# 1 Python自然语言处理实践笔记

## 中文分词技术

### 中文简介

三种方法：

1）规则分词：通过人工设立词库，按照一定方式进行匹配切分，简单高效。缺点：对新词很难处理。分为：正向最大匹配法、逆向最大匹配法、双向最大匹配法。

2）统计分词：基于统计机器学习方法。缺点：过于依赖语料的质量。

3）混合分词（规则+统计）

### 1.1.2 统计分词

1 语言模型

用概率论的专业术语描述语言模型：为长度为m的字符串确定其概率分布：

*P*(w1,w2,w3...,wm)

*P*(w1,w2,w3...,wm) = *P*(w1)|P(w2|w1)*P*(w3|w1,w2).....*P*(wm|w1,w2,w3...wm-1)

## 2.1 jieba中文处理

安装：pip install jieba

### 1 基本分词函数与用法

jieba.cut 以及 jieba.cut\_for\_search 返回的结构都是一个可迭代的 generator，可以使用 for 循环来获得分词后得到的每一个词语(unicode)

1 jieba.cut 方法接受三个输入参数:

需要分词的字符串

cut\_all 参数用来控制**是否采用全模式**

HMM 参数用来控制**是否使用 HMM 模型**

2 jieba.cut\_for\_search 方法接受两个参数

需要分词的字符串

是否使用 HMM 模型。

# encoding=utf-8

import jieba

seg\_list = jieba.cut("我在学习自然语言处理", cut\_all=True)

print seg\_list

print("Full Mode: " + "/ ".join(seg\_list)) # 全模式

seg\_list = jieba.cut("我在学习自然语言处理", cut\_all=False)

print("Default Mode: " + "/ ".join(seg\_list)) # 精确模式 -- 默认

seg\_list = jieba.cut("他毕业于上海交通大学，在百度深度学习研究院进行研究") # 默认是精确模式

print(", ".join(seg\_list))

seg\_list = jieba.cut\_for\_search("小明硕士毕业于中国科学院计算所，后在哈佛大学深造") # 搜索引擎模式

print(", ".join(seg\_list))

# 2 nlp必知必会

## 2.1 基础知识

### 1 xgboost和xgboost区别

机器学习算法中GBDT和XGBOOST的区别有哪些？

**基分类器的选择：**传统GBDT以CART作为基分类器，XGBoost还支持线性分类器，这个时候XGBoost相当于带L1和L2正则化项的逻辑斯蒂回归（分类问题）或者线性回归（回归问题）。

**二阶泰勒展开：**传统GBDT在优化时只用到一阶导数信息，XGBoost则对代价函数进行了二阶泰勒展开，同时用到了一阶和二阶导数。顺便提一下，XGBoost工具支持自定义损失函数，只要函数可一阶和二阶求导。

**方差-方差权衡**：XGBoost在目标函数里加入了正则项，用于控制模型的复杂度。正则项里包含了树的叶子节点个数、每个叶子节点上输出分数的L2模的平方和。从Bias-variance tradeoff角度来讲，正则项降低了模型的variance，使学习出来的模型更加简单，防止过拟合，这也是XGBoost优于传统GBDT的一个特性。

**Shrinkage（缩减）：**相当于学习速率（xgboost中的）。XGBoost在进行完一次迭代后，会将叶子节点的权重乘上该系数，主要是为了削弱每棵树的影响，让后面有更大的学习空间。实际应用中，一般把eta设置得小一点，然后迭代次数设置得大一点。（补充：传统GBDT的实现也有学习速率），而对于xgboost来说，学习率是每次减去eta

列抽样（column subsampling）：XGBoost借鉴了随机森林的做法，支持列抽样，不仅能降低过拟合，还能减少计算，这也是XGBoost异于传统GBDT的一个特性。

**缺失值处理：**XGBoost考虑了训练数据为稀疏值的情况，可以为缺失值或者指定的值指定分支的默认方向，这能大大提升算法的效率，paper提到50倍。即对于特征的值有缺失的样本，XGBoost可以自动学习出它的分裂方向。

XGBoost工具支持并行：Boosting不是一种串行的结构吗?怎么并行的？注意XGBoost的并行不是tree粒度的并行，XGBoost也是一次迭代完才能进行下一次迭代的（第次迭代的损失函数里包含了前面次迭代的预测值）。XGBoost的并行是在特征粒度上的。我们知道，决策树的学习最耗时的一个步骤就是对特征的值进行排序（因为要确定最佳分割点），XGBoost在训练之前，预先对数据进行了排序，然后保存为block(块)结构，后面的迭代中重复地使用这个结构，大大减小计算量。这个block结构也使得并行成为了可能，在进行节点的分裂时，需要计算每个特征的增益，最终选增益最大的那个特征去做分裂，那么各个特征的增益计算就可以开多线程进行。

线程缓冲区存储：按照特征列方式存储能优化寻找最佳的分割点，但是当以行计算梯度数据时会导致内存的不连续访问，严重时会导致cache miss，降低算法效率。paper中提到，可先将数据收集到线程内部的buffer（缓冲区），主要是结合多线程、数据压缩、分片的方法，然后再计算，提高算法的效率。

可并行的近似直方图算法：树节点在进行分裂时，我们需要计算每个特征的每个分割点对应的增益，即用贪心法枚举所有可能的分割点。当数据无法一次载入内存或者在分布式情况下，贪心算法效率就会变得很低，所以xgboost还提出了一种可并行的近似直方图算法，用于高效地生成候选的分割点。大致的思想是根据百分位法列举几个可能成为分割点的候选者，然后从候选者中根据上面求分割点的公式计算找出最佳的分割点。

参考资料：<https://blog.csdn.net/chengfulukou/article/details/76906710>