 官方地址 | 简介 | 优缺点 | 使用参考 |
| Prodigy | 官网地址：  <https://prodi.gy/>  体验地址：  <https://prodi.gy/demo?view_id=ner> | 1. 文本分类语料标注  2. 实体信息标注 | 优点：  1. 界面清晰  2. 标注方便  3. 标注功能齐全  缺点：  1. 付费 | 无 |
| YEDDA | 官网地址：  <https://github.com/jiesutd/YEDDA> | 1. 实体信息及关系标注 | 优点：  1. 界面比较清晰  2. 标注比较方便  3. 系统构建省时  缺点：  1. 只能单机本地运行  2. 开发语言为Python2  3. 没有分类标注功能 | <https://arxiv.org/pdf/1711.03759.pdf> |
| BRAT | 官网地址：  <http://brat.nlplab.org/index.html>  github地址：  <https://github.com/nlplab/brat> | 1. 实体及关系标注 | 优点：  1. 界面比较清晰  2. 标注比较方便  3. 服务端、客户端模式  缺点：  1. 软件本身存在bug  2. 系统构建耗时  3. 没有分类标注功能 | <https://blog.csdn.net/owengbs/article/details/49780225> |

由上边总结以下实践建议：

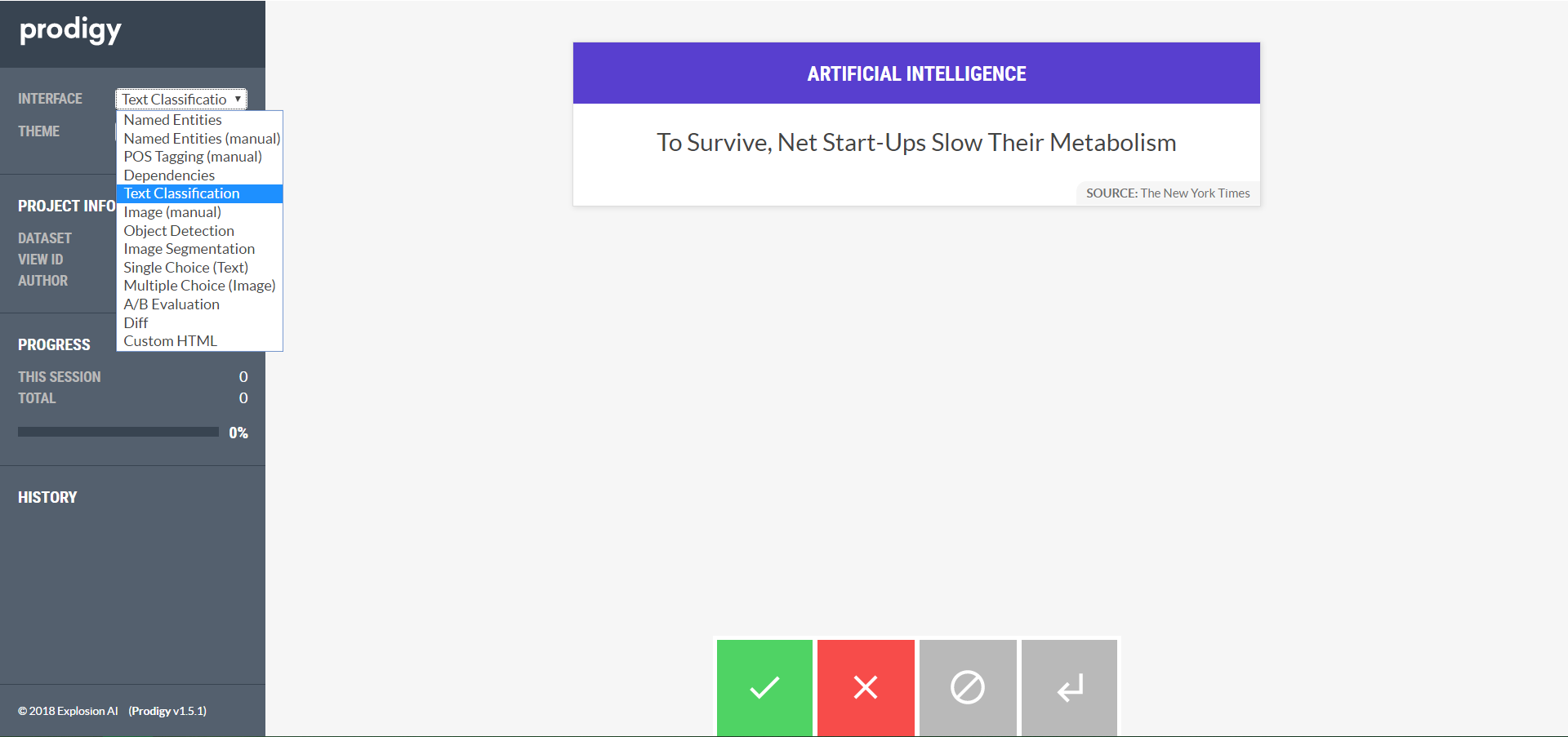
1. 如果标注语料可在标注人员计算机本地存储，则YEDDA是首选。
2. 如果标注语料不能在标注人员计算机本地存储，则BRAT是首选。
3. 对于标注界面设计和人机交互功能设计，可以参考Prodigy。

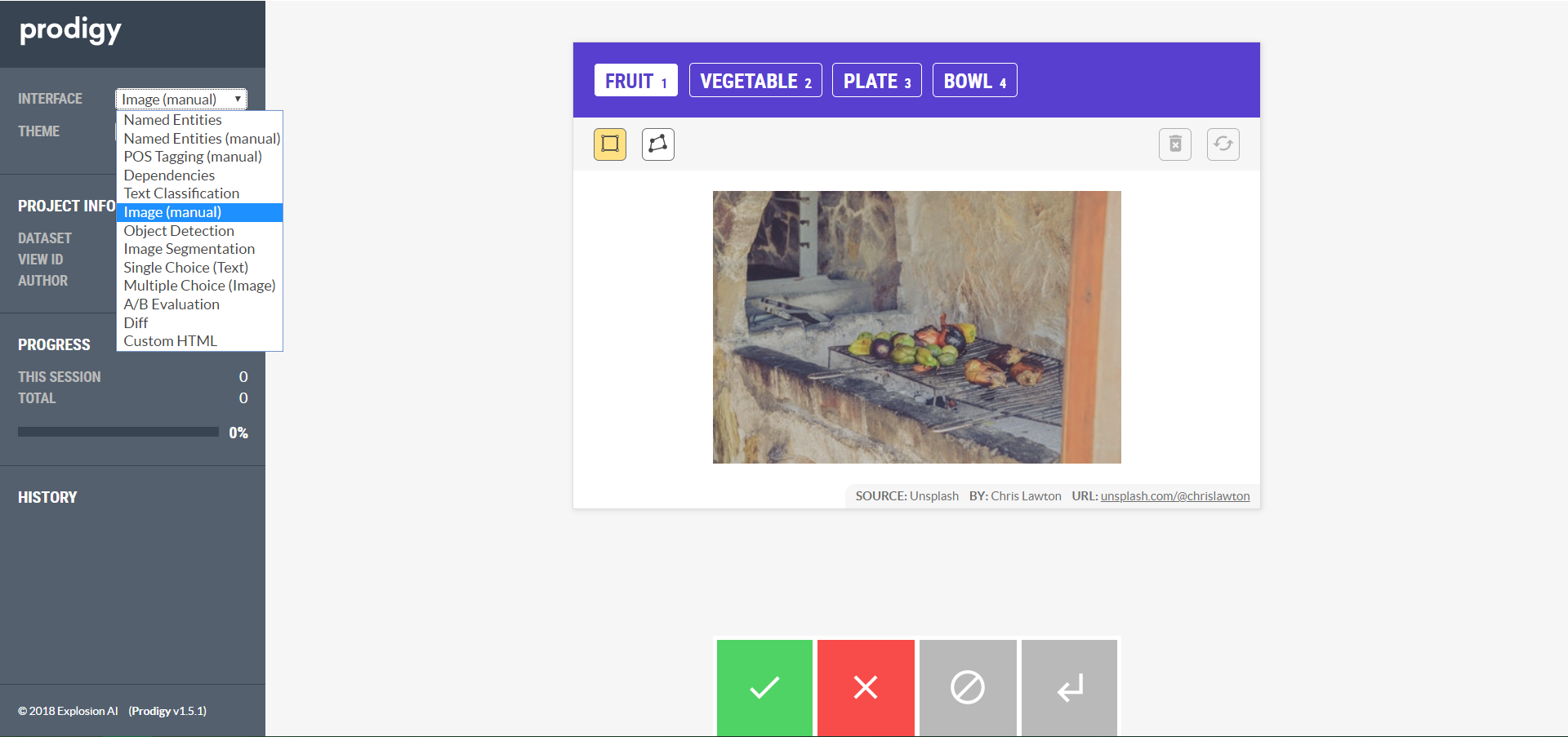
### 标注工具应具备的功能

#### 界面功能

##### 界面类型的选择

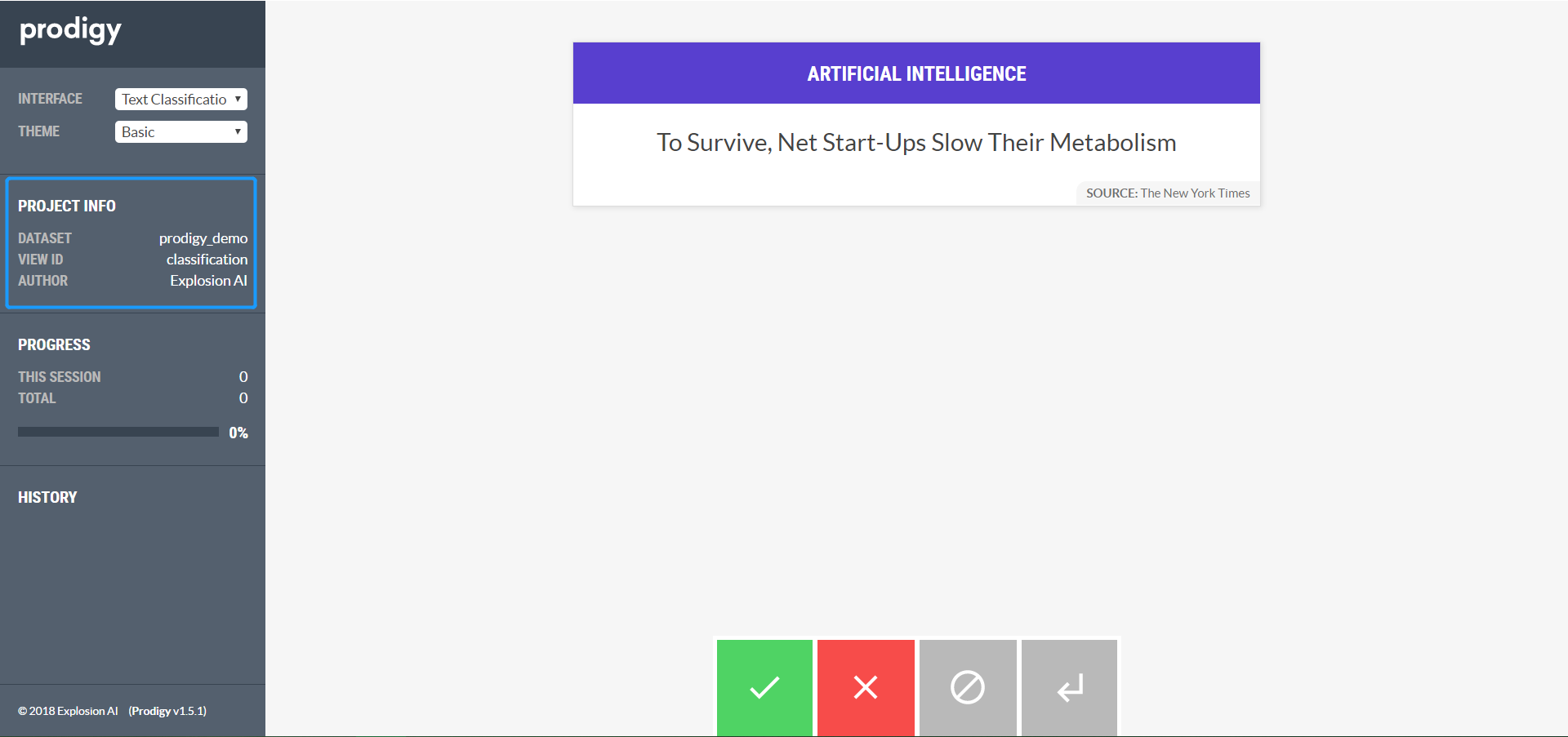
用户可以根据具体的标注任务选择标注界面。比如：对于文本分类语料的标注可以选择“文本分类”选项，对于图像识别训练语料可以选择“图像标注”选项，不同标注选项会产生不同的标注界面。可以参考下边两张图中“INTERFACE”栏和相应标注界面。





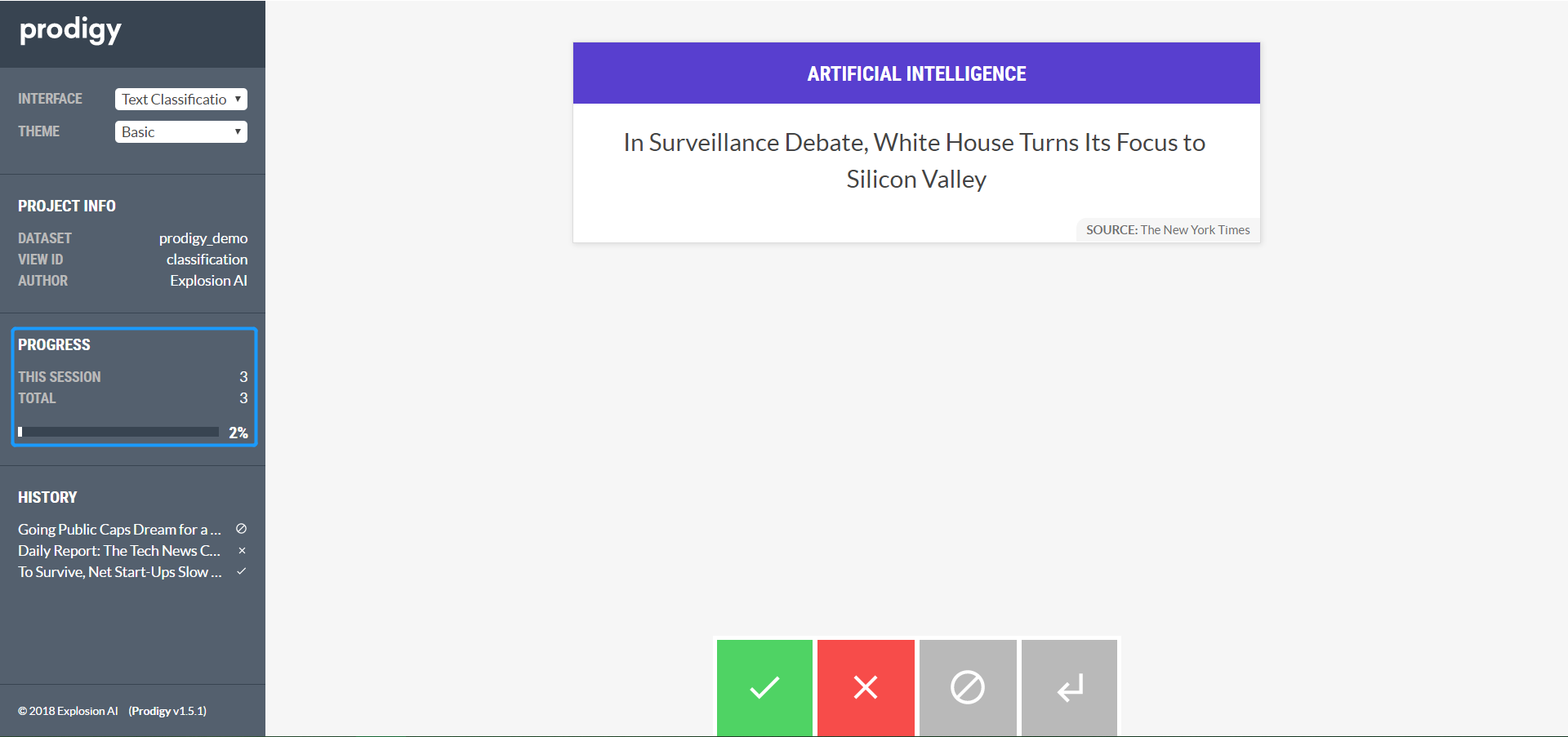
##### 项目信息

项目信息栏应该展示数据集名称，本标注界面的类型，当前标注者等信息。如下图“PROJECT INFO”信息栏所示：



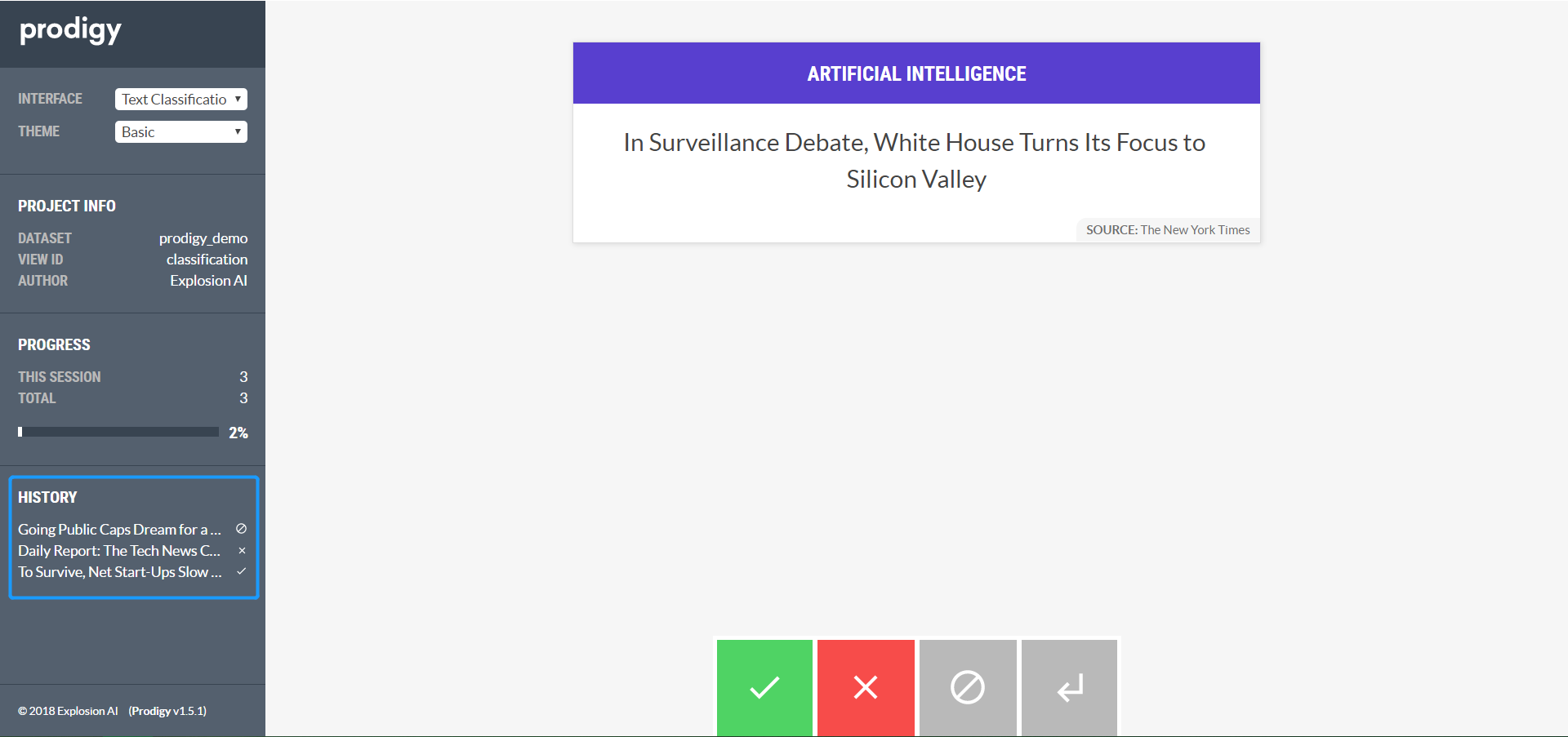
##### 标注进展栏

在标注进展栏中，应该显示当前所标注文本的ID，已标注的文本总数，以及标注进度条等。如下图“PROGRESS”栏所示：



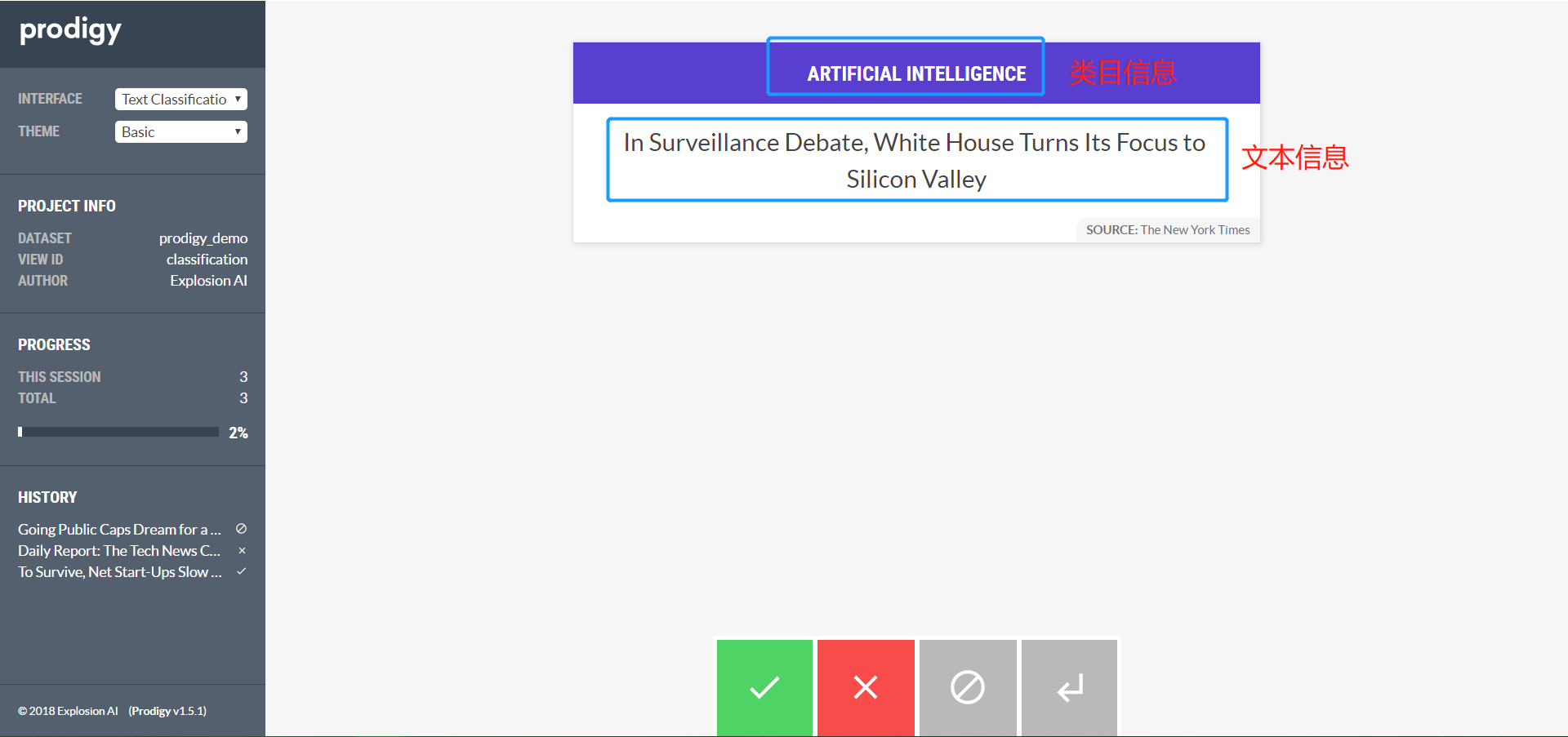
##### 标注历史摘要

显示标注过的句子和标注内容摘要，如果用户需要返回某一句重新标注，则可通过点击句子返回。如下图“HISTORY”栏所示：



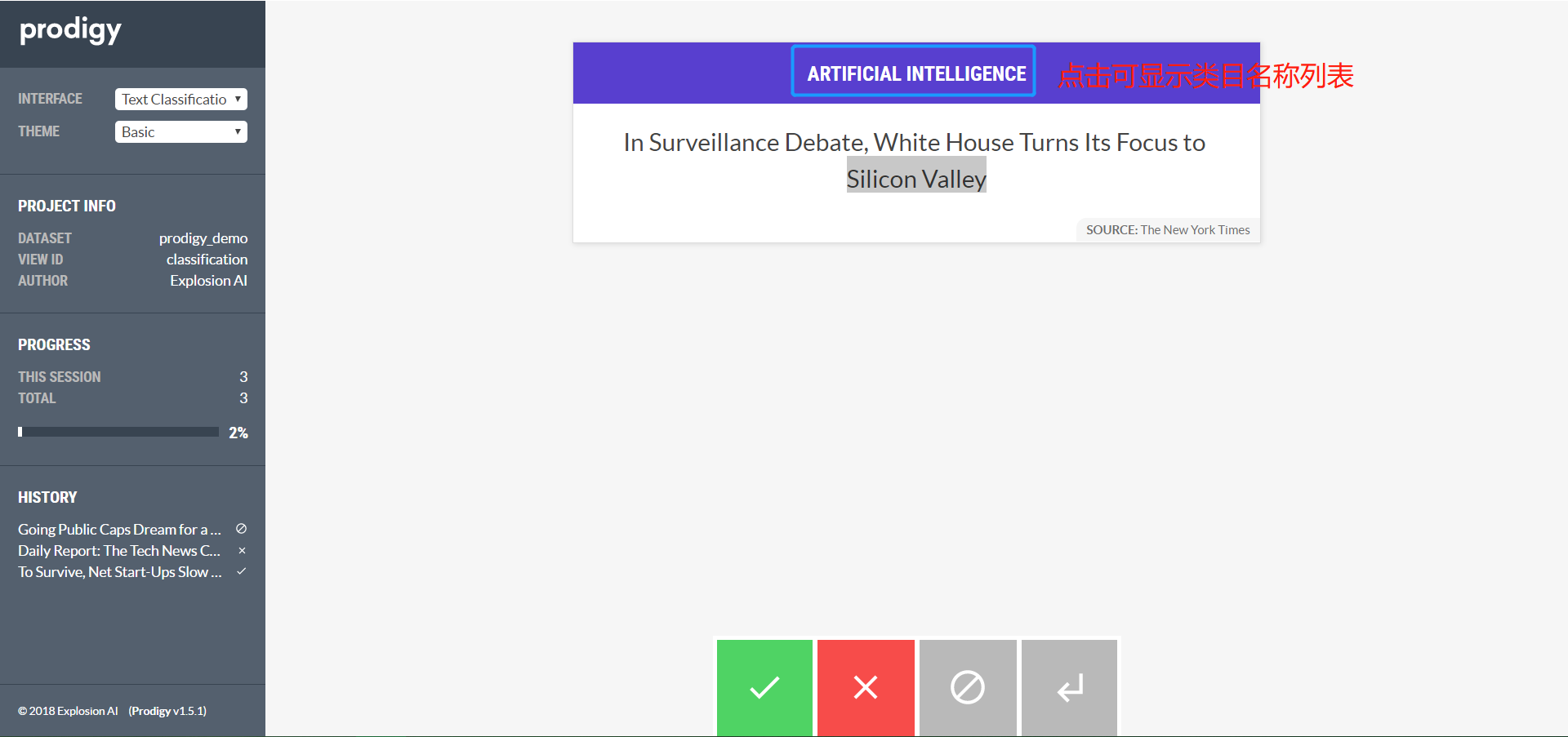
##### 显示类目和文本信息

在标注一个句子时，界面首先展示待标注文本，以及该文本的推荐类目名称。如下图所示：

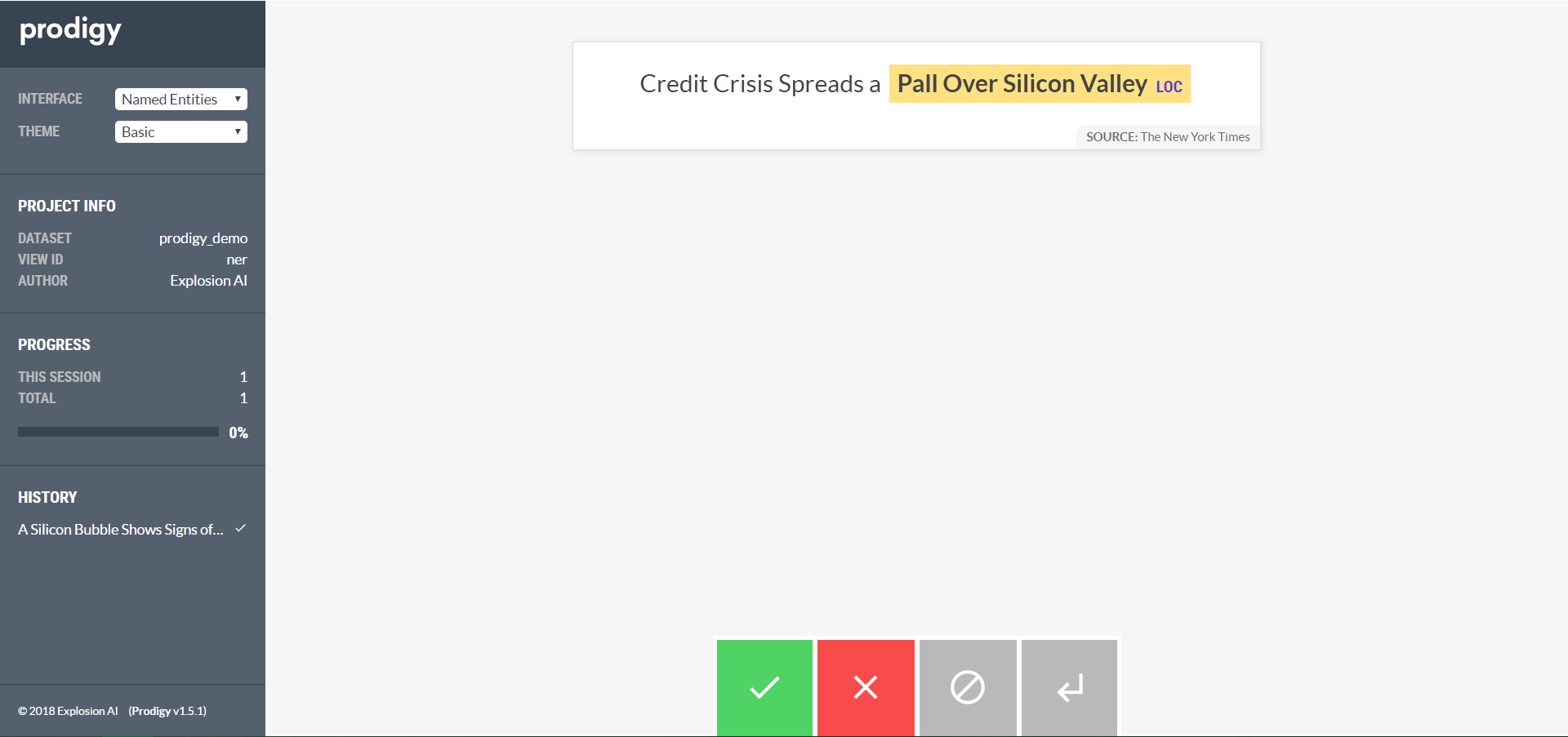


##### 标注类目信息

标注人员如果认为标注系统给出的推荐类目不合适，则可以标注出合适的类目。具体操作是：（1）右键点击类目名称区域（2）在弹出的三级类目列表中选择合适的类目。如下图所示：



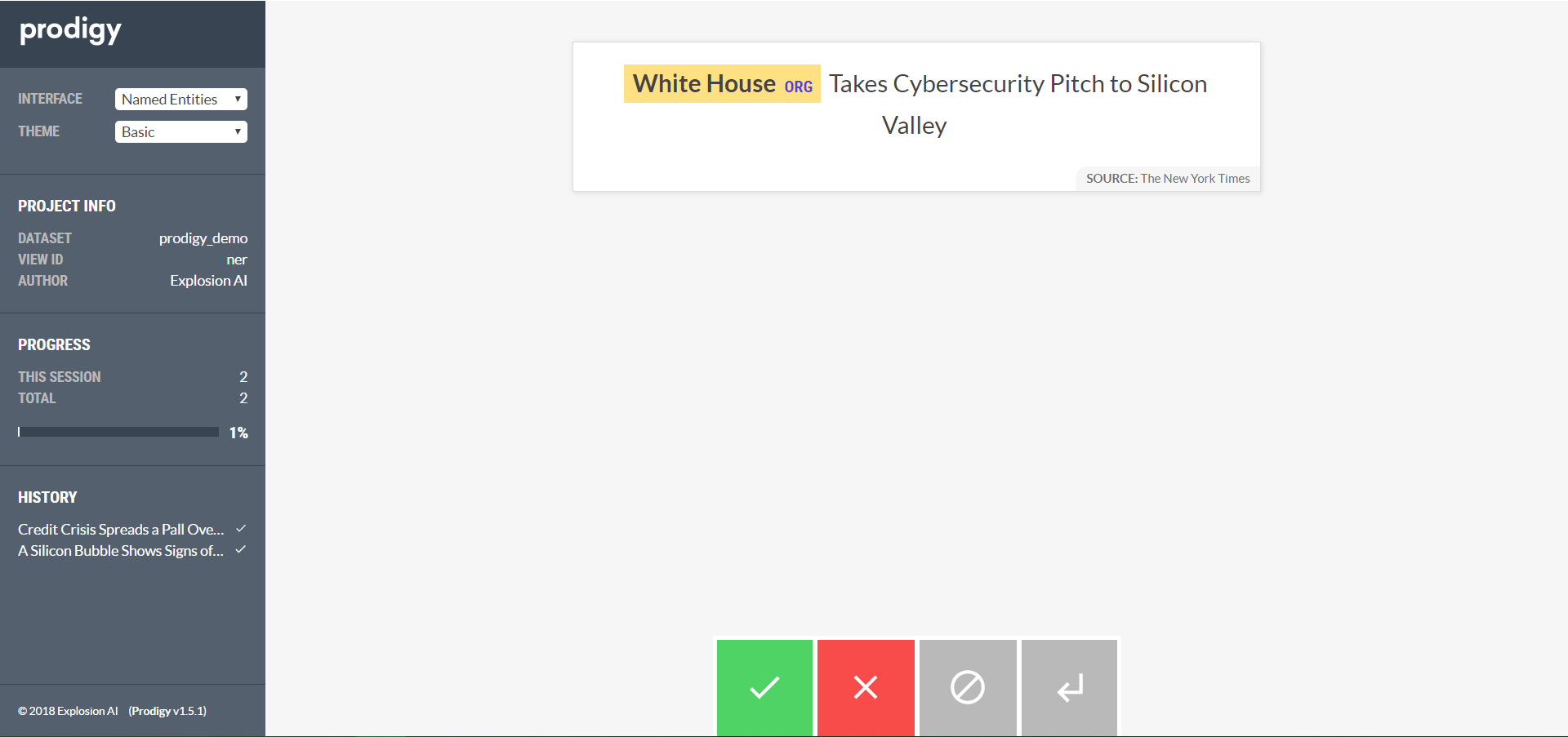
一旦标注人员给出了与系统推荐类目不一致的类目，系统应该强制要求标注人员标注出文本中的类目关键词（组）。用户的具体操作是：（1）选中1个类目关键词（组）（2）点击鼠标右键在弹出的列表中选择“KEYWORDS”选项（3）如果句子中含有多个类目关键词（组），需要重复（1）、（2）两步。标注效果如下图所示：



注意（1）上图为实体标注的界面，在制作标注界面时，需要将该界面的功能添加到“文本分类”标注界面中。（2）上图中黄色区域为类目关键词组，右下角的蓝色字体为词组的类型，对于类目关键词（组）此处应该显示“KEYWORDS ”而不是 “LOC”。

##### 标注实体信息

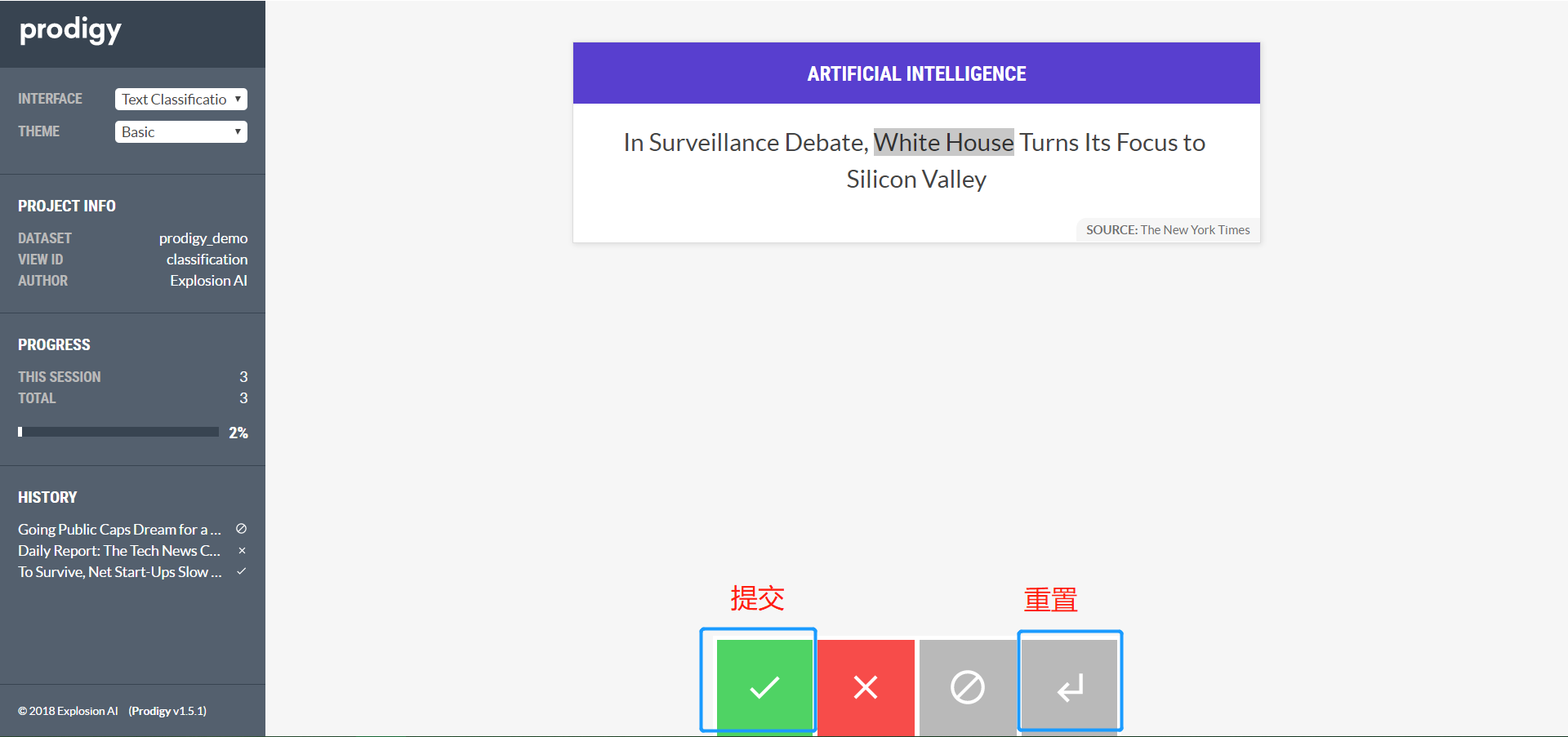
当文本中出现“人名”、“地名”、“机构名”等实体时，标注工具应该为用户提供标注实体的功能。用户的具体操作是：（1）选中实体词或词组（2）鼠标点击右键在弹出列表中选择实体类型，可供选择的类型有（PER、LOC、ORG、DRU、DIS 等）。（3）如果句子中含有多个实体，需要重复（1）、（2）两步。标注效果如下图所示：



##### 提交、重置功能

提交功能：当用户完成标注操作，系统检查必填项（如果用户标注了类目名称，则必须标注类目关键词），确认都已填好后，才允许用户通过点击提交按钮，将标注信息存储到数据库。在下边的示例图中该功能为绿色对勾按钮。

重置功能：当用户需要对标注进行修改时，可以点击重置按钮，此时系统将文本信息恢复到原始状态。在下边示例图中该功能为灰色返回按钮。



#### 标注数据格式

|  |  |  |
| --- | --- | --- |
| 字段名称 | 字段类型 | 字段内容举例 |
| 待标注文本 | 字符串 | 大肠癌预后怎么样？ |
| 类目名称 | 字符串 | 医疗卫生 |
| 类目关键词（组） | 字符串 | 0,6,KEYWORDS,大肠癌 |
| 实体 | 字符串 | 0,6,DIS,大肠癌 |
| 标注人员 | 字符串 | 张三 |
| 标注时间 | 日期 | 20190623 |

注意：上表中“类目关键词（组）”字段和“实体”字段中如果有多个类目关键词（组）或实体，不同类目关键词（组）或实体之间以分号分隔。

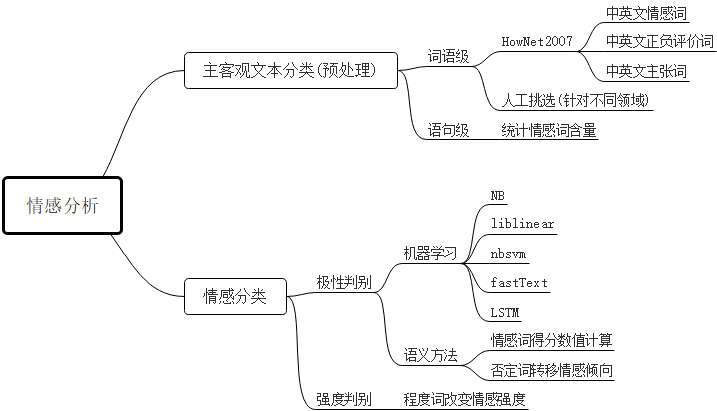
### 标注方法

#### 分类语料标注方法

#### 实体信息标注方法

# 文本情感分析

## 短文本情感分析概述

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，通过统计三类词语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**。

## 情感分析语料和词典资源

### 情感词典资源

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。
5. 《大连理工大学情感本体库》该词典收录27467个情感词，同时给出情感词类别，情感极性值以及情感强度值。

情感词典资源下载链接<https://github.com/baiziyuandyufei/sentiment_analysis/tree/master/data/collected_dictionay>

### 语料库资源

* COAE2014评测语料

task1 长新闻文本中抽取主观句并判断极性。语料未经过标注。

task2 非中文多语种文本极性判别。对中文文本情感分析没有太大用处。

task3 蒙牛评论语料，给定旧词典，要求抽取出新的词语并判断极性。语料极性已标注。领域太窄，没有太大用处。

task4 微博短文本极性判别，正、负、中。语料未标注极性

task5 微博短文本抽取评价对象（实体识别）语料未标注极性。

* COAE2015评测语料

微博短文本极性判别，正、负、中。语料已标注极性。

* 谭松波酒店评论语料

酒店评论语料，可以作为训练语料，已标注。

* 微博评论语料

微博短文本语料，可以作为训练语料，已标注。

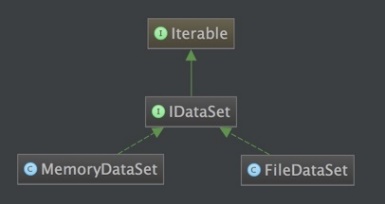
语料资源下载链接 <https://github.com/baiziyuandyufei/sentiment_analysis/tree/master/data/evaluation_corpus>

注：COAE是中文倾向性分析评测会议，能够搜索到的关于该会议的最新消息是2017年的，似乎之后就不再举办了。

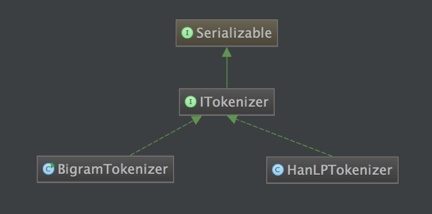
## PyHanLP搭建短文本情感分析系统

### HanLP中有关情感分析的三个类

#### 数据加载存储类



1. MemoryDataSet是将训练数据全部加载到内存的，FileDataSet是利用文件系统存储数据的。
2. 两个类MemoryDataSet和FileDataSet都继承自AbstractDataSet类，这个类中有一个tokenizer实例，它是某一种分词方法实例



这里有两个具体分词类可供使用，其中BigramTokenizer是考虑的2grams字串作为一个词语，HanLPTokenizer使用的是实词分词器。

#### 模型存储类

这里的类结构是有一个AbstractModel，NaiveBayesModel实现了该抽象类。类中主要存储模型中的样本数量，类别数量，特征数量，特征词语，**分词器**，类目名称映射，针对朴素贝叶斯模型还有似然概率以及先验概率。

#### 贝叶斯分类器类

HanLP中有一个分类器接口IClassifier，抽象类AbstractClassifier实现了该接口，所有具体的分类器都继承自抽象类AbstractClassifier，提供独有的训练和预测方法。常用的预测方法为classify给出最可能的类目，predict给出属于每个类目的可能性。

### 训练朴素贝叶斯分类器

首先将得到的训练数据放在合适的位置，对于windows+anaconda的话，可以将数据放在C:\ProgramData\Anaconda3\Lib\site-packages\pyhanlp\static\data\test目录下。加载/训练朴素贝叶斯分类器的代码如下

|  |
| --- |
| **def** train\_or\_load\_classifier():  *# 朴素贝叶斯模型文件名* model\_path = sentiment\_corpus\_path + **'.ser'** *# 检查模型文件是否存在，如果存在则加载模型并返回朴素贝叶斯分类器对象* **if** os.path.isfile(model\_path):  **return** NaiveBayesClassifier(IOUtil.readObjectFrom(model\_path))  *# 模型文件不存在，则首先构建朴素贝叶斯分类器实例* classifier = NaiveBayesClassifier()  *# 传入训练文件路径名称进行训练* classifier.train(sentiment\_corpus\_path)  *# 获取训练后得到的模型* model = classifier.getModel()  *# 保存模型为模型文件* IOUtil.saveObjectTo(model, model\_path)  *# 传入模型到朴素贝叶斯分类器并返回朴素贝叶斯分类器对象* **return** NaiveBayesClassifier(model) |

### 朴素贝叶斯分类器预测

#### 直接预测

|  |
| --- |
| **def** predict(classifier, text):  print(**"《%16s》\t属于分类\t【%s】"** % (text, classifier.classify(text)))  *# 如需获取离散型随机变量的分布，请使用predict接口  # print("《%16s》\t属于分类\t【%s】" % (text, classifier.predict(text)))* |

#### 预测前首先进行主、客观分类

#### 通过“形容词+连词+形容词”给出情感极性

#### 通过“程度副词+情感词”给出情感极性

#### 设置分类的分词器

## 情感分析标注

### 情感类别

|  |  |  |  |
| --- | --- | --- | --- |
| 1级情感类别 | 2级情感类别 | 标记符号 | 举例 |
| 无情感 | 无情感 | WS |  |
| 乐 | 快乐 | PA | 喜悦、欢喜、笑眯眯、欢天喜地 |
| 安心 | PE | 踏实、宽心、定心丸、问心无愧 |
| 好 | 赞扬 | PH | 英俊、优秀、通情达理、实事求是 |
| 相信 | PG | 信任、信赖、可靠、毋庸置疑 |
| 尊敬 | PD | 恭敬、敬爱、毕恭毕敬、肃然起敬 |
| 喜爱 | PB | 倾慕、宝贝、一见钟情、爱不释手 |
| 期望（祝愿） | PK | 渴望、保佑、福寿绵长、万寿无疆 |
| 怒 | 愤怒 | NA | 气愤、恼火、大发雷霆、七窍生烟 |
| 哀 | 悲伤 | NB | 忧伤、悲苦、心如刀割、悲痛欲绝 |
| 失望 | NJ | 憾事、绝望、灰心丧气、心灰意冷 |
| 疚 | NH | 内疚、忏悔、过意不去、问心有愧 |
| 思 | PF | 思念、相思、牵肠挂肚、朝思暮想 |
| 惧 | 慌 | NI | 慌张、心慌、不知所措、手忙脚乱 |
| 恐惧 | NC | 胆怯、害怕、担惊受怕、胆颤心惊 |
| 羞 | NG | 害羞、害臊、面红耳赤、无地自容 |
| 恶 | 烦闷 | NE | 憋闷、烦躁、心烦意乱、自寻烦恼 |
| 贬责 | NN | 呆板、虚荣、杂乱无章、心狠手辣 |
| 憎恶 | ND | 反感、可耻、恨之入骨、深恶痛绝 |
| 妒忌 | NK | 眼红、吃醋、醋坛子、嫉贤妒能 |
| 怀疑 | NL | 多心、生意、将信将疑、疑神疑鬼 |
| 惊 | 惊奇 | PC | 奇怪、奇迹、大吃一惊、瞠目结舌 |
| 有害 | 有害 | NM |  |

情感类别对于情感分析没有太大作用，我们关心的主要还是句子的情感极性，对于更细粒度的情感类别不太关心。

### 情感极性

|  |  |
| --- | --- |
| 情感极性 | 举例 |
| 褒义 |  |
| 贬义（有害） |  |
| 中性 |  |
| 褒贬义 |  |

### 语料标注

#### 标注信息

|  |  |  |
| --- | --- | --- |
| 字段名称 | 字段意义 | 举例 |
| 被评价对象 | 也可以称为情感主题，指人名、地名、机构名、事件名、物体名、服务名、属性等实体 |  |
| 情感极性 | 文本的情感极性，可取值有“正面”、“负面（有害）”、“中性” |  |
| 情感词或短语 |  |  |
| 评价词或短语 |  |  |
| 主张短语 | 现有人工挑选出了XX个主张词语，需要标注出“X+主张词”，或“主张词+X”的2grams短语 |  |
| 程度副词 |  |  |
| 否定副词 | 否定副词对情感极性具有反转作用，因此不应该将否定副词作为停用词去除。如果发现在分词后否定词被滤掉了，应该在否定副词处标记出来 | 原句：当兵我不后悔  分词去停用词后：当兵/vi 后悔/v  显然，“不”字不应该滤除 |
| 表情符号 |  |  |
| **有害信息词或短语** |  |  |
| 转折连词 | 有转折意义的连词。 | 原句：宝宝们下午好！有妈的孩子,个宝，可是我妈离开我二十六年了，婆婆也离开了！我现在是个没娘的孩子！心真的好痛！！！  分词结果：宝宝/n 妈/n 孩子/n 宝/n 妈/n 离开/v 婆婆/n 离开了/v 没/d 娘/n 孩子/n 心/n 真的/d 好痛/nz  这里的“可是”有一种转折的意义，因此应该标注出来。 |
| 可去除的停用词 | 句子中对情感分析没有意义的字、字符、词。此项为可选标注。 | 原句：来了老弟#临沂  句子中的“#”可以被滤掉 |
| 新词或错分词 | 物品词，品牌词，命名实体等新词，不应该被当作停用词去除的词语。 | **例子1：**  原句：国民党空军特级教官李向阳孤身驾机起飞毫无犹豫迎战整个日寇机群@抖音小助手  分词去停用词后：国民党/n 空军/nis 教官/nnt 李向阳/nr 孤身/n 驾机/v 起飞/vi 毫无/v 犹豫/a 迎战/v 日寇/n 机群/n @/nx 抖/vi 音/n 小助手/nz  显然“抖音小助手”是一个抖音的类似账户名的名词，应该成词。  **例子2：**  原句：苹果信号门，苹果X Max PK 苹果X 果断选择京东退货，谁说双卡双待可以用两个电信卡，只有老大才会吧  分词去停用词后：苹果/nf 信号/n 门/n 苹果/nf X/nx Max/nx PK/nx 苹果/nf X/nx 果断/a 选择/v 京东/ntc 退货/vi 说/v 卡/n 电信卡/nz 老大/n 会/v  “双卡双待”是一种手机功能应该被分出来 |

这里，语料标注信息中同时也包含了构建情感词典所需要做的标注，目的是实现做一次人工标注，尽可能得到更多有用的信息。情感词标注时，如果发现了因分词错误或新词导致没有切分出来的情感词，也需要在“情感词或短语”字段中给出。

### 词典构建

#### 情感词典

##### 情感词信息

lexicon=(B,R,E)

B-情感词的基本信息包括：编号、词条、英文、词性、版本。（标注系统应该提供该条信息）

R-情感词的近义词或同义词序列（标注系统提供，人工也可输入）

E-情感词属于给类别的强度，可取值有0，1，3，5，7，9，其中的0表示不属于该类别，9表示最可能属于类别。情感词极性，可取值有褒、贬、中性、褒贬兼有。

##### 构建基础情感词表

基础词汇表选择大连理工大学情感词汇本体库，这里需要对本体库中的词语作筛选。筛选的依据是：（1）只保留情感极性单一的词语（对于含有多种极性的词语可以作为主客观判别的基础词典的一部分）（2）只用情感词来预测已标注语料，保留预测结果准确的词语（注意已标注语料中不应该含有否定副词）。

##### 扩展基础情感词表

用生语料和5grams窗口共现词语法（互信息等）扩展基础情感词汇表，扩展出的词汇描述信息复制被扩展词语的。

##### 检查扩展出的情感词

可以通过人工筛选或者用已有标注语料筛选。

#### 评价词词典

评价词词典中的词语一般出现在含有被评价对象的短文本中。评价词描述了对被评价对象的态度。因此，评价词词典中的一部分词语应该与情感词是重合的，同时也应该注意到会有部分评价词并非是情感词。在标注过程中，如果标记出了被评价对象且句子含有情感极性时，那么应当标记出评价词。

#### 程度副词词典

词汇量小且较为固定，这里选择《中文微博文本的大数据挖掘》中的程度副词词典。

|  |  |
| --- | --- |
| 强度 | 程度词 |
| 1.5 | 最、最为、极、极为、极其、极度、极端、非常非常、真的很、真的十分、真的蛮 |
| 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 |

使用前，注意将程度副词从停用词表中去除，标注过程中发现了不能分出的程度副词时需要将其添加到自定义词典。

#### 否定副词词典

词汇量小且较为固定，这里选择《中文微博文本的大数据挖掘》中的否定副词词典。

|  |  |
| --- | --- |
| 基础词 | 并非、不、不对、不再、不曾、不至于、从不、毫不、毫无、绝非、决非、没、没有、尚未、未、未必、未尝、未曾、永不、不大、不是、无、不太、不是很、没有那么、太不、很不、有点不、有些不、特别不、非常不、 |

使用前，注意将否定副词从停用词表中去除，标注过程中发现了不能分出的否定副词时需要将其添加到自定义词典。

双重否定

|  |
| --- |
| 没有不、不是不 |

使用前，注意将否定副词从停用词表中去除，标注过程中发现了不能分出的否定副词时需要将其添加到自定义词典。

#### 主张词词典

主张词词典中的词语代表文本的创作者想要表达自己的主张。主张词只可作为1种文本主客观的分类特征。应该注意到并不能断言含有主张词的文本就具有主观极性。在标注任务中暂不做主张词语的标注。搜集到的主张词列于下表：

|  |
| --- |
| 察觉、发觉、发现、感觉、感觉到、感受到、见到、觉得、窥见、领教、听说、预感、抱定、感到、看待、认定、认为、认准、想、相信、以为、主张 |

仅通过主张词来判断文本的主客观性有时会产生错误，扩展到含有主张词的2grams短语对主客观判别的准确性会有所提高。 2grams主张短语表的自动候选生成方法为：准备好已标注的训练语料XXX.txt（训练语料中既有主观句也有客观句），生成2grams主张短语（“词语+主张词”、“主张词+词语”），统计2grams主张短语在主观语料和客观语料中的频次，人工筛选2grams主张短语。个人认为主张词以及主张短语并不能准确确定文本的主客观性。作为验证，可以先生成2grams主张短语表，在标注软件中高亮，由人工进行判别。

2grams主张短语

|  |
| --- |
|  |

#### 表情符号词典

在当前网络文本中表情符号也是很常见的。但是从实际语料中来看，表情符号不一定能确定文本情感极性。可以在标注过程结束后抽取具有一定辨识能力的表情符号作为特征词。抽取方法，把非“中文”、“英文”、“数字”、“标点符号”的字符抽取出来，人工筛选。

抽取示例代码

|  |
| --- |
|  |

#### 有害信息词典

#### 转折连词词典

### 标注工具

#### 标注工具调研

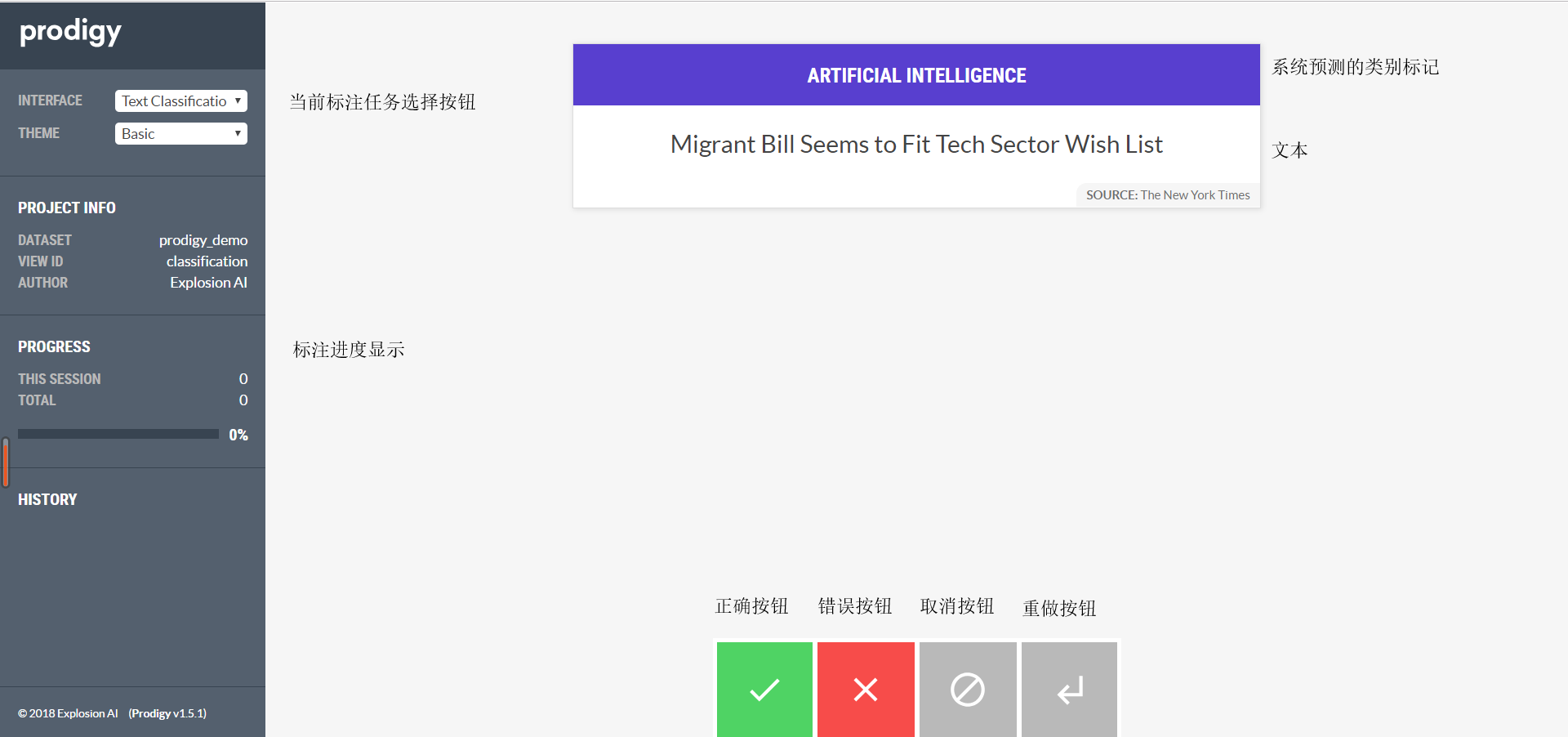
##### BRAT

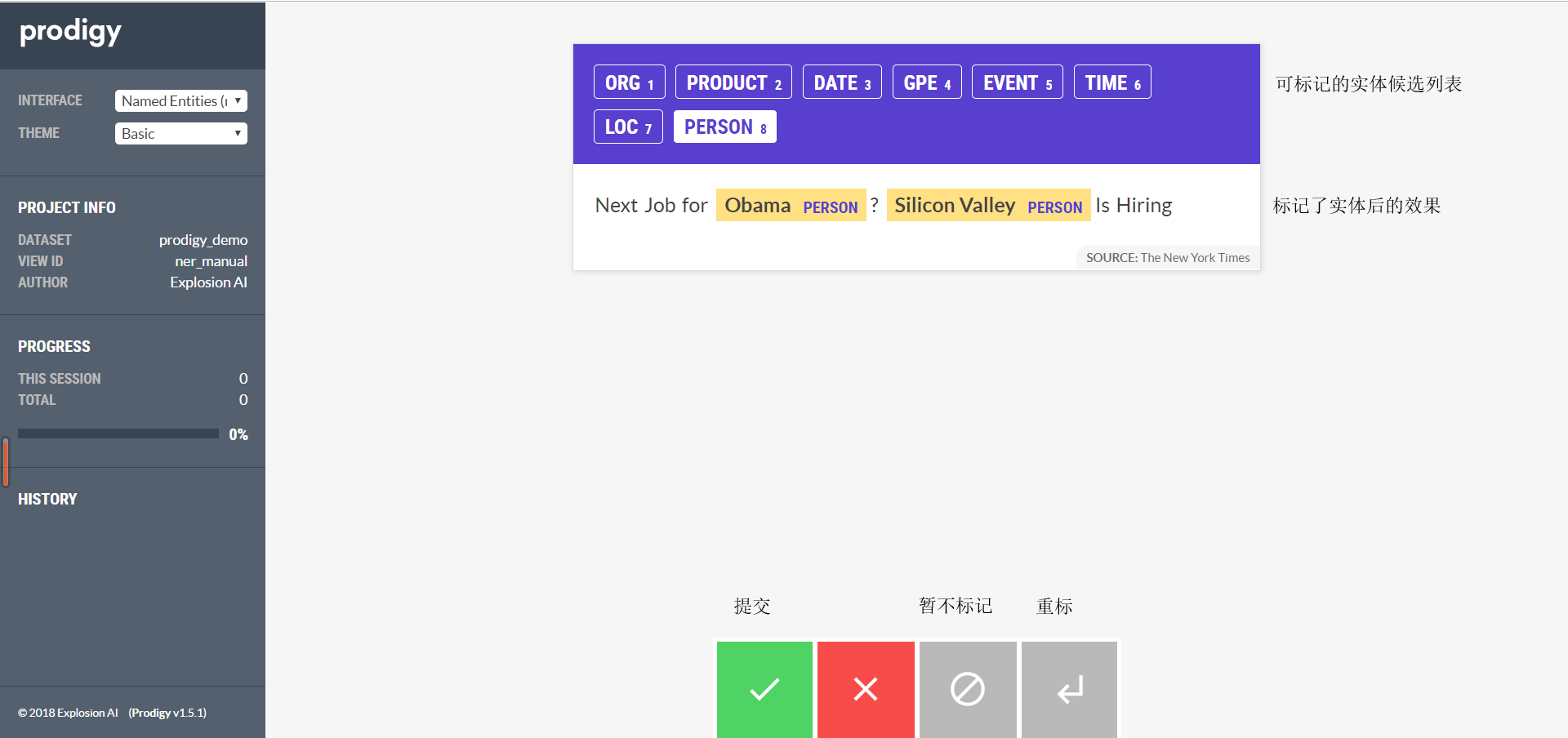
BRAT虽然也是实体、实体关系、事件抽取的标注工具，但是中文里有人利用它标注了情感语料。项目地址：<http://brat.nlplab.org/introduction.html> 。github地址：<https://github.com/nlplab/brat> 。利用BRAT标注中文情感语料csdn地址：<https://blog.csdn.net/owengbs/article/details/49780225> 。

|  |  |
| --- | --- |
|  |  |
|  | Upload.tar.gz按钮无效。官网说明该按钮一直以来没有起作用。 |
| 无法通过普通用户向data目录下发送数据 | $ sudo usermod -aG www-data $(whoami)  上传数据后，再运行一遍  $ sudo chgrp -R www-data data  $ sudo chmod -R g+rwx data work |
| Unable to parse the following line  NOTE: SOME NONPRINTABLE CHARACTERS REMOVED FROM MESSAGE |  |

##### Prodigy

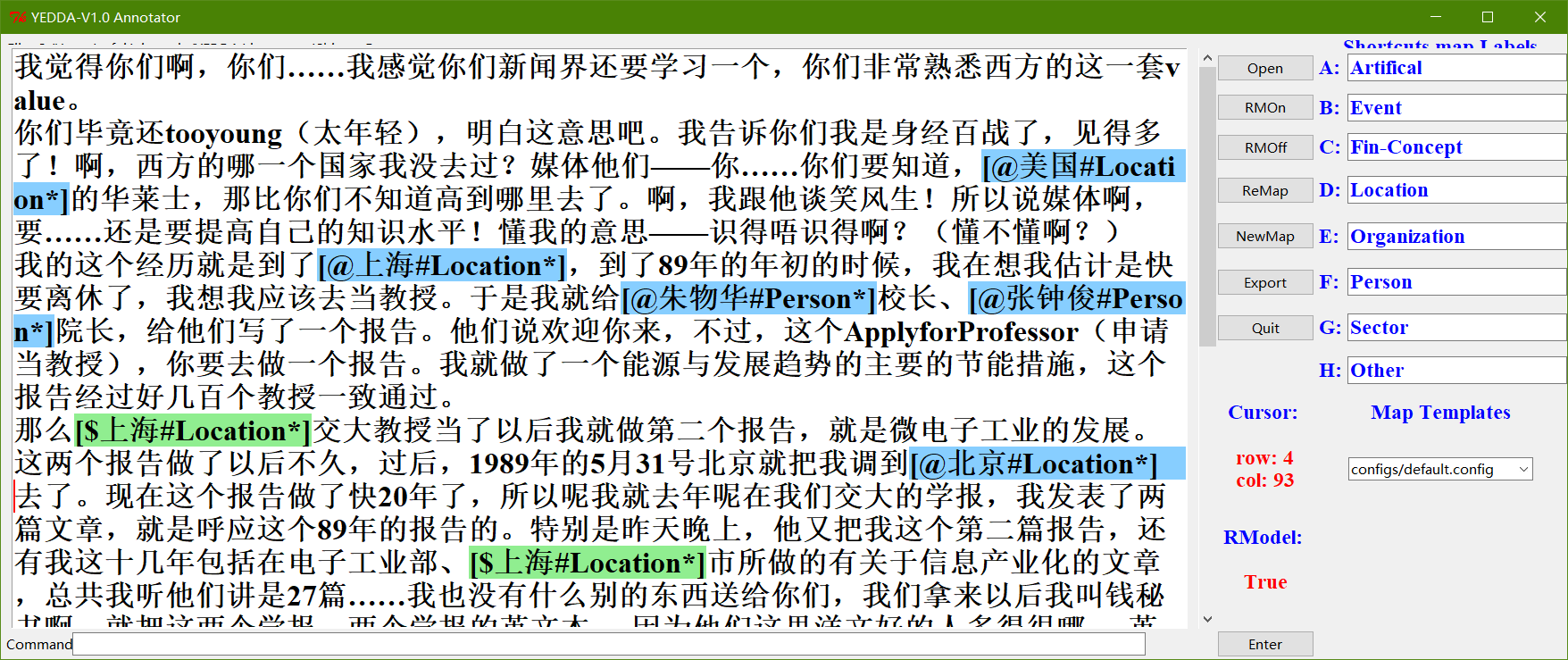
支持实体标注，分类标注，情感标注，但是都是英文的。有一个基于Prodigy的中文github项目。Prodigy官方项目地址：<https://prodi.gy/> 。由于其是**收费**的，所以没有必要再去深究了。这里给出它的文本分类标注界面和实体识别标注界面。





##### YEDDA

YEDDA是本地桌面的标注工具，由于当前只能远程终端访问语料，因此该标注工具也不会再考虑。YEDDA可以进行chunk/entity/event三种标注任务。它使用tkinter开发，当前只支持在Python2.7下运行。对外提供两种用户界面，一种是标注员界面（终端执行python YEDDA.py），一种是管理员界面（终端执行python YEDDA\_Admin.py）。标注员界面用于给句子做标注，管理员界面提供针对同一文件的不同人员标注结果的对比等功能。标注界面如下所示



其中的按钮的大致功能为`Open`按钮用于打开待校对的文件。`ReMap`用于将用户设置的标注快捷键更新到当前配置文件。`NewMap`按钮用于将用户设置的标注快捷键保存到新的配置文件。`RMOn`是打开机器自动标注功能，`RMOff`是关闭机器自动标注功能。`Export`到处当前标注后的文本。`Quit`退出当前标注。字母`A`,`B`,`C`,`D`,`E`,`F`,`G`分别对应右侧所列功能的快捷键，比如Location地名的快捷键为`D`。`Cursor`表示当前光标在文本中的为止，`RMModel`指的是是否启动了机器自动标注。`Map Templates`快捷键配置文件名。YEDDA的缺点有：（1）可标记种类数只有7种（2）没有情感类别或分类类别的标记功能（3）一次性将文本内容全部展示出来不利于标注者标记（4）没有分词功能也不能使用自己的分词器（5）没有类别预测功能（6）只能运行在Python2.7版本下。可改进的点（1）添加类别预测展示及标注功能（2）添加标注进度展示区（3）添加分词功能（4）添加实体识别功能（5）导出时，计算本次标注后系统预测的准确率。

#### BRAT标注工具的使用

##### 安装与部署

<https://zhuanlan.zhihu.com/p/65101293>

##### 数据的导入

##### 标注的配置

标注工具的基本配置可以查看BRAT官网的配置节<http://brat.nlplab.org/configuration.html>。所有自定义的配置文件都应该放在数据所在目录中。本小节的参考：

<https://blog.csdn.net/tcx1992/article/details/80580089>

1. annotation.conf的配置

* 实体定义

1. 基本格式

|  |  |
| --- | --- |
| [entities]  OTH  ORG  SENPOL  SENCLASS |  |

1. 层级格式

|  |  |
| --- | --- |
| [entities]  LOC  continent  nation  province  scenic |  |

1. 禁止父级实体标注

|  |  |
| --- | --- |
| [entities]  !NAME  per  animal  plant |  |

* 属性定义

1. 二元属性值

|  |  |
| --- | --- |
| [attributes]  polarity Arg:<ENTITY> |  |

1. 多元属性值

|  |  |
| --- | --- |
| [attributes]  class Arg:<ENTITY>,Value:Education|Military|Financial|Disaster |  |

### 标注工具应具备功能

标注工具应该提供以下的基本功能，（1）输入文本的分词结果（2）输入文本的词性标注结果（3）高亮情感词（4）高亮评价词（5）高亮主张词（6）高亮程度副词（7）高亮否定副词。

人工标注出的新词应该直接添加到CustomDictionary.txt文件中，同时应该加入验证是否生效功能，因为HanLP添加到CustomDictionary.txt中的词语不一定能分出来。

#### 高亮主张短语

标注软件需要对输入文本中的在主张短语表中列出的2grams给予高亮显示或者在单独输出对话框中进行列出。

## 标注方法

情感词标注时，如果发现了因分词错误或新词导致没有切分出来的情感词，也需要在“情感词或短语”字段中给出。

### 主张短语显示

## 标注示例

## 否定副词对情感极性的影响

否定副词对于情感极性具有转折作用，如果否定副词在主观语料中的出现频次不多，则它会干扰分类器的准确判别。因此，这里需要对标注的语料中的主观文本中的否定副词出现频次进行统计，如果确实出现频次较低，则应该从训练语料中去除含有否定副词的文本。这里根据直观经验，我们大致猜测含有否定副词的主观文本的出现频率应该不会太大。

## 表转折的连词对情感极性的影响

表转折关系的连词连接了两个在意义上有冲突的小句。针对主观文本来说，如果含有了转折关系连词，则整个句子的情感极性应当以后边句子为主。因此，在构建训练语料时，也应该将含有转折关系连词的句子去掉。

## 疑问句的情感分析

## 反讽表达

## 造因-情感-反应框架

# 实践经验

## 实践经验

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。
13. 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。
14. 分类训练语料一旦确定后，没有特别明显的badcase不要改变语料特别是不要加入含有多个类目关键词的句子。特别不要因为某一两个极个别的badcase而改动语料。
15. **混淆矩阵在评价多类问题性能时很好用**。混淆矩阵**对角线元素的元素值**较非对角线元素的元素值**大**很多时，说明分类器**整体效果较好**。单独从某**列**看，如果某列j上除了对角线元素外还存在一个**非对角线元素i**，i的值较列中其他元素（非对角线元素）的元素值**大**很多，则说明分类器对于**区别类别j和类别i的能力不够**，可以考虑丰富两个类别的数据以使特征词具有区分性或者直接将两个类别合并为一个类别。
16. 可以通过设定预测的各个类目的概率方差值提高准确率，当然代价是牺牲召回率。

### 特征选择保留词语数量不能太少

特征选择后保留特征数量为3530 / 46483 = 7.59%时的准确率和召回率。

|  |
| --- |
| precision recall f1-score support  社会保障 0.00 0.00 0.00 14  旅游事务 0.00 0.00 0.00 36  农村农业 0.86 0.01 0.02 6961  部队军事 0.98 0.13 0.23 8517  交通运输 1.00 0.00 0.00 21110  医疗卫生 0.97 0.00 0.01 26490  学校教育 0.00 0.00 0.00 14779  住房城建 0.00 0.00 0.00 9  文化艺术 0.92 0.00 0.00 21105  avg / total 0.81 0.01 0.02 99021 |

特征选择后保留特征数量为20402 / 46598 = 43.78%时的准确率和召回率。

|  |
| --- |
| precision recall f1-score support  部队军事 0.97 0.19 0.32 8517  社会保障 0.00 0.00 0.00 14  交通运输 0.93 0.15 0.26 21110  农村农业 0.88 0.38 0.54 6961  住房城建 0.09 0.33 0.14 9  旅游事务 0.03 0.17 0.05 36  学校教育 0.95 0.13 0.22 14779  文化艺术 0.89 0.07 0.13 21105  医疗卫生 0.99 0.01 0.03 26490  avg / total 0.94 0.11 0.19 99021 |

### 特征选择算法不是万能的

不要指望通过特征选择就可以构成特征向量词表了，特别是语料少。即使语料充足，如果训练语料不是由实际预测的文本中构成的，依然会因为某些词而导致badcase，只不过看实际预测文本与训练语料文本的相似性了。比如下边的这个例子：

[1] 高血压患者注意事项，为爸妈收藏起来吧！

系统预测该文本属于“艺术”类，它本应该属于“医疗”类，造成误分类的原因是“收藏”这个词在“艺术”类里是个高频词，而在医疗类里几乎没有出现过。解决方法就是将收藏这个词从特征向量词表中去除掉。或者需要增加有关“收藏”这个词的“医疗”类文本语料。从这个例子我们也可以看出，特征选择算法强烈依赖于训练语料的完备性。

### 朴素贝叶斯预测概率时防止溢出

朴素贝叶斯分类模型计算文本属于某个类别的概率时需要取对数变相乘为相加，防止预测结果溢出，同时最好不要将最后的概率值转换回为概率值，因为概率值太小依然可能会溢出。

### 类目关键词构造方法

构建类目关键词时，（1）如果有标注的训练语料，类目关键词从训练语料抽取。（2）如果没有标注的训练语料，但是有待预测语料，类目关键词通过对语料聚类后由簇关键词抽取。（3）完全没有语料，则根据类目人工列举。这会带入列举人对类目的理解以及列举人的文本经验储备。比如“艺术”这个类目，我会很快想到古玩字画收藏，但是却忽略了很多带有广告性文本（比如药膳推荐）会在文本的最后加上“建议收藏”等等。我们在做语料提纯的时候，是用类目关键词来过滤的，因此“收藏”这个词肯定是“艺术”类文本集中的高频词，而在“医疗”类目中我们没有想到“收藏”，于是“收藏”这个词或者没有在“医疗”类中出现，或者出现次数极少。当待预测文本中“收藏”成为核心词语的时候，势必就会出现badcase。

## 实用脚本

### 文本处理实用脚本

#### 等间隔采样

#### 文件2目录

#### 目录2文件

### 文本处理Shell命令

## 数据收藏

### Boson数据集

Boson数据的下载地址为<https://bosonnlp.com/dev/resource>。所提供的数据集有：

1. 11款开放中文分词引擎测试数据
2. 命名实体数据
3. 情感词典
4. 字频数据

## 常用各语言代码

### Java入口参数使用

应该注意到Java的入口参数索引从0开始编号。

|  |
| --- |
| **if**(args[0].equals("debug")) { // 调试模式  ;  }**else** {  ;  } |

### Java从终端读取字符串

|  |
| --- |
| Scanner s = **new** Scanner(System.***in***);  **while**(**true**) {  System.***out***.println("please input your text:");  String text = s.nextLine();  }  s.close(); |

# 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 |

sklearn实践

|  |
| --- |
| In [1]: from sklearn.datasets import load\_files  # 加载数据  In [2]: data = load\_files(container\_path="../data/news", # 分类语料根目录  ...: categories=['Agriculture', 'Communication', 'Education',  ...: 'Electronics', 'Medical', 'Sports'], # 需要加载的类目名称列表  ...: encoding="gbk", decode\_error="ignore") # 文件编码  ...:  # 打印文本数据  In [3]: for text\_ser, text in enumerate(data.data[:5]):  ...: print("第%d篇文本" % (text\_ser+1))  ...: print(text)  ...:  第1篇文本  中国选手再夺加拿大杯游泳赛,枚奖牌,新华社渥太华,月,日电,记者廖振云,历时,天,的,年加拿大杯国际游泳赛,日在温哥华结束,中国队派出,名女选手参赛,共获,枚金牌,枚银牌和,枚铜牌,取得自,年参加这项赛事以来的最好成,绩,在,日进行的颁奖式上,中国选手林莉被评为本届,杯赛唯一的女子,最佳运动员,同时被授予,得分最高,者,奖,她在这次比赛中共获得两枚金牌,在,日的女子组比赛中,中国选手共获得,枚金牌,和,枚银牌,岁的林莉和,岁的阎明分别夺得,米个人混合泳金牌和银牌,林莉还以,分,秒,的,成绩刷新了这个项目的赛会 纪录,岁的孙春莅以,秒,的成绩夺得,米自由泳的金牌,岁的李洁,以,分,秒,的成绩获得,米仰泳的金牌,中国,队还以,分,秒,的成绩获,米混合泳接力,赛的第一名,并打破了这个项目的赛会纪录,这次杯赛是,日开始的,参加比赛的除中国选手外,还有瑞典,澳大利亚,美国,英国,新西兰及东道主加,拿大等国的好手,完  # 加载到内存中文件的路径名称列表  In [4]: # 加载到内存中文件的路径名称列表  ...: data.filenames  Out[4]:  array(['../data/news\\Sports\\61.txt',  '../data/news\\Agriculture\\C32-Agriculture0893.txt',  '../data/news\\Agriculture\\C32-Agriculture0919.txt', ...,  '../data/news\\Sports\\128.txt',  '../data/news\\Agriculture\\C32-Agriculture1120.txt',  '../data/news\\Agriculture\\C32-Agriculture1370.txt'], dtype='<U50')  # 加载到内存中文件的类别序号label  In [5]: data.target  Out[5]: array([5, 0, 0, ..., 5, 0, 0])  # 每个类别label对应的类别名称列表  In [6]: data.target\_names  Out[6]:  ['Agriculture',  'Communication',  'Education',  'Electronics',  'Medical',  'Sports'] |

#### load\_iris

加载并返回iris数据集（分类）。鸢尾花数据集是一个经典的用于分类的数据集。数据集详细情况如下：

================= ==============

Classes 3

Samples per class 50

Samples total 150

Dimensionality 4

Features real, positive

================= ==============

**原型**

**sklearn.datasets.load\_iris(return\_X\_y=False)**

**参数**

|  |
| --- |
| * **return\_X\_y**： boolean, 默认为False。如果为True，则返回(data,target)；如果为False，则返回捆绑对象。向下看以发现更多有关data和target对象的信息。 |

**返回**

|  |
| --- |
| * **data**：Bunch类型，类字典对象，拥有的属性有： * ’data’，训练数据矩阵 * ’target’，类别label（索引号） * ’target\_names’，类别名称列表，类别名称所在索引即为类别label * ’feature\_names’，特征的意义，数据矩阵中各列的特征名称 * ’ DESCR’，数据集的完整描述 * **(data, target)** ：tuple 当 **return\_X\_y** 是True时 |

### 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 |

## 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频次特征都被置为1。对于离散概率模型来说，必须将该参数设置为True。因为离散模型为二元事件模型而非计数模型。 * **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次外，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：**该函数取代preprocessor和tokenizer两个参数所要做的处理。默认的analyzer会分别调用preprocessor和tokenizer两个参数对象来对文本做处理，但是用户定制的analyzer不会再调用preprocessor和tokenizer。N-gram抽取和停用词过滤都是需要analyzer对象内处理的，因此analyzer必须完成以上所述的这些任务。** * **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。**该参数接受的是一个函数对象，该函数的输入是用单一字符串表示的一篇文本，函数内部对输入的字符串做预处理比如去掉HTML标签，将字母大写转成小写等等，函数的返回值仍任是一个经过预处理后的字符串** * **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。**该参数接受的是一个函数对象，该函数的输入参数是preprocessor的输出字符串，函数内部对输入的串进行分词，函数的返回值是词语列表。**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（df值）由fit函数或fit\_transform函数学习到，拿一篇测试文档向量化举例，首先从训练得到的模型中取得特征词汇表，接着计算每个词的权重，tf\*idf，这里tf是词语在本测试文档中的频次，idf为词语在训练得到的模型中的idf值。对于短文本来说，tf值基本上不是0就是1，那么唯一的区分就在于idf值了 |

## 特征选择

sklearn.feature\_selection模块实现了特征选择算法。目前它实现了单变量过滤选择方法和递归特征淘汰算法。sklearn.feature\_selection模块中的类可用来在样本集上实现特征选择和维度规约，既可以提高估计器的准确率也可以提升系统在高维数据集上的性能。

### VarianceThreshold

设置方差阈值是一种简单的特征选择baseline方法。它移除那些方差不满足阈值的特征。默认情况下，它移除0方差特征，即某一特征在所有样本中的取值都相同时该特征将被移除。

原型

class sklearn.feature\_selection.VarianceThreshold(threshold=0.0)

参数

|  |
| --- |
| * threshold：浮点数，可选的。训练集中方差值低于该值的特征将被移除。默认值为0即保留方差值大于0的特征，即当特征的所有样本值都相同时该特征将被移除。 |

属性

|  |
| --- |
| * variances\_：数组，形状为(n\_features,)，每个独立特征的方差值 |

方法

|  |
| --- |
| * fit(X[, y])：学习数据矩阵X各列的经验方差 * transform(X)：在数据矩阵X上执行特征选择 * fit\_transform(X[, y])：学习经验方差并执行特征选择 * get\_support([indices])：获取选出的特征的mask或整数索引列表 |

sklearn实践

|  |
| --- |
| In [1]: from sklearn.feature\_selection import VarianceThreshold  #定义数据矩阵  In [2]: import numpy as np  In [3]: X = np.array([[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1, 1]])  In [4]: X  Out[4]:  array([[0, 0, 1],  [0, 1, 0],  [1, 0, 0],  [0, 1, 1],  [0, 1, 0],  [0, 1, 1]])  # 构建VarianceThreshold特征选择实例  In [5]: sel = VarianceThreshold(threshold=(.8 \* (1 - .8)))  # 学习矩阵X各列经验方差并进行特征选择  In [6]: sel.fit\_transform(X)  Out[6]:  array([[0, 1],  [1, 0],  [0, 0],  [1, 1],  [1, 0],  [1, 1]])  # 返回选出特征的整数索引列表  In [7]: sel.get\_support(indices=True)  Out[7]: array([1, 2], dtype=int64)  # 返回选出特征的mask  In [8]: X[:,sel.get\_support()]  Out[8]:  array([[0, 1],  [1, 0],  [0, 0],  [1, 1],  [1, 0],  [1, 1]]) |

### chi2

用于计算每个非负特征与类别之间的chi2统计量。该得分可以用来从数据矩阵X中选择chi2值最高的n\_features个特征，数据X只能包含相对于类的非负特征，如布尔值或词项频度。回想一下，卡方检验衡量的是随机变量之间的相关性，因此使用这个函数“剔除”了最有可能独立于类的特征，因此这些特征与分类无关。

原型

sklearn.feature\_selection.chi2(X, y)

参数

|  |
| --- |
| * X：类数组，稀疏矩阵。形状=(n\_samples, n\_features\_in)，样本向量 * y：类数组，形状=(n\_samples,),每行数据的类别 |

返回

|  |
| --- |
| * chi2：数组，形状=(n\_features,)，每个特征的卡方统计值 * pval：数组，形状=(n\_features,)，每个特征的p-values |

算法的复杂性为O(n\_classes\*n\_features)

### SelectKBest

按照K最高得分进行特征选择

例：使用卡方特征选择方法，对鸢尾花数据集只保留2个特征。

|  |
| --- |
| In [9]: from sklearn.datasets import load\_iris  In [10]: from sklearn.feature\_selection import SelectKBest  In [11]: from sklearn.feature\_selection import chi2  In [12]: iris = load\_iris()  In [13]: X, y = iris.data, iris.target  In [14]: X.shape  Out[14]: (150, 4)  **In [15]: X\_new = SelectKBest(chi2, k=2).fit\_transform(X, y)**  In [16]: X\_new.shape  Out[16]: (150, 2) |

### SelectFromModel

SelectFromModel将分类模型中的系数向量中各维度值看作对应词语权重，值越小认为词语对分类表征越小，用户设定一个阈值，所有权重小于该阈值的词语都将被移除，从而达到特征选择的目的。因此，如果要使用SelectFromModel类，那么所用分类模型必须具有coef\_属性或feature\_importances\_属性。除了可以指定阈值外，该类还支持用权重均值或权重中值或两者的倍数作为阈值。

原型

class sklearn.feature\_selection.SelectFromModel(estimator, threshold=None, prefit=False, norm\_order=1)

参数

|  |
| --- |
| * estimator：对象。构建特征选择实例的基本分类器。如果参数prefit为True，则该参数可以由一个已经训练过的分类器初始化。如果prefit为False，则该参数只能传入没有经过训练的分类器实例 * threshold：字符串，浮点数，（可选的）默认为None。该参数指定特征选择的阈值，词语在分类模型中对应的系数值大于该值时被保留，否则被移除。如果该参数为字符串类型，则可设置的值有”mean”表示系数向量值的均值，”median”表示系数向量值的中值，也可以为”0.x\*mean”或”0.x\*median”。当该参数设置值为None时，如果分类器具有罚项且罚项设置为l1，则阈值为1e-5，否则该值为”mean” * prefit：布尔类型。默认值为False。是否对传入的基本分类器事先进行训练。如果设置该值为True，则需要对传入的基本分类器进行训练，如果设置该值为False，则只需要传入分类器实例即可 * norm\_order：非零的整数。该值设置为类别数-1 |

属性

|  |
| --- |
| * estimator\_：对象。基本分类器。只有不对分类器进行预训练时该属性才有值 * threshold\_：浮点数 |

方法

|  |
| --- |
| * fit(X[, y])：对基本分类器进行训练 * fit\_transform(X[, y])：对基本分类器进行训练并执行特征选择 * get\_support([indices])：获取选出的特征的索引序列或mask * partial\_fit(X[, y])：对基本分类器的训练只进行一次 |

## 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，也就是每个词项在每个类别下的似然概率值 * **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（不同之处在于这个超平面的求法）。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 |

## 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]。计算得到的混淆矩阵，混淆矩阵各行号对应实际类别，各列号对应预测类别 |

实例化confusion\_matrix类时，应该直接指定y\_true，y\_pred和labels，以免造成类别对应错误。

### 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>

**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

HanLP提供了多种词法分析工具。这里介绍基于viterbi思想的最短路径分词器，由于使用了viterbi的思想求解最短路径，因此在HanLP中也称它为viterbi分词器。着重介绍viterbi分词器的原因是HanLP对外封装的很多开箱即用的分词器都是基于最短路径分词器做的。viterbi分词器属于基于词语的分词器，当输入文本很长时，内部自动对文本进行分句，并采用多线程技术对多个小句并行分词。这里我们介绍封装了viterbi分词器的实体分词器。

### 实词分词器NotionalTokenizer

在进行文本分类（非情感分类）时，我们经常只保留实词（名、动、形）等词，为了文本分类的分词方便，HanLP专门提供了实词分词器类NotionalTokenizer，同时在分类数据集加载处理时，默认使用了NotionalTokenizer分词器。

**HanLP中涉及NotionalTokenizer的类及方法有：**

1. AbstractDataSet.java文件中的AbstractDataSet方法
2. HanLPTokenizer.java文件中的segment方法
3. NotionalTokenizer.java文件中的segment方法

这里的1、2两个类是HanLP文本分类器所使用的数据加载及预处理的类，在预处理分词中所使用的就是NontionalTokenizer类的实例。

**NotionalTokenizer类的主要实现步骤：**

1. 初始化了一个维特比分词器实例（最短路径方法，用viterbi思想实现）
2. 用CoreStopWordDictionary类的shouldInclude方法对维特比分词结果进行过滤，该方法只保留属于名词、动词、副词、形容词并且不在停用词表中的词。详见CoreStopWordDictionary.java文件中的shouldInclude(Term)方法

**HanLP提供的使用示例：**

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

<https://github.com/hankcs/HanLP/blob/master/src/test/java/com/hankcs/demo/DemoNotionalTokenizer.java>

### 定制用户词表

HanLP.segment和NotionalTokenizer两个类别的方法都允许用户自定义分词词表。在文本分类中有的badcase是由于分词错误导致的，此时最有效的方法便是将未登录的新词更新到未登录词表中。

**自定义词表主要要做两个步骤：**

1. 将新的未登录词补充到pyhanlp\static\data\dictionary\custom\CustomDictionary.txt文件中
2. 删除pyhanlp\static\data\dictionary\custom\CustomDictionary.txt.bin文件，运行分词代码后即可自动生效

**HanLP提供的使用示例：**

1. <https://github.com/hankcs/HanLP/blob/master/src/test/java/com/hankcs/demo/DemoCustomDictionary.java>
2. <https://github.com/hankcs/HanLP/blob/master/src/test/java/com/hankcs/demo/DemoCustomNature.java>
3. <https://github.com/hankcs/pyhanlp/blob/master/tests/demos/demo_custom_dictionary.py>
4. <https://github.com/hankcs/pyhanlp/blob/master/tests/demos/demo_custom_nature.py>

### 定制停用词词表

#### 自定义停用词表

**自定义停用词表主要做两个步骤：**

1. 将停用词添加到pyhanlp\static\data\dictionary\stopwords.txt
2. 删除pyhanlp\static\data\dictionary\stopwords.txt.bin，运行分词代码后即可自动生效

#### 自定义词语过滤方法

用户可以通过编写“pyhanlp\static”路径下的“MyFilter.java”文件设置自己的词语过滤方法。应当注意这里处理的语言单位是词语，而不是字。编辑完毕后需要编译该文件并生成字节码文件，之后运行CoreStopWordDictionary.apply方法时就会自动调用用户自己的词语过滤方法了。这里给出一个自定义过滤方法的编写示例代码。

这里给出一个示例：

|  |
| --- |
| import os  from pyhanlp.static import STATIC\_ROOT, HANLP\_JAR\_PATH  java\_code\_path = os.path.join(STATIC\_ROOT, 'MyFilter.java')  with open(java\_code\_path, 'w') as out:  java\_code = """  import com.hankcs.hanlp.dictionary.stopword.CoreStopWordDictionary;  import com.hankcs.hanlp.dictionary.stopword.Filter;  import com.hankcs.hanlp.seg.common.Term;  public class MyFilter implements Filter  {  public boolean shouldInclude(Term term)  {  if (term.nature.startsWith('m')) return false; // 数词过滤  if (term.nature.startsWith('q')) return false; // 量词过滤  if (term.nature.startsWith('t')) return false; // 时间词过滤  if (term.nature.startsWith("w")) return false; // 过滤标点符号  return !CoreStopWordDictionary.contains(term.word); // 停用词过滤  }  }  """  out.write(java\_code)  os.system('javac -cp {} {} -d {}'.format(HANLP\_JAR\_PATH, java\_code\_path, STATIC\_ROOT)) |

HanLP提供的使用示例：

1. <https://github.com/hankcs/pyhanlp/blob/master/tests/demos/demo_stopword.py>
2. <https://github.com/hankcs/HanLP/blob/master/src/test/java/com/hankcs/demo/DemoStopWord.java>

## 实体识别

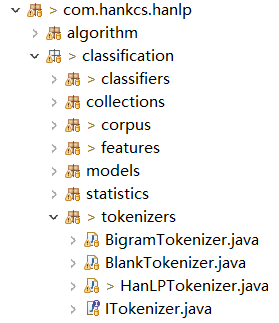
## 短语抽取

## 文本分类

### 分类模块的分词器

如下图所示，HanLP的分类模块中单独封装了适用分类的分词器，当然这些分词器都是对HanLP提供的分词器的封装。分类模块中提供的分词器都在tokenizer包中。包括：

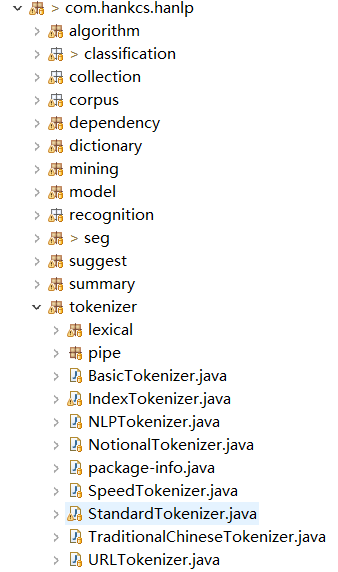
1. BigramTokenizer这是一个2gram分词器，也就是把连续的2字认为是一个词
2. BlankTokenizer这是一个空白符分词器，以文本中的空白符作为词语间隔
3. HanLPTokenizer这是一个实词分词器，也就是只保留实词，它是分类模块的默认分词器



如果需要修改分类模块的分词器，需要在加载文本数据之前，设置分词器，示例代码如下：

|  |
| --- |
| IDataSet trainingCorpus = **new** FileDataSet(). // FileDataSet省内存，可加载大规模数据集  setTokenizer(**new** HanLPTokenizer()). // 支持不同的ITokenizer，详见源码中的文档  load(***CORPUS\_FOLDER***, "UTF-8", 0.9); // 前90%作为训练集  IClassifier classifier = **new** NaiveBayesClassifier();  classifier.train(trainingCorpus); |

还需要注意的是，这三种分词器其实都不太适合做情感分类的分词器，如果想用标准分词器作为情感分类的分词器，则用户需要自定义一个分词器类，该类需要实现ITokenizer接口并封装标准分词器。（如果直接调用setTokenizer方法传递下图HanLP分词器包中的分词器实例，则会报错）



### 文本分类主要模块和方法调用关系



### 新闻文本分类示例

这里HanLP使用的是搜狗的新闻文本分类语料迷你版。涉及5个类目，各类目下文本数量分布如下：

|  |  |
| --- | --- |
| 类目名称 | 文本数量 |
| 健康 | 1000 |
| 教育 | 1000 |
| 军事 | 1000 |
| 汽车 | 1000 |
| 体育 | 1000 |

运行示例代码输出：

|  |
| --- |
| >python 中文新闻分类.py  《C罗获2018环球足球奖最佳球员 德尚荣膺最佳教练》 属于分类 【体育】  《英国造航母耗时8年仍未服役 被中国速度远远甩在身后》 属于分类 【军事】  《 研究生考录模式亟待进一步专业化》 属于分类 【教育】  《如果真想用食物解压,建议可以食用燕麦》 属于分类 【健康】  《通用及其部分竞争对手目前正在考虑解决库存问题》 属于分类 【汽车】 |

示例代码：

|  |
| --- |
| # -\*- coding:utf-8 -\*-  # Author：hankcs  # Date: 2018-05-23 17:26  import os  from pyhanlp import SafeJClass  from test\_utility import ensure\_data  NaiveBayesClassifier = SafeJClass('com.hankcs.hanlp.classification.classifiers.NaiveBayesClassifier')  IOUtil = SafeJClass('com.hankcs.hanlp.corpus.io.IOUtil')  sogou\_corpus\_path = ensure\_data('搜狗文本分类语料库迷你版',  'http://file.hankcs.com/corpus/sogou-text-classification-corpus-mini.zip')  def train\_or\_load\_classifier():  model\_path = sogou\_corpus\_path + '.ser'  if os.path.isfile(model\_path):  return NaiveBayesClassifier(IOUtil.readObjectFrom(model\_path))  classifier = NaiveBayesClassifier()  classifier.train(sogou\_corpus\_path)  model = classifier.getModel()  IOUtil.saveObjectTo(model, model\_path)  return NaiveBayesClassifier(model)  def predict(classifier, text):  print("《%16s》\t属于分类\t【%s】" % (text, classifier.classify(text)))  # 如需获取离散型随机变量的分布，请使用predict接口  # print("《%16s》\t属于分类\t【%s】" % (text, classifier.predict(text)))  if \_\_name\_\_ == '\_\_main\_\_':  classifier = train\_or\_load\_classifier()  predict(classifier, "C罗获2018环球足球奖最佳球员 德尚荣膺最佳教练")  predict(classifier, "英国造航母耗时8年仍未服役 被中国速度远远甩在身后")  predict(classifier, "研究生考录模式亟待进一步专业化")  predict(classifier, "如果真想用食物解压,建议可以食用燕麦")  predict(classifier, "通用及其部分竞争对手目前正在考虑解决库存问题") |

### 酒店评论情感分类示例

运行示例代码输出：

|  |
| --- |
| >python 中文情感分类.py  模式:训练集  文本编码:UTF-8  根目录:C:/ProgramData/Anaconda3/lib/site-packages/pyhanlp/static\data\test\ChnSentiCorp情感分析酒店评论  加载中...  [正面]...100.00% 2000 篇文档  [负面]...100.00% 2000 篇文档  耗时 5854 ms 加载了 2 个类目,共 4000 篇文档  原始数据集大小:4000  使用卡方检测选择特征中...耗时 28 ms,选中特征数:5834 / 15040 = 38.79%  贝叶斯统计结束  《前台客房服务态度非常好！早餐很丰富，房价很干净。再接再厉！》 情感极性是 【正面】  《结果大失所望，灯光昏暗，空间极其狭小，床垫质量恶劣，房间还伴着一股霉味。》 情感极性是 【负面】  《可利用文本分类实现情感分析，效果不是不行》 情感极性是 【正面】 |

示例代码：

|  |
| --- |
| # -\*- coding:utf-8 -\*-  # Author: hankcs  # Date: 2019-01-07 13:53  from pyhanlp import \*  from test\_utility import ensure\_data  IClassifier = JClass('com.hankcs.hanlp.classification.classifiers.IClassifier')  NaiveBayesClassifier = JClass('com.hankcs.hanlp.classification.classifiers.NaiveBayesClassifier')  # 中文情感挖掘语料-ChnSentiCorp 谭松波  chn\_senti\_corp = ensure\_data("ChnSentiCorp情感分析酒店评论", "http://file.hankcs.com/corpus/ChnSentiCorp.zip")  def predict(classifier, text):  print("《%s》 情感极性是 【%s】" % (text, classifier.classify(text)))  if \_\_name\_\_ == '\_\_main\_\_':  classifier = NaiveBayesClassifier()  # 创建分类器，更高级的功能请参考IClassifier的接口定义  classifier.train(chn\_senti\_corp)  # 训练后的模型支持持久化，下次就不必训练了  predict(classifier, "前台客房服务态度非常好！早餐很丰富，房价很干净。再接再厉！")  predict(classifier, "结果大失所望，灯光昏暗，空间极其狭小，床垫质量恶劣，房间还伴着一股霉味。")  predict(classifier, "可利用文本分类实现情感分析，效果不是不行") |

### 线程安全性

文本分类系统内部实现没有任何线程锁，但任何预测接口都是线程安全的。

### 性能指标

版本库中有一个在搜狗语料库上的测试com.hanks.demo.DemoTextClassificationFMeasure，含有完整的参数，可自运行评估。

### PyHanLP卡方特征选择有问题

相同的训练语料，相同的停用词表，用Java版训练时特征选择后的特征词数量为2万，用PyHanLP特征选择后的特征词数量3000。

|  |
| --- |
| Java版 使用卡方检测选择特征中...耗时 76 ms,选中特征数:20402 / 46598 = 43.78%  Python版 使用卡方检测选择特征中...耗时 60 ms,选中特征数:3530 / 46483 = 7.59% |

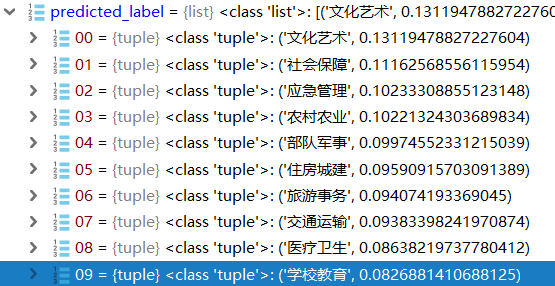
我们可以知道Python版输出了很多概率值相同的类目原因。在计算文本属于某个类目的概率时，首先计算了一个类目的先验概率即1/类目数量，而由于句子中不含有任何模型词表里的词，于是句子属于某个类目的概率最后就等于1/类目数量。因此文本属于任何类目的概率都是1/类目数量。这也就是为什么Python版代码通过设定预测概率方差阈值可以提高准确率的原因。

看下边的例子

[1] 化验指标一变化患者就六神无主，看医生怎么讲解

核心词：患者 看医生

这里“患者”和“看医生”两个词都没在“艺术”类训练语料中出现，但是预测概率最大的反倒是“艺术”，同时注意到基本上预测到每个类目的概率值都是0.1左右，这个0.1很显然是类目的先验概率，也就是说“患者”和“看医生”都没有在模型特征词表中出现，所以根本就没有词语的似然概率值，而且HanLP代码没有做平滑处理，即使做了也是所有类目的概率值全相等。



HanLP中概率计算的相关代码如下

|  |
| --- |
| @Override  public double[] categorize(Document document) throws IllegalArgumentException, IllegalStateException  {  Integer category;  Integer feature;  Integer occurrences;  Double logprob;  double[] predictionScores = new double[model.catalog.length];  for (Map.Entry<Integer, Double> entry1 : model.logPriors.entrySet())  {  category = entry1.getKey();  logprob = entry1.getValue(); //用类目的对数似然初始化概率  //对文档中的每个特征  for (Map.Entry<Integer, int[]> entry2 : document.tfMap.entrySet())  {  feature = entry2.getKey();  if (!model.logLikelihoods.containsKey(feature))  {  continue; //如果在模型中找不到就跳过了  }  occurrences = entry2.getValue()[0]; //获取其在文档中的频次  **logprob += occurrences \* model.logLikelihoods.get(feature).get(category); //将对数似然乘上频次**  }  predictionScores[category] = logprob;  }  if (configProbabilityEnabled)  **MathUtility.normalizeExp(predictionScores);**  return predictionScores;  } |

这里就是用概率值取对数计算的文本属于类目的概率，之后又对计算后的值做了指数得到0-1之间的概率值。

### 将取对数后的预测类目概率值序列去掉对数并归一化

朴素贝叶斯在计算一篇文本属于某个类目的概率时，由于概率值相乘会造成小值溢出，因此利用对数函数的单调递增性质，对预测概率公式两端取对数，这样就将概率乘法变成概率值取对数后的加法，用朴素贝叶斯预测类目公式表示如下：

我们都知道对数函数在[0,1]区间上的值都为负值，为了表达概率值的意义，对上边等式两边做指数运算

为了防止溢出进行下边的操作

其中，表示预测为第i个类目的概率值，为类别总数。

最后做归一化操作：

HanLP中相关代码

|  |
| --- |
| /\*\*  \* 使用log-sum-exp技巧来归一化一组对数值  \*  \* @param predictionScores  \*/  public static void normalizeExp(Map<String, Double> predictionScores)  {  Set<Map.Entry<String, Double>> entrySet = predictionScores.entrySet();  double max = Double.NEGATIVE\_INFINITY;  for (Map.Entry<String, Double> entry : entrySet)  {  max = Math.max(max, entry.getValue());  }  double sum = 0.0;  //通过减去最大值防止浮点数溢出  for (Map.Entry<String, Double> entry : entrySet)  {  Double value = Math.exp(entry.getValue() - max);  entry.setValue(value);  sum += value;  }  if (sum != 0.0)  {  for (Map.Entry<String, Double> entry : entrySet)  {  predictionScores.put(entry.getKey(), entry.getValue() / sum);  }  }  } |

# fastText库

## fastText简介

1. 误解1 fastText是文本分类的模型

fastText是Facebook Research<https://github.com/facebookresearch>在2016年开源的一个词向量及文本分类工具。在模型架构上跟word2vec非常相似，毕竟作者都是**Tomas Mikolov**。其实从另一种角度理解，fastText算法是word2vec的一种衍生模型。

1. 误解2 fastText无法在windows平台运行

fastText在github <https://github.com/facebookresearch/fastText/tree/master/python>上提供了绑定了fastText库的Python接口及使用示例代码。注意使用它的平台上应该安装gcc-4.8以上或clang-3.3以上的编译器，因为fastText需要c++ 11的支持。这里利用pybind11来创建绑定了C++代码的Python代码。

基本函数与相关词检索

# Numpy库

## loadtxt

如果数据类似于一个表格形式，则可以直接通过loadtxt方法加载为numpy矩阵形式。以下为loadtxt的简单用法。

|  |
| --- |
| input\_file\_name = '../data/阅读小说与性别关系.txt'  data = np.loadtxt(input\_file\_name, # 文件名  delimiter='\t', # 属性值之间的分隔符  dtype=str, # 各字段类型  skiprows = 1,  comments= None, # 注释开始的字符标识（注意一般设置为None，即不用）  usecols=(0,1), # 需要加载的列，列号索引从0开始  encoding='utf-8') # 文件编码  print(data.shape)  print(data[:2,:]) |

## mean

样本均值的计算公式为：

## var

总体方差计算公式为：

样本方差计算公式为：

注意：numpy中的var方法计算的是总体方差。

# Keras库

**张量（tensor）**的概念。0阶张量即标量，1阶张量即向量，2阶张量即矩阵，3阶张量即立方体，4阶张量即多个立方体。维度概念的说明

|  |
| --- |
| In [6]: import numpy as np  In [7]: a = np.array([[1,2],  ...: [3,4]])  In [8]: np.sum(a, axis=0)  Out[8]: array([4, 6])  In [9]: np.sum(a, axis=1)  Out[9]: array([3, 7]) |

**数据格式（Data Format）**的概念，它是指对训练或测试数据的表达方法。例如100张RGC三通道彩色图片高度为16，宽读为32，在Keras中表示为(100, 3, 16, 32)，但是在TensorFlow中表示为(100, 16, 32, 3)，Keras默认的数据组织方式在~/.keras/keras.json中定义，可以查看此文件的image\_data\_format项，也可以通过K.image\_data\_format()函数返回得到数据组织方式。

**函数式模型**（Functional Model），在Keras0.x中有两种模型分别为Sequential序列化模型也就是层与层之间只有相邻关系，没有跨层连接。Graph图模型，层与层之间想怎么连都可以。Keras1和Keras2中，不再支持Graph模型，但是增加了“functional model API”，在使用时利用的是“函数式编程”的风格，因此称之为函数式模型。

**批数据（batch）**Keras的模块中经常会出现batch\_size，指的是Mini-batch的大小。需要注意的是Keras中用的优化器SGD是随机梯度下将方法，不代表是一个样本就更新一回模型参数，SGD仍然是基于Mini-batch的。Mini-batch Gradient Descent小批量梯度下降算法，这种方法把数据分为若干个批次，按批来更新参数，这样，一个批中的一组数据共同决定了本次梯度的方向，下降起来就不容易跑偏，减少了随机性。另一方面，因为批的样本数与整个数据集相比小了很多，计算量也减少了很多。

**训练周期数(epochs)**指的是训练过程中整个数据集将被循环训练多少次。

Keras的核心数据结构是“**模型**”，模型是一种组织网络层的方式。Keras中主要的模型是Sequential模型。Sequential是一系列网络层按顺序构成的栈。一般首先通过实例化一个Sequential栈。然后通过栈的add(网络层)方法添加网络层。常见的核心网络层有全连接层Dense，丢弃层Dropout，激活层activation（注意一般通过网络层类的激活函数参数来设置，而不是单独再添加一个激活层）。完成模型的搭建后，需要使用栈的compile(…)方法来编译模型，该方法的常用参数有损失函数设置loss，优化器设置optimizer等。完成编译后，调用栈的fit方法来训练模型，该方法可设置的参数有在整个训练集上执行训练的周期数epochs，每个周期都会得到一个模型，最终选择一个最优的模型。小批量随机梯度下降训练过程中每个批次抽取的数据量batch\_size。最后，通过栈的evaluate方法来评估模型在测试集上的准确率。

Sequential对象的方法

|  |  |  |
| --- | --- | --- |
| 方法名 | 用途 | 示例 |
| **model.summary()** | 打印模型概况 |  |
| **model.get\_config()** | 返回模型配置信息的字典。模型可以从配置信息重构回去，其中config是一个Json格式的字典对象 | config = model.get\_config()  model = Sequential.from\_config(config) |
| model.get\_layer() | 依据层名或下标获得层对象 |  |
| **model.get\_weights()** | 返回模型权重张量的列表，类型为Numpy array |  |
| model.set\_weights() | 表示从Numpy array里将权重载入给定模型，要求数组具有与model.get\_weights()相同的形状 |  |
| model.to\_json() | 表示返回代表模型的Json字符串，仅包含网络结构，不包含权值。用户可以从Json字符串中重构原模型 | json\_str = model.to\_json()  model = model\_from\_json(json\_string) |
| model.save\_weights(filepath) | 表示将模型权重保存到指定路径，文件类型是HDF5 |  |
| model.load\_weights(filepath, by\_name=False) | 表示从HDF5文件中加载权重到当前模型中，默认情况下模型的结构保持不变。如果想将权重载入不同的模型（有些层相同）中，则设置by\_name=True，只有名字匹配的层才会载入权重 |  |

compile方法的参数

|  |  |  |
| --- | --- | --- |
| 参数名 | 用途 | 示例 |
| optimizer | 优化器名或优化器对象，设置网络训练时的优化方法 | 可用的优化器详见官方文档 |
| loss | 损失函数名或损失函数对象，设置网络训练时的损失函数 | 可用的损失函数详见官方文档 |
| metrics | 网络训练时评估网络性能的指标。 | metrics=[‘accuracy’] |