第2章 ‖

相似度计算算法

相似度计算算法是实体识别中的核心基础算法，本章介绍基于字段(token)的相似度算法(Jaccard 相似度算法、基于 TF-IDF 相似度算法、基于 q-grams 的相似度算法)、基于编辑距离的相似度算法(Levenshtein 距离算法、Jaro 和 Jaro-Winkle 距离算法)、混合的相似度算法(扩展的 Jaccard 算法、Monge-Elkan 算法、Soft TF-IDF 算法)和其他的相似度算法。

2.1 基于字段的相似度算法

2.1.1 Jaccard 相似度算法

Jaccard 相似度[1]最早由 Paul Jaccard 提出，是一种最常见的评判相似程度的统计指标。Jaccard 相似度作用在两个集合上，其值为集合交与并的比值，见公式(2.1)。

$$\text{Jaccard}(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.1}$$

其中，A、B 表示两个集合，$|A \cap B|$ 表示这两个集合的交集，$|A \cup B|$ 表示两个集合的并集。

应用 Jaccard 算法，字符串之间的相似度可表示为公式(2.2)：

$$\text{StringJaccard}(S_1, S_2) = \frac{|\text{tokenize}(S_1) \cap \text{tokenize}(S_2)|}{|\text{tokenize}(S_1) \cup \text{tokenize}(S_2)|} \tag{2.2}$$

其中，S_1、S_2 代表两个待比较的字符串，函数 tokenize(S)可以将字符串 S 转换为由字段(token)组成的集合$\{S_1, S_2, \cdots, S_n\}$。

下面举个例子来说明 Jaccard 相似度在字符串上的应用。假设存在两个字符串 S_1 和 S_2，S_1 = “Thomas Sean Connery”，S_2 = “Sir Sean Connery”，计算两个字符串的相似度。先由函数 tokenize 以空格作分隔，得到：

$$\text{tokenize}(S_1) = \{\text{Thomas}, \text{Sean}, \text{Connery}\}$$
$$\text{tokenize}(S_2) = \{\text{Sir}, \text{Sean}, \text{Connery}\}$$

然后利用公式(2.2)计算得到字符串 S_1 和 S_2 的相似度：

$$\text{StringJaccard}(S_1, S_2) = \frac{2}{4}$$

Jaccard 相似度多用于检测无拼写错误字符串的相似度，且对于字符串内各 token 顺序的变化是不敏感的。

2.1.2　基于 TF-IDF 的相似度算法

余弦相似度计算[2]是基于字段的一种相似度计算方法，其计算如公式(2.3)所示：

$$\text{CosineSimilarity}(\boldsymbol{V}, \boldsymbol{W}) = \cos(\alpha) = \frac{\boldsymbol{V} \cdot \boldsymbol{W}}{\|\boldsymbol{V}\| \cdot \|\boldsymbol{W}\|} \tag{2.3}$$

其中，$\boldsymbol{V}$、$\boldsymbol{W}$ 是两个 n 维向量，这两个向量需要由 TF-IDF 方法求得。

TF-IDF(Term Frequency-Inverse Document Frequency)是一种统计方法，其主要思想是：如果某个词比较少见，但是它在这篇文章中多次出现，那么它很可能就反映了这篇文章的特性，则认为此词或者短语具有很好的类别区分能力，适合用来分类。TF(Term Frequency)表示某个词在文档中出现的频率，IDF(Inverse Document Frequency)与包含关键词 t 的文档数成反比，IDF 越大，说明 t 的类别区分能力越好。

在字符型属性值的相似度计算中，将属性值作为文档来处理，并用向量表示它们。设两个字符串型属性值向量形式分别为：$\boldsymbol{V}(a_1, a_2, \cdots, a_n)$，$\boldsymbol{W}(b_1, b_2, \cdots, b_m)$，TF-IDF 值的计算步骤如下。

步骤 1：应用统计的方法得出每个关键词在相应的属性值里面出现的频率，即词频，记为 $\mathrm{TF}_{\boldsymbol{V},a_1}$，$\mathrm{TF}_{\boldsymbol{V},a_2}$，…，$\mathrm{TF}_{\boldsymbol{V},a_n}$，$\mathrm{TF}_{\boldsymbol{W},b_1}$，$\mathrm{TF}_{\boldsymbol{W},b_2}$，…，$\mathrm{TF}_{\boldsymbol{W},b_m}$。

步骤 2：由公式 $\mathrm{IDF}_t = \log(|D_t|/1+|D|)$ 计算得出每个关键词的 IDF 值，其中 $|D|$ 表示文档总数，$|Dt|$ 表示包含关键词 t 的文档数，分别记为 $\mathrm{IDF}_{\boldsymbol{V},a_1}$，$\mathrm{IDF}_{\boldsymbol{V},a_n}$，$\mathrm{IDF}_{\boldsymbol{W},b_1}$，$\mathrm{IDF}_{\boldsymbol{W},b_m}$。

步骤 3：将向量中各个关键词的 TF、IDF 值分别相乘，得到各个关键词的 TF-IDF 值，记为 $\mathrm{TFIDF}_{\boldsymbol{V},a_1}$，$\mathrm{TFIDF}_{\boldsymbol{V},a_2}$，…，$\mathrm{TFIDF}_{\boldsymbol{V},a_n}$，$\mathrm{TFIDF}_{\boldsymbol{W},b_1}$，$\mathrm{TFIDF}_{W,b_m}$。将这些关键值组成新的向量，记为

$$\mathrm{TFIDF\_V}(\mathrm{TFIDF}_{\boldsymbol{V},a_1}, \mathrm{TFIDF}_{\boldsymbol{V},a_2}, \cdots, \mathrm{TFIDF}_{\boldsymbol{V},a_n})$$

$$\mathrm{TFIDF\_W}(\mathrm{TFIDF}_{\boldsymbol{W},b_1}, \mathrm{TFIDF}_{\boldsymbol{W},b_2}, \cdots, \mathrm{TFIDF}_{\boldsymbol{W},b_m})$$

使用上面计算得出的向量以及余弦算法计算出相似度值，记为 $\mathrm{sim}(\boldsymbol{V}, \boldsymbol{W})$。

2.1.3 基于 q-grams 的相似度算法

在基于 q-grams 的字符串相似度计算中，先将各字符串切割成长度为 q 的 grams，然后再进行相似度计算。下面举例说明基于 q-grams 的相似度计算过程。

假设有两个字符串 S_1 = “Henri Waternoose” 和 S_2 = “Henry Waternose”，两个字符串分别生成 3-grams 如下所示：

```
    3-grams of S1= {##H, #He, Hen, enr, nri, ri_, i_W, _Wa, Wat, ate, ter, ern,
rno, noo, oos, ose, se# , e## }
    3-grams of S2= {##H, #He, Hen, enr, nry, ry_, y_W, _Wa, Wat, ate, ter, ern,
rno, nos, ose, se# , e## }
```

其中，“_”代表空格，“#”代表填补符号。

分别利用 2.1.1 节和 2.1.2 节中提到的 Jaccard 和 Cosine 相似度算法计算 S_1 和 S_2 的相似度，其中 $\boldsymbol{V}$、$\boldsymbol{W}$ 分别代表 S_1、S_2 的 3-grams 集合。

$$\text{StringJaccard}(S_1, S_2) = 13/22 = 0.59$$

$$\text{CosineSimilarity}(\boldsymbol{V}, \boldsymbol{W}) = \frac{1.04^2 \times 13}{\sqrt{1.04^2 \times 13 + 1.34^2 \times 5} \times \sqrt{1.04^2 \times 13 + 1.34^2 \times 4}} \approx 0.64$$

基于 q-grams 的相似度计算更适用于存在拼写错误的字符串间的比较。

2.2　基于编辑距离的相似度算法

2.2.1　Levenshtein 距离算法

Levenshtein 距离算法[3]是由俄国科学家 Levenshtein 提出的。两个字符串 str_1 和 str_2 的编辑距离是将字符串 str_1 转换成 str_2 所使用的最少编辑操作次数。编辑操作有三种：

1）插入操作：在字符串中插入一个字符。

2）删除操作：从字符串中删除一个字符。

3）替换操作：将字符串中某个位置的字符替换成另外一个字符。

每一个编辑操作的代价都是由 1 来指代 Levenshtein 距离。例如，“best”和“best”的编辑距离是 0，而“best”和“bent”的编辑距离是 1，从“best”转换成“bent”需要一个编辑操作。

例如，S_1＝“Sean”，S_2＝“Shawn”。从 S_1 到 S_2 有很多种变换方式，利用动态规划算法可以得到最小的编辑距离。其大概过程是：初始化一个$(|S_1|+1)\times(|S_2|+1)$的矩阵 $\boldsymbol{M}$，$\boldsymbol{M}_{i,j}$代表矩阵 $\boldsymbol{M}$ 中第 i 行第 j 列的值。其中，$0\leqslant i\leqslant|S_1|$，$0\leqslant j\leqslant|S_2|$，$S_{1,i}$表示字符串 S_1 中的第 i 个字符。如公式(2.4)、(2.5)、(2.6)所示。

$$\boldsymbol{M}_{i,0}=i \tag{2.4}$$

$$\boldsymbol{M}_{0,j}=j \tag{2.5}$$

$$\boldsymbol{M}_{i,j}=\begin{cases}\boldsymbol{M}_{i-1,j-1} & S_{1,i}=S_{2,j}\\ 1+\min(\boldsymbol{M}_{i-1,j},\boldsymbol{M}_{i,j-1},\boldsymbol{M}_{i-1,j-1}) & \text{其他}\end{cases} \tag{2.6}$$

则动态规划过程如图 2-1 所示。

		S	h	a	w	n
	0	1	2	3	4	5
S	1					
e	2					
a	3					
n	4					

a）初始化第一行和第一列后的矩阵

		S	h	a	w	n
	0	1	2	3	4	5
S	1	0	1	2	3	4
e	2					
a	3					
n	4					

b）利用公式（2.5）计算第二行后的矩阵

		S	h	a	w	n
	0	1	2	3	4	5
S	1	0	1	2	3	4
e	2	1	1	2	3	4
a	3	2	2	1	2	3
n	4	3	3	2	2	2

c）编辑距离存在最终矩阵的右下角

图 2-1　动态规划计算过程

最终 LevDist(Sean，Shawn)＝2。

2.2.2　Jaro 和 Jaro-Winkler 距离算法

Jaro 算法[4]是一种主要用于比较姓名的字符串比较算法，对于字符串 str_1 和 str_2，这个算法的基本计算步骤如下。

1）计算字符串的长度$|str_1|$和$|str_2|$。

2）寻找两个字符串中的公共字符 c：公共字符指的是满足以下标准的所有 $str_1[i]$和 $str_2[j]$，其中 $str_1[i]=str_2[j]$，同时 $|i-j|\leqslant\frac{1}{2}\min\{|str_1|,|str_2|\}$。

3）找到变换的数量 t：比较 str_1 和 str_2 的第 i 个公共字符，每一个不相匹配的字符就是一个变换。

Jaro 计算公式如式(2.7)所示：

$$\text{Jaro}(str_1,str_2)=\frac{1}{3}\left(\frac{c}{|str_1|}+\frac{c}{|str_2|}+\frac{c-t/2}{c}\right) \tag{2.7}$$

Winkler 和 Thibaudeau 修正了 Jaro 度量[5]，并给出了一个前缀匹配，因为前缀匹配比姓名匹配更加重要。Jaro-Winkler 定义了一个前缀 p，如果前缀部分有长度为 ℓ 的部分相同，则 Jaro-Winkler 距离计算公式如式(2.8)所示：

$$d_w=d_j+[\ell p(1-d_j)] \tag{2.8}$$

其中：d_j 是两个字符串的 Jaro 距离；ℓ 是前缀相同部分的长度，但是规定最大为 4；p 是调整分数的常数，规定不能超过 0.25，不然可能出现 d_w 大于 1 的情况，Winkler 将这个常数定义为 0.1。

这样，字符串“MARTHA”和“MARHTA”的 Jaro-Winkler 距离为：

$$d_w=0.944+[3\times0.1\times(1-0.944)]=0.961$$

2.3　混合的相似度算法

2.3.1　扩展的 Jaccard 相似度算法

本小节介绍两种扩展的 Jaccard 方法，第一种增添了相似的token[6]，

第二种引入了权重函数[7]。

假设有两个字符串 S_1 和 S_2，应用 tokenize 函数将两个字符串分割成由 token 组成的集合，第一种扩展的 Jaccard 方法是对 token 进行相似度计算，找出相似的 token，这样可以包容较小的拼写错误。

通常，TokenSim(t_1，t_2)作为相似度计算函数来计算 token 的相似度，其中 $t_1 \in \text{tokenize}(S_1)$，$t_2 \in \text{tokenize}(S_2)$。相似的 token 定义如下：

$$\text{Shared}(S_1, S_2) = \{(t_i, t_j) \mid t_i \in \text{tokenize}(S_1) \wedge t_j \in \text{tokenize}(S_2) : \text{TokenSim}(t_i, t_j) > \theta_{\text{string}}\}$$

其中，θ_{string}为判断两个 token 是否相似的阈值。

唯一存在于 S_1 中的 token 为：

$$\text{Unique}(S_1) = \{t_i \mid t_i \in \text{tokenize}(S_1) \wedge (t_i, t_j) \notin \text{Shared}(S_1, S_2)\}$$

唯一存在于 S_2 中的 token 为：

$$\text{Unique}(S_2) = \{t_i \mid t_i \in \text{tokenize}(S_2) \wedge (t_i, t_j) \notin \text{Shared}(S_1, S_2)\}$$

第二种扩展为匹配和未匹配的 token 引入了权重函数 w，通常与第一种结合起来使用。集合函数 A 将各权重集合起来，其计算如公式(2.9)所示：

$$\text{HybridJaccard} = \frac{A_{(t_i, t_j) \in \text{Shared}(S_1, S_2)} w(t_i, t_j)}{A_{(t_i, t_j) \in \text{Shared}(S_1, S_2)} w(t_i, t_j) + A_{(t_i) \in \text{Unique}(S_1)} w(t_i) + A_{(t_j) \in \text{Unique}(S_2)} w(t_j)} \tag{2.9}$$

下面举例说明扩展的 Jaccard 方法。

假设有两个字符串 S_1 = “Henri Waternoose”，S_2 = “Henry Peter Waternose”。利用编辑距离方法度量字符串间的相似度，且 θ_{string}=1。

利用上述公式，得到

$$\text{Unique}(S_1)=\varnothing,\text{Unique}(S_2)=\{\text{Peter}\}$$
$$\text{Shared}(S_1,S_2)=\{(\text{Henri},\text{Henry}),(\text{Waternoose},\text{Waternose})\}$$

我们假设两个 token(t_i，t_j)的权重计算公式为 $1-\frac{\text{LevDist}(t_i,\ t_j)}{\max(|t_i|,\ |t_j|)}$，集合函数 A 简单地将权重进行加和运算。基于以上假设，两个字符串的扩展 Jaccard 相似度计算如下：

$$\text{HybridJaccard}(S_1,S_2)=\frac{0.8+0.9}{0.8+0.9+0+1}=0.63$$

2.3.2　Monge-Elkan 相似度算法

本小节介绍 Monge-Elkan 相似度算法[8]。首先假设有两个字符串 S_1 和 S_2，应用 tokenize 函数将两个字符串分割成由 token 组成的集合，然后将字符串 S_1 中的每个 token t_i 与 S_2 中的所有 token 进行相似度计算，找出 S_2 中与 t_i 相似度最大的 t_j。然后将 S_1 中所有 token 的相似度最大值相加。Monge-Elkan 相似度计算如公式(2.10)所示：

$$\text{MongeElkanSim}(S_1,S_2)=\frac{1}{|\text{tokenize}(S_1)|}\sum_{i=1}^{|\text{tokenize}(s_1)|}\max_{j=1}^{|\text{tokenize}(s_2)|}\text{TokenSim}(t_i,t_j)\quad(2.10)$$

下面举例说明 Monge-Elkan 相似度计算方法。假设有两个字符串 S_1＝“Henri Waternoose”，S_2＝“Henry Peter Waternose”。在 S_2 中与“Henri”最相近的是“Henry”，与“Waternoose”最相近的是“Waternose”。假设以上两个 token 的相似度最大值分别为 0.8 和 0.9，则计算得到：

$$\text{MongeElkanSim}(S_1,S_2)=\frac{0.8+0.9}{2}=0.85$$

2.3.3　Soft TF-IDF 相似度算法

本小节讨论基于 TF-IDF 扩展的余弦相似度方法[9]。该方法的基本

思想与扩展的 Jaccard 方法相同。其中 TokenSim(t_1，t_2)应用辅助串相似度计算函数来计算 token 的相似度，相似的 token 定义如下：

$$\begin{aligned}&\text{Close}(\theta_{\text{string}},S_1,S_2)\\&=\{t_i\mid t_i\in\text{tokenize}(S_1)\wedge\exists t_j\in\text{tokenize}(S_2):\text{TokenSim}(t_i,t_j)>\theta_{\text{string}}\}\end{aligned}$$

与扩展的 Jaccard 中 Shared(S_1，S_2)不同的是，Close(θ_{string}，S_1，S_2)只包括了来自于 S_1 中的 token。来自于 S_2 中的与 Close 中 S_1 相似的 token 定义如下：

$$\max\text{Sim}(t_i,t_j)=\max_{t_j\in\text{tokenize}(S_2)}\text{TokenSim}(t_i,t_j)$$

其中，$t_j\in\text{tokenize}(S_2)$，$t_i\in\text{Close}(\theta_{\text{string}}$，$S_1$，$S_2)$。

扩展的余弦相似度计算方法也称作 Soft TF-IDF，定义如公式(2.11)所示：

$$\text{SoftTFIDF}(S_1,S_2)=\sum_{t_i\in\text{Close}(\theta_{\text{string}},S_1,S_2)}\left(\frac{\text{tf}-\text{idf}_{t_i}}{\|\boldsymbol{V}\|}\times\frac{\text{tf}-\text{idf}_{t_j}}{\|\boldsymbol{W}\|}\times\max\text{Sim}(t_i,t_j)\right)\tag{2.11}$$

其中，$\boldsymbol{V}$，$\boldsymbol{W}$ 分别代表 S_1、S_2 的向量值。

下面举个例子来说明 Soft TF-IDF 相似度计算方法。假设有两个字符串 S_1＝“Henri Waternoose”，S_2＝“Henry Peter Waternose”，代表 S_1、S_2 的向量值 $\boldsymbol{V}$、$\boldsymbol{W}$ 分别是 $\boldsymbol{V}$＝{0.6，0.6，0，0，0}，$\boldsymbol{W}$＝{0，0，0.5，0.3，0.6}。

可以确定 Close(θ_{string}，S_1，S_2)＝{Henri，Waternoose}，那么 S_1、S_2 的 Soft TF-IDF 相似度计算如下：

$$\begin{aligned}\text{softTFIDF}(S_1,S_2)&=\frac{0.6}{\sqrt{0.6^2+0.6^2}}\times\frac{0.5}{\sqrt{0.5^2+0.3^2+0.6^2}}\times0.8\\&\quad+\frac{0.6}{\sqrt{0.6^2+0.6^2}}\times\frac{0.6}{\sqrt{0.5^2+0.3^2+0.6^2}}\times0.9\\&\approx0.79\end{aligned}$$

2.4　数值型数据相似度算法

数值型数据也是实体识别中经常遇到的数据类型，比如生日、年龄、年份、价格、折扣等。在一般的数据处理中，都将数字当作字符对待，然而这种处理方式不适合于数值型属性的比较。比如，给定年份 1999 和 2000，两者的字符串相似度非常小，但是两者实际只相差一年，应该通过求差值来解决：|1999－2000|。在实际应用中，根据实际可采用精确距离或范围距离来度量数值型数据的相似度。日期型相似度需综合年、月、日来计算其相似度，而价格相似度不仅需要考虑不同币种的差异，还需要考虑相对的差值。下面分别介绍这几种数值型相似度的计算方法。

2.4.1　数字型相似度算法

1. 精确距离算法

若两个数字型字符串完全相同，则相似度为 1，否则为 0。例如，对于图书的 ISBN 属性，只有当两条图书记录的 ISBN 属性完全相同时，才能认为它们所描述的为同一本书，因此可以利用精确距离算法来计算 ISBN 属性间的相似度。

2. 范围距离算法

如果当两个数字型属性值 n_1 和 n_2 在数值上的差小于一个阈值，对应的两条实体记录仍存在相互等价的可能性，则对于这样的数字型数据我们可以采用范围距离算法来计算 n_1 和 n_2 的相似度（如公式(2.12)所示）。其中，n_1 和 n_2 是两个数字型数据的值，n 是 n_1 和 n_2 的平均值。

$$\mathrm{Sim}(n_1,n_2)=1-\frac{\sqrt{\dfrac{(n_1-n)^2+(n_2-n)^2}{2}}}{n} \tag{2.12}$$

2.4.2 日期型相似度算法

日期型数据的表达方式多种多样。在计算日期型数据的相似度前，首先要将所有的日期型数据都转换成统一的表示形式“*yyyy.mm.dd*”，其中“*yyyy*”表示年份，“*mm*”表示月份，“*dd*”表示日期。日期型数据相似度的计算方法如下。设 d_1，d_2 是两个日期型属性值：

- 若日期的比较要求精确到年，则只比较 d_1 和 d_2 的“*yyyy*”，若两个属性值的“*yyyy*”相等，则两个日期型数据的相似度为 1，否则为 0。
- 若日期的比较要求精确到月，则比较 d_1 和 d_2 的“*yyyy*”和“*mm*”，若“*yyyy*”和“*mm*”都相等，则两个日期型数据的相似度为 1，否则为 0。
- 若日期的比较要求精确到日，则比较 d_1 和 d_2 的“*yyyy*”“*mm*”和“*dd*”，若“*yyyy*”“*mm*”和“*dd*”全部相等，则两个日期型数据的相似度为 1，否则为 0。

2.4.3 价格型相似度算法

对于价格这种特殊的数据类型，标准的计算文本相似度的方法是不适用的，而需要一种新的衡量标准来计算。考察两个价格之间的匹配程度，绝对的数值差异是不恰当的，需要考察数值的相对差值。如 $28 与 $30 的绝对差值是 $2，而 $2800 与 $3000 的绝对差值是 $200，可见，价格的匹配程度不能由绝对差值来衡量。可采用类似于范围距离定义价格相似度。

假设 p_1 和 p_2 是两条价格类型数据，则 p_1 和 p_2 的相似度定义如公式(2.13)所示。其中，p 是指两个价格 p_1 和 p_2 的平均值。

$$\mathrm{Sim}(p_1,p_2)=1-\left(\frac{\sqrt{\frac{(p_1-p)^2+(p_2-p)^2}{2}}}{p}\right) \tag{2.13}$$

另外需要注意的是，不同领域数值的相似度度量准则大不相同，比如在微观测量中(如分子结构)，1 毫米的差距是非常大的；而在宏观测量中(如天文测距)，1 毫米的差距是微不足道的。由此可见，数值型数据的比较函数需要根据领域知识来设计。

2.5　本章小结

本章介绍了基于字段的相似度计算算法、基于编辑距离的相似度计算算法、混合的相似度计算算法和数值型相似度计算方法。在实体识别方法中，针对不同类型的属性，应基于相应的相似度计算算法度量记录属性的相似度。

参考文献

[1] Alvaro E Monge, Charles P Elkan. The field matching problem: Algorithms and applications[C]. In Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, 1996, 24, 35: 267-270.

[2] Felix Naumann, Melanie Herschel. An Introduction to duplicate detection[M]. Morgan and Claypool pablishers, 2010.

[3] Gonzalo Navarro. A guided tour to approximate string matching[J]. ACM Comput. Survey, 2001, 33(1): 31-88.

[4] Matthew A Jaro. Advances in record linking methodology as applied to matching the 1985 census of tampa florida[J]. American Statistical Association, 1989, 84(406): 414-420.

[5] William E Winkler, Yves Thiboudeau. An application of the Felligi Sunter model of record linkage to the 1990 US decennial census[R]. Technical report, US Bureau of the Census, 1991.

[6] Rohit Ananthakrishna, Surajit Chaudhuri, Venkatesh Ganti. Eliminating fuzzy duplicates in data warehouses[C]. In Proc. 28th Int. Conf. on Very Large

Data Bases，2002：586-597.

[7] Melanie Weis，Felix Naumann. Dogmati X tracks down duplicates in XML[C]. In Proc. ACM SIGMOD Int. Conf. on Management of Data，2005：431-442.

[8] Alvaro E Monge，Charles P Elkan. The field matching problem：Algorithms and applications[C]. In Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining，1996：267-270.

[9] Cohen W，Ravikumar P，Fienberg S. A comparison of string distance metrics for namematching tasks[C]. In Workshop on Information Integration on the Web，held at IJCAI，2003：73-78.