SEU 知识融合b

2019年11月28日 14:30

- **实例匹配问题**

  首先我们明确实例匹配问题不是一个新问题

  - Instance Matching 实例匹配
    - 知识工程领域（知识图谱、语义web等）
  - Entity Resolution/Coreference 实体解析
    - 自然语言处理领域、数据库领域
  - Record Linkage 记录链接
    - 数据库领域
  - Duplicate Detection重复检测
    - 数据库领域
  - Name Disambiguation 名字消解
    - 数字图书馆 社交网络

  - 意义相近，不同领域的称谓
    - 实例instance 知识工程领域
    - 实体entity 自然语言处理
    - 记录 record 数据库
    - 对象 object 知识工程领域

  - **问题分析-术语解释**
    - 指代

      对真实世界中的实例的某种描述方式，包括ID、名字、绰号、文本、图、URI等
    - 匹配

      给定任意两个指代信息（可以想尽也可以不同），判断其描述的是否是同一实例
    - 实例匹配

      这里指对知识图谱的实例进行匹配的过程

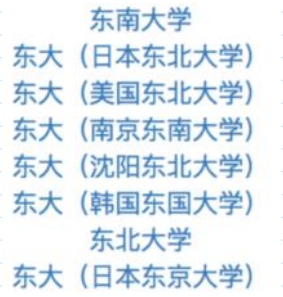
  - **问题分析-挑战**

    E.G. NewYork NewYorkCity NY BigApple

    特点：异名同意

    E.G.“东大”的不同指代

    特点：同名异译

    i. **多义现象（polysemy）**

       多个实体有相近或者相同的指代或呈现

    ii. **同意现象（synonym）**

       一个实体有多个不同的指代或呈现形式

    iii. **匹配效果和匹配性能**

       1）真实知识图谱的实例规模大

       2）匹配效果和匹配性能如何平衡

       3）时间复杂度和空间复杂度都很重要

       在知识图谱的实例匹配中，一般而言，有限保证匹配性能，在性能可接受的前提下去优化匹配效果

同义问题——名字写法不同，实际为同一作者

问题场景	场景举例
名字缩写	J Zhao, Jianyu Zhao
编码兼容	Jonas Bjork, Jonas Björk
识别错误	M Ackerman, M Ackernnan
中间名缺失	Wil van der Aalst, Wil Aalsty

多义问题——名字写法相同，实际为不同作者

问题场景	场景举例
同音字	Chen Chen对应“陈辰”、“陈晨”和“陈琛”
常用名	Paul Erdos， John Smith，Michael Jordan

- 启发：多的信息有助于实力匹配的解决
- 现实情况：我们能获得的信息总是有限的，因此只能尽可能的有效利用现有的信息

- **问题分析-与传统问题的比较**
  - 优势：更丰富的语义信息
    - 丰富的语义信息
    - 丰富的实例间联系
    - 可以使用推理技术
  - 劣势：更多的异构性
    - 任何人都可以构造本体
    - 数据规模大

- **解决思路-思路分析**
  - 基于推理的匹配方法
    - 利用本体语义构造推理规则
    - 寻找更多的匹配实例或者过滤错误的匹配
      - 优点：易于实现
      - 缺点：真实知识图谱中可能用于实例匹配的语义信息非常有限；
        大规模推理的性能

  - **直观的方法-基于实例对相似度的匹配**
    - 设计相似度匹配器
    - 时间复杂度O（n^2）
      - 优点：易于实现
      - 缺点：复杂度太高，无法处理大规模实例匹配

  - **基于机器学习的匹配方法**
    - 将匹配问题转化为二分类问题：匹配和不匹配
    - 构造分类特征
    - 人工构造训练集
    - 对分类模型进行训练
      - 优点：易于实现
      - 缺点：特征选取，学习策略等对结界影响大：
        需要大量的训练集，工作量大；受限于领域

  - **基于分块的大规模实例匹配**
    - 核心思想：避免做pair-wise comparision
    - 将复杂度高的大规模匹配计算转化为众多复杂度低的小闺蜜匹配计算
    - 设计分块规则
    - 匹配计算只能在块内进行
    - 复杂度：O(k*n^2) k个块 n远小于N
      - 优点：大规模实例匹配的高效解决方案
      - 不足：存在匹配遗漏问题
    - **候选集的确定**
      - **分块法**

- ◇ 用聚类或规则将可能匹配的实例作为一个块
- ◇ 将所有的实例间的比较转化为块中实例的匹配
- ◇ 典型的“分治法”思想的应用

◆ **无论使用什么属性分块技术，都将面临两个问题**

- ◇ 匹配效果：分块越细，造成的分块冗余越多，但未命中的匹配也越少，即匹配效果会更好
- ◇ 匹配性能：分块越细，造成的不必要匹配计算越多，降低了匹配的性能

所以，很多基于属性分块的方法都力图在匹配效果和匹配性能上达到平衡

◆ **分块法总结**

- ◇ 效率：分块越多，效率越高
- ◇ 效果：分块越多，召回率越低
- ◇ 平衡：考虑重叠的块来弥补召回率问题，但要保证效率第一

◆ 性能提升

- ◇ 单机的多线程
  - ▸ 优势：对原有方法的改动小
  - ▸ 受CPU核数的限制
- ◇ 多机的分布式计算
  - ▸ 效率：分块越多 效率越高
  - ▸ 分块过多 匹配效果会下降

◆ **倒排索引法**

- ◇ 把实例信息转化为向量、文档等适合索引的形式
- ◇ 构造相应的倒排索引
- ◇ 根据索引确定候选集
  - ▸ 候选规则：长向量有>=2个共有词；短向量有1个共有词
  - ▸ 根据倒排索引确定匹配候选对

◆ **五种实例信息**

- ◇ URI：本体中唯一标识符
- ◇ Meta：语义信息，即实例所属概念和所拥有的属性
- ◇ Name：名称和相关描述
- ◇ Property Values：属性取值
- ◇ Neighbors：与其他实例的联系信息

◆ **自底向上的本体划分**

- ◇ 每个聚类内结点间的内聚度比较高
- ◇ 不同聚类内结点的耦合度较低
- ◇ RDF句子：RDF三元组的集合，保证匿名结点的完整性

◆ **匹配的局部性方法**

- ◇ 正锚点：两个点匹配，则其邻居也可能匹配，进行匹配
- ◇ 负锚点：两个点不匹配，则其邻居也很有可能不匹配，也跳过匹配

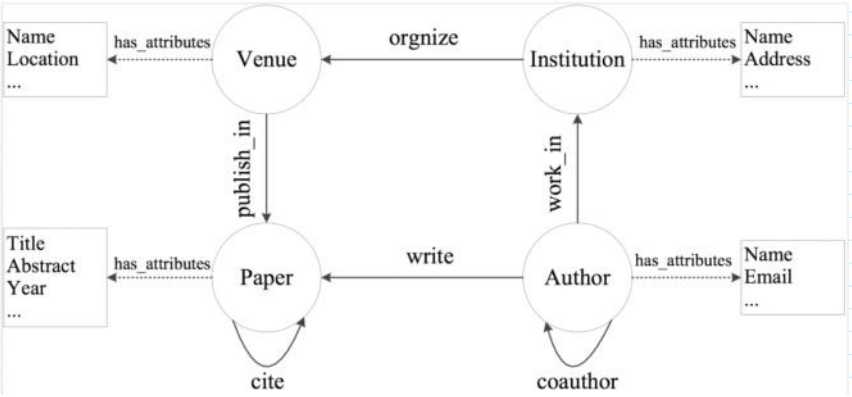
▪ **方法原理：利用本体中的语义信息**

- □ owl：sameAs：发现等价实例集合
- □ owl： differentFrom：避免错误的匹配
- □ owl：InverseFunctionalProperty：反函数属性语义保证了对于单个对象，其反函数类型的值唯一邮件地址（foaf：mbox）为一个反函数型属性，表明如果两个实例标识符拥有相同的邮件地址，则这两个标识符，指称相同的对象
- □ 函数基数：owl：cardinality or owl：maxCardinality用于发现匹配或者过滤匹配

▪ **方法分析**

- □ 无法发现隐式匹配
- □ 可用于推理的语义信息往往很有限
- □ 可作为其他方法的辅助

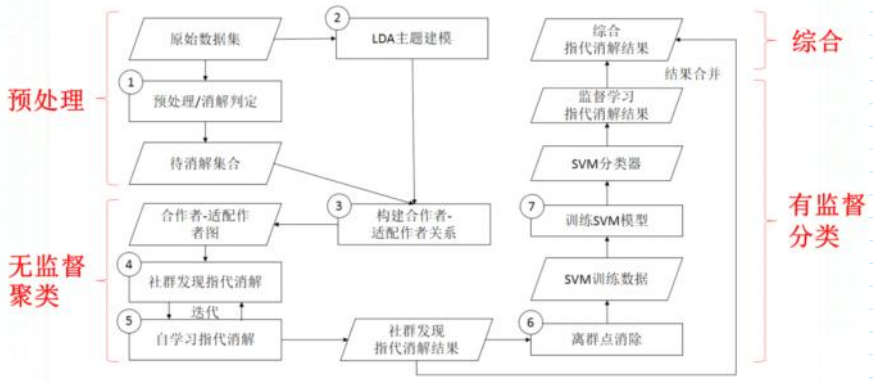
• **实例分析-学术知识图谱**



a. Schema局部，其他还包括学术活动、论文信息、学者族谱等；

b. Schema可根据引用扩展或剪裁

解决方案：基于半监督学习的大规模学术知识图谱消解技术



候选匹配集合的确定：名字消解判定规则

同义词适配类型

规则名	适配类型	规则描述
高频签名规则	D1	签名频数超过阈值T1的两个作者名
拼音规则	D2	汉语、粤语和韩语等语言的两个作者名

多义词适配类型

规则名	适配类型	规则描述
签名形式规则	D3	两个作者名其一的完全形式和签名形式相同
编辑距离规则	D4	两个作者姓名拼接串的编辑距离大于T2
中间名缺失规则	D5	一个作者无中间名，另一为签名形式
活跃年份规则	D6	活跃年份相似度小于T3的两个作者名

候选匹配集合的确定：名字消解判定规则

■D7型：满足适配必要条件，但不满足上述适配类型的两个作者名

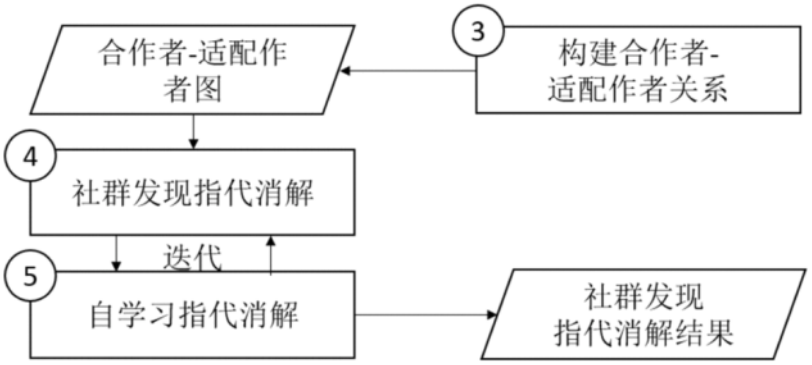
■不适配：不满足上述适配类型的两个作者名，不进行指代消解

■适配型优先级：

$$D3 = D4 = D5 = D6 > D2 > D1 > D7$$

对结构信息的利用：学术社群发现

基于社群发现的指代消解



- **实例分析-OpenKG的百科知识融合**

- 动机

目前OpenKG中已收录了76个中文数据集，目前已经融合几个百科类数据集Zhishi.me, CN-DBpedia, PKU-PIE和Belief Engine。

- 数据集

http://openkg.cn/dataset

- 链接结果

http://openkg.cn/dataset/l

名称	实体数
CN-DBpedia	16,546,273
Zhishi.me	10,337,505 (zhwiki, hudongbaike, baidubaike)
PKUBase	11,554,258
Belief Engine	1,423,828 (currently, hudongbaike)
XLore	10,856,042 (label only)

	CN-DBpedia	Zhishi.me	PKUBase	Belief Engine
CN-DBpedia	-	5,393,835	6,347,721	1,538,865
Zhishi.me	5,393,835	-	5,687,428	697,123
PKUBase	6,347,721	5,687,428	-	552,066
Belief Engine	1,538,865	697,123	552,066	-

- 思路设计

```
public void run() {
    /** 1.读本体、解析本体、初始化参数 **/
    init();
    parseOnt();
    /** 2.相似度计算 **/
    calculateSimilarity();
    /**进行实例消解**/
    ontDisambiguation();
    /**3.抽取匹配**/
    extractMatching();
    /**4.映射结果评估 **/
     evaluate();
    /**5.保存结果 **/
     outputMatch();
}
```

解析和预处理

相似度计算

消解

抽取匹配

评估

- 实现分析—相似度计算

```
/**相似度计算**/
private void calculateSimilarity() {
```

实现分析—相似度计算

```
/**相似度计算**/
private void calculateSimilarity() {
    /*基本描述文档相似度计算*/
    System.out.print("Calculating basic similarity ....");
    calBaseSimilarity();
    System.out.println("done!");
    /*层次相似度计算*/
    System.out.print("Calculating hierarchy similarity ....");
    calHierarchySimilarity();
    System.out.println("done!");
    /*非层次相似度计算*/
    System.out.print("Calculating normal similarity ....");
    calNomalSimilarity();
    System.out.println("done!");
    /*相似度传播计算*/

    /*合并相似度*/
    System.out.print("Combining similarity matrix....");
    combineSimlarityMatrix();
    System.out.println("done!");

    /*相似度传播*/
    System.out.print("Similarity propagation....");
    simPropagation();
    System.out.println("done!");

}
```

基本相似度

层次相似度

非层次相似度

相似度合并

相似度传播

实现分析—消解

```
/**对源本体中的实例进行消解处理**/
private void targetOntDisambiguation() {
    //源本体消解
    double[][] targetBSMatrix = new double[targetInsNum][targetInsNum];
    targetBSMatrix = getTextSim(targetInsNum, targetInsNum, targetInsBa

    double[][] targetHSMatrix = new double [targetInsNum][targetInsNum]
    double[][] targetFSMatrix = getTextSim(targetInsNum, targetInsNum,
    double[][] targetSSMatrix = getTextSim(targetInsNum, targetInsNum,
    for (int i=0;i<targetInsNum;i++)
        for (int j=0;j<targetInsNum;j++){
            targetHSMatrix[i][j]=0.6*targetFSMatrix[i][j]+0.6*targetSSM
            if (targetHSMatrix[i][j]>1.0) targetHSMatrix[i][j]=1.0;
        }

    double[][] targetNSMatrix = new double [targetInsNum][targetInsNum]
    double[][] targetPSMatrix = getTextSim(targetInsNum, targetInsNum,
    double[][] targetOSMatrix = getTextSim(targetInsNum, targetInsNum,
```

计算得到
实例相似度矩阵

实现分析—消解

```
for (int i = 0; i < targetInsNum; i++) {
    for (int j = 0; j <= i; j++) {
        if (tMatrix[i][j] > upboundThreshold) {
            //找到一个相似实例后记录对称结果两次，便于后面选择遍历方便
            MapRecord m = new MapRecord();
            m.sourceLabel = (String) targetInsName.get(i);
            m.targetLabel = (String) targetInsName.get(j);
            m.similarity = tMatrix[i][j];
            sameIns.add(m);
            m = new MapRecord();
            m.sourceLabel = (String) targetInsName.get(j);
            m.targetLabel = (String) targetInsName.get(i);
            m.similarity = tMatrix[i][j];
            sameIns.add(m);
            newSameFlag = true;
            System.out.println("DEBUG:" + m.sourceLabel + " "+ m.targe
            targetSMatrix[i][j] = 0.0; // 清空当前匹配位置，供下一个消解用
            targetSMatrix[j][i] = 0.0; // 清空当前匹配位置，供下一个消解用
        }
    }
}
```

遍历实例相似度
矩阵进行消解