概念

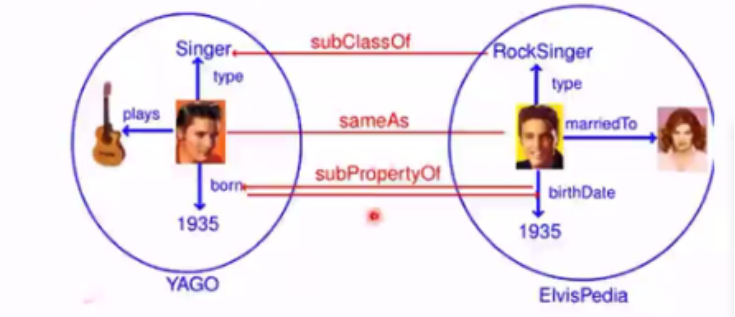
将来自多个来源的关于同一个实体或概念的描述信息融合起来

等价实例

等价类/子类

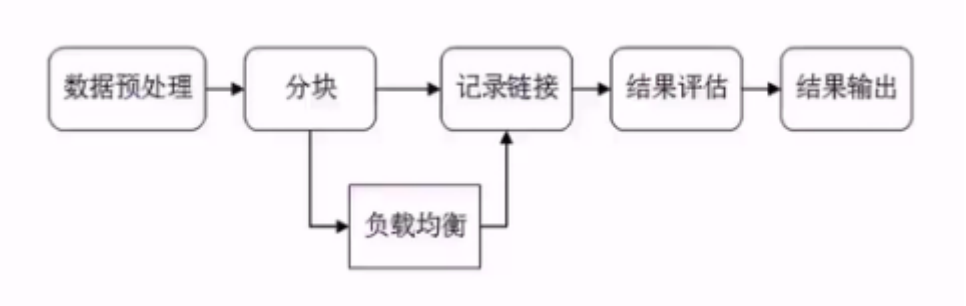
等价属性/子属性

同一个人，不同的知识库：



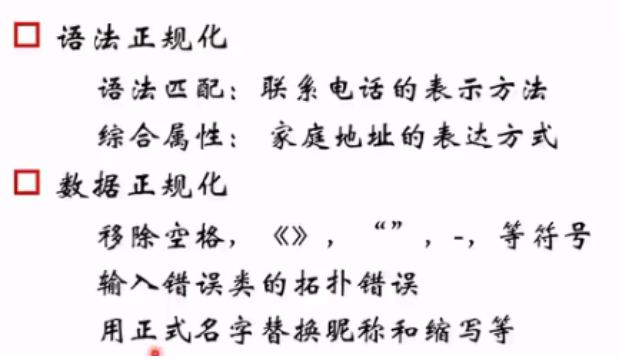
基本技术流程

两步：本体对齐、实体匹配



数据预处理

数据仓库ETL...



记录链接

假设两个实体的记录x和y，x和y在第i个属性上的值是x_i,y_i，那么通过如下两步进行记录链接：

1. 属性相似度：
   综合单个属性相似度得到属性相似度向量
   $[sim(x_1, y_1), sim(x_2, y_2), \ldots, sim(x_N, y_N)]$
2. 实体相似度：
   根据属性相似度向量得到一个实体的相似度

属性相似度的计算

编辑距离

Levenstein，Wagner and Fisher，Edit Distance with Affine Gaps

集合相似度计算

Jaccard系数，Dice

基于向量的相似度计算

Cosine相似度，TFIDF相似度

……

编辑距离

Levenshtein Distance

Levenshtein distance（最小编辑距离），目的是用最少的编辑操作将一个字符串转成另一个，如下：

'Lvensshtain' —插入'e'→ 'Levensshtain'

'Levensshtain' —删除's'→ 'Levenshtain'

'Levenshtain' —替换'a'→'e'→ 'Levenshtein'

上述将'Lvensshtain'转换成'Levenshtein'，总共的操作3次，编辑距离也就是3。

Levenshtein distance是典型的动态规划问题，可以通过动态规划算法计算，如下：

$$\begin{cases} D(0,0) = 0 \\ D(i,0) = D(i-1,0) + 1 & 1 < i \leqq N \\ D(0,j) = D(0,j-1) + 1 & 1 < j \leqq M \end{cases}$$

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + 1 \end{cases}$$

其中，+1表示的是插入，删除和替换的代价。

Wagner and Fisher Distance

Wagner and Fisher distance是Levenshtein 的一个扩展，将这个模型中编辑操作的代价赋予了不同的权重，如下：

$$\begin{cases} D(0,0) = 0 \\ D(i,0) = D(i-1,0) + del[x(i)] & 1 < i \leqq N \\ D(0,j) = D(0,j-1) + del[y(j)] & 1 < j \leqq M \end{cases}$$

$$D(i,j) = \min \begin{cases} D(i-1,j) + del[x(i)] \\ D(i,j-1) + ins[y[j]] \\ D(i-1,j-1) + sub[x(i),y(j)] \end{cases}$$

其中，*del* 和 *ins* 以及 *sub* 分别是删除和插入以及替换的代价

Edit Distance with affine gaps

Edit Distance with affine gaps，在上面两种算法的基础上，引入了gap的概念，将上述的插入，删除和替换操作用用gap opening和gap extension代替，编辑操作的代价也就表示为：

$$Cost(g) = s + e * l$$

其中，s是open gap的代价，e是extend gap的代价，l是gap的长度。

例：

结合前面'Lvensshtain'转换的例子：

$$L\ \boxed{\varepsilon}\ v\ e\ n\ s\ s\ h\ t\ a\ \boxed{\varepsilon}\ i\ n$$

$$L\ e\ v\ e\ n\ s\ \boxed{\varepsilon}\ h\ t\ \boxed{\varepsilon}\ e\ i\ n$$

其中，$\boxed{\varepsilon}$代表一个gap，结合上述代价公式，若设置s = 2，e = 1上述编辑操作代价为$(2 + 1 * 1) * 4 = 8$。

\\=12

集合相似度计算

Dice系数

Dice系数用于度量两个集合的相似性，因为可以把字符串理解为一种集合，因此Dice距离也会用于度量字符串的相似性，Dice系数定义如下：

$$sim_{Dice}(s,t)=\frac{2|S\cap T|}{|S|+|T|}$$

以‘Lvensshtain’和‘Levenshtein’为例，两者相似度为 2*9/ (11+11)= 0.82

Jaccard系数

Jaccard 系数适合处理短文本的相似度，定义如下：

$$sim_{Jaccard}(s,t)=\frac{|S\cap T|}{|S\cup T|}$$

可以看出与Dice系数的定义比较相似。

两种方法,将文本转换为集合,除了可以用符号分格单词外,还可以考虑用n-gram分割单词,用n-gram分割句子等来构建集合,计算相似度。

基于向量的相似度

TF-IDF

TF-IDF主要用来评估某个字或者某个词对一个文档的重要程度。

$$tf_{i,j}=\frac{n_{i,j}}{\sum_k n_{k,j}} \qquad idf_i=\log\frac{|D|}{1+|\{j:t_i\in d_j\}|}$$

$$sim_{TF-IDF}=tf_{i,j}\times idf_i$$

比如某个语料库中有5万篇文章，含有“健康”的有2万篇，现有一篇文章，共1000个词，‘健康’出现30次，则$sim_{TF\text{-}IDF}$ = 30/1000 * log(50000/(20000+1)) = 0.012

实体相似度的计算

1. 聚合

   加权平均，手动制定规则，分类器

2. 聚类

   层次聚类，相关性聚类

3. 表示学习

聚合

加权平均：对相似度得分向量的各个分量进行加权求和，得到最终的实体相似度

手动制定规则：给每一个相似度向量的分量设置一个阈值，超过该阈值则将两实体相连

- 加权平均：$w_1 * sim(x_1, y_1) + \ldots + w_N * sim(x_N, y_N)$
- 手动制定规则：$sim(x_1, y_1) > T_1$ and (or) …. $sim(x_i, y_i) > T_i$
- 分类器：逻辑回归，决策树，SVM和条件随机场等

聚类

- 层次聚类
- 相关性聚类
- Canopy + K-means

层次聚类

通过计算不同类别数据点之间的相似度对在不同的层次的数据进行划分,最终形成树状的聚类结构。

- 层次聚类

底层的原始数据可以通过相似度函数计算，类之间的相似度有如下三种算法：

SL(Single Linkage)算法

SL算法又称为最邻近算法 (nearest-neighbor)，是用两个类数据点中距离最近的两个数据点间的相似度作为这两个类的距离。

CL (Complete Linkage)算法

与SL不同的是取两个类中距离最远的两个点的相似度作为两个类的相似度。

AL (Average Linkage)算法

用两个类中所有点之间相似度的均值作为类间相似度。

举例：有左图的数据，用欧氏距离和SL进行层次聚类。

A	16.9
B	38.5
C	39.5
D	80.8
E	82
F	34.6
G	116.1

	A	B	C	D	E	F	G
A	0	21.60	22.60	63.90	65.10	17.70	99.20
B	21.60	0	1.00	42.30	43.50	3.90	77.60
C	22.60	1.00	0	41.30	42.50	4.90	76.60
D	63.90	42.30	41.30	0	1.20	46.20	35.30
E	65.10	43.50	42.50	1.20	0	47.40	34.10
F	17.70	3.90	4.90	46.20	47.40	0	81.50
G	99.20	77.60	76.60	35.30	34.10	81.50	0

B,C之间的欧式距离是：$\sqrt{(B-C)^2} = \sqrt{(38.5-39.5)^2}$

上页B,C之间距离最小为1.00，将B，C组合，再次计算距离

	A	(B,C)	D	E	F	G
A	0	22.10	63.90	65.10	17.70	99.20
(B,C)	22.10	0	41.80	43.00	4.40	77.10
D	63.90	41.80	0	1.20	46.20	35.30
E	65.10	43.00	1.20	0	47.40	34.10
F	17.70	4.40	46.20	47.40	0	81.50
G	99.20	77.10	35.30	34.10	81.50	0

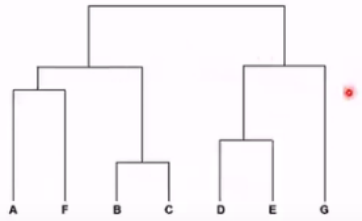
其中，单个数据与类之间的距离，如下：

$$D = \frac{\sqrt{(B-A)^2} + \sqrt{(C-A)^2}}{2} = \frac{21.6 + 22.6}{2}$$

最终得到如下两个类：

	(A,F,B,C)	(D,E,G)
(A,F,B,C)	0	60.59
(D,E,G)	60.59	0

前面的每一步的计算结果以树状图的形式展现出来就是层次聚类树。



相关性聚类

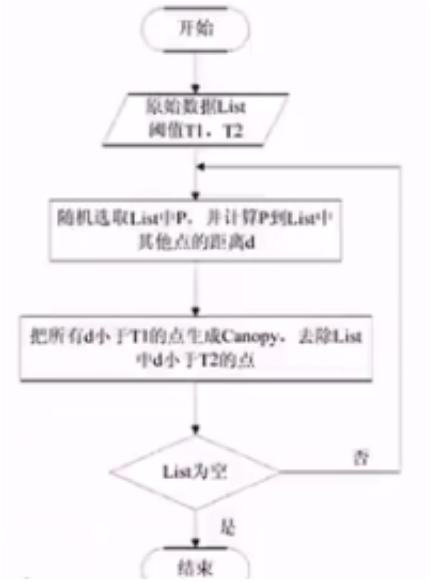
r_{xy}表示x,y被分配在同一类中，p_{xy}代表x,y是同一类的概率（x,y之间的相似度），$w^{+}_{xy}(=p_{xy})$和$w^{-}_{xy}(=1-p_{xy})$分别是切断x,y之间的边的代价和保留边的代价。相关性聚类的目标就是使用最小的代价找到一个聚类方案。

$$minimize \sum r_{xy} w^{-}_{xy} + (1 - r_{xy}) w^{+}_{xy}$$

是个NP-Hard问题，可用贪婪算法近似求解。

Canopy + K-means

与K-means不同，Canopy聚类最大的特点是不需要事先指定k值（即clustering的个数），因此具有很大的实际应用价值，经常将Canopy和K-means配合使用。



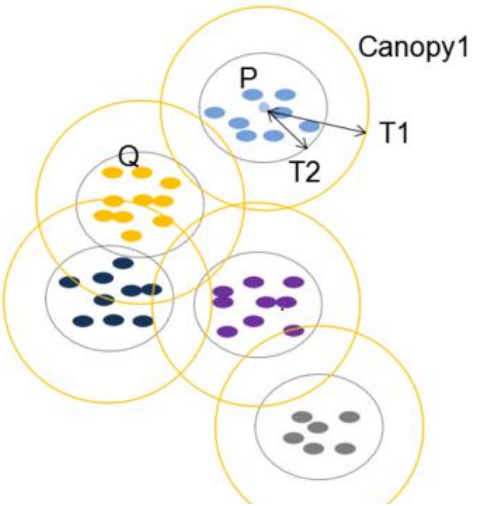
（1）从集合选出一个点P，将其做第一个类的中心，并将这个点从List中删除，称其为Canopy1。（后续产生第i个类成为Canopyi）；

（2）对剩下集合的所有点计算到点P的distance。

（3）将所有distance<T2的点都从集合List中删除，说明这些点离Canopy1已经足够近，避免重复加入到其他Canopy。

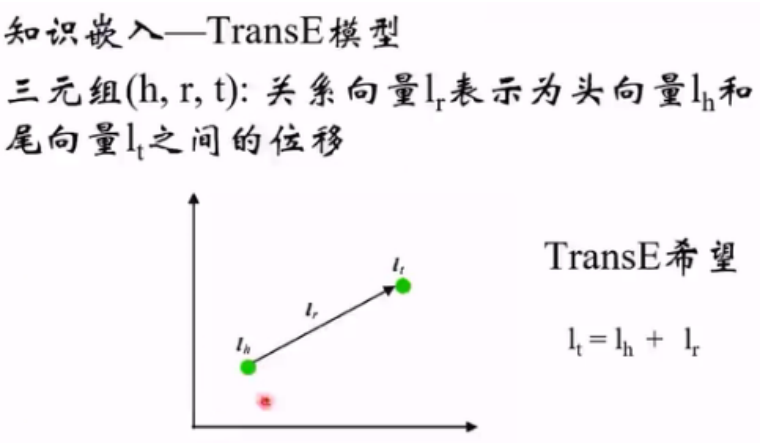
（4）将所有distance<T1的点都对到以P为中心的Canopy1中，若点i的distance>T1，则将其作为第i个类Canopyi；

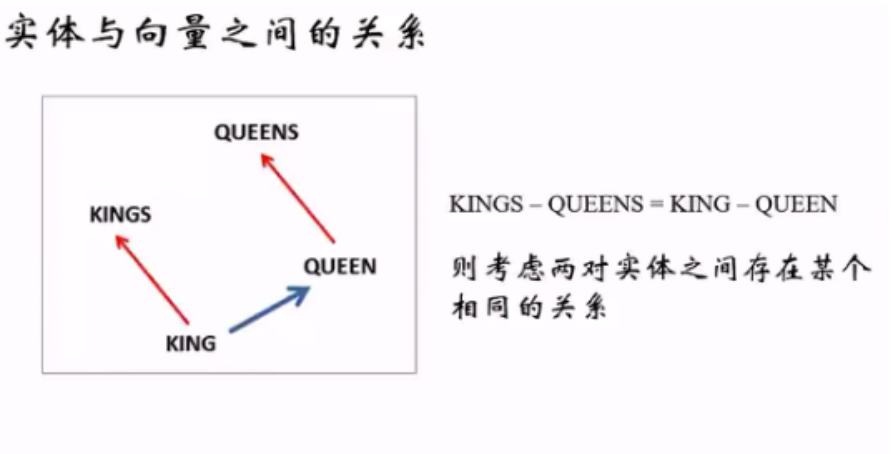
（5）对List重复步骤（1）~（4）知道List为空，则可以形成多个Canopy类。

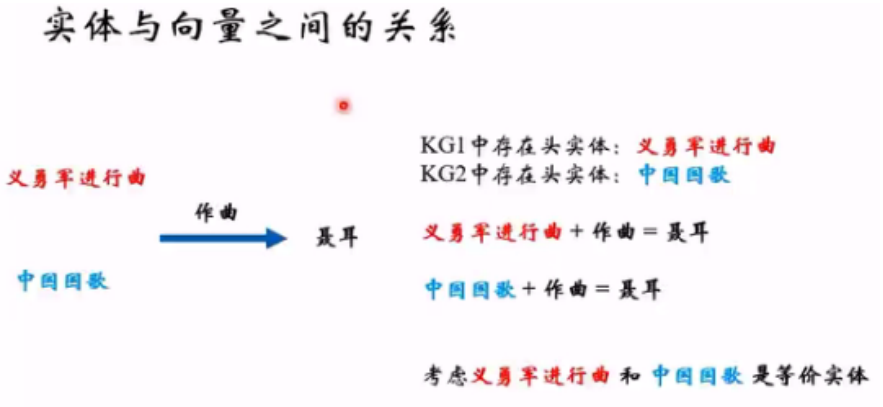


知识表示学习–知识嵌入

将知识图谱中的实体和关系都映射低维空间向量,直接用数学表达式来计算各个实体之间相似度。这类方法不依赖任何的文本信息,获取到的都是数据的深度特征







◆ 如何将两个知识图谱嵌入到同一个空间
- ◆ 桥梁：预链接实体对（训练数据）
- ◆ 方法：
  - ◆ 联合知识嵌入：将两个KG的三元组糅合在一起共同训练，并将预链接实体对视为具有SameAS关系的三元组，从而对两个KG的空间进行约束

  具体实现： Hao Zhu et al. Iterative Entity Alignment via Knowledge Embeddings, IJCAI 201
  - ◆ 双向监督训练：两个KG单独进行训练，使用预链接数据交替进行监督。

◆ 如何链接实体

KG向量训练达到稳定状态之后，对于KG1每一个没有找到链接的实体，在KG2中找到与之距离最近的实体向量进行链接，距离计算方法可采用任何向量之间的距离计算，例如欧式距离或Cosine距离。

分块

从给定的知识库中的所有实体对中,选出潜在匹配的记录对作为候选项,并将候选项的大小尽可能的缩小。

原因很简单，因为数据太多了，，我们不可能去一一连接。

1. 基于Hash函数的分块

对于记录x，有hash(x)=h_i,则x映射到与关键字h_i绑定的块C_i上。

常见的hash函数：

—字符串的前n个字

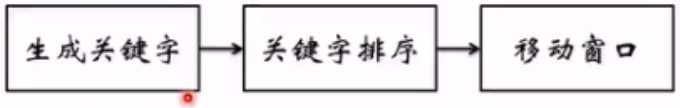
—n-grams

—结合多个简单的hash函数等

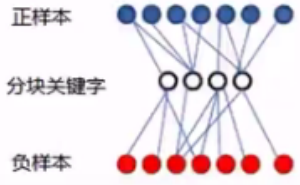
2. 邻近分块

☐ Canopy聚类

☐ 排序邻居算法



☐ Red-Blue Set Cover

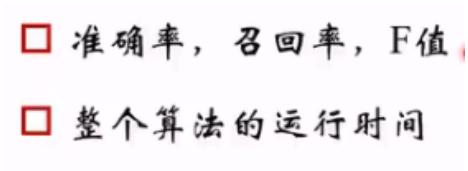


负载均衡

保证所有块中的实体数目相当,从而保证分块对性能的提升程度。

最简单的方法是多次Map-Reduce操作。

结果评估



典型知识融合工具简介

本体对齐——Falcon-AO

实体匹配——Dedupe、实体匹配——Limes、实体匹配——Silk