相似度计算整理

liz*

May 22, 2009

Contents

*Email: shengyan1985@gmail.com

1 概述

摘自Dekang Lin的An Information-Theoretic Definition of Similarity.
已经有很多中相似度衡量方法，如:

1. Information content
2. Mutual information
3. Dice coefficient
4. Cosine coefficient
5. Distance-based measurements
6. Feature contrast model

每种方法都适合于特定的应用领域，给予距离的衡量方法，是假设这个领域中的东西可以表示成一个网络，如果文档集合不能表示成一个网络，就不能使用基于距离的方法. Cosine方法，只适用于当对象表示成数值向量时.

1.1 相似度的直觉

1. A和B的相似度和它们的共性有关，共同的东西越多，就越相似.
2. 和差异有关，差别越大，不同点越多，A和B就越不相似.
3. 同一事物，如果A和B是同一个东西，那么它们最大相似，不管它们拥有多少共同点或不同点.

1.2 相似度的假设

1. I(common(A, B)
   衡量A和B的commonality, common(A,B) is a proposition on that states the commonalities between A and B; I(s) is the amount of information contained in proposition s. 命题s包含的信息量.
   如，A是苹果，B是橙子，它们公共的命题可以描述为A和B都是水果. 而信息量可以用负log概率来计算.
   $$I(common(A,B)) = -\log P(fruit(A) and fruit(B))$$
2. I(description(A, B)) - I(common(A, B)), description(A, B)是表示A和B是什么的一个命题. 那么，这个式子表示出的意思为，在A和B的所有描述中，除了共同的信息外，剩余的就是A和B的不同点.
3. A和B的相似度
   $$sim(A,B) = f(I(common(A,B)), I(description(A,B)))$$
   $$\{(x,y)|x \geq 0, y > 0, y \geq x\}$$
4. 如果两个对象是同一个对象，那么定义相似度为1
   在I(common(A,B)) = I(description(A,B))时，可以看成是所有的描述都是相同的，那么很容易想到它们是同一东西，$\forall x > 0, f(x,x) = 1$
5. $$\forall y > 0, f(0,y) = 0$$
   对象A和B的相似度衡量可以从不同的视角计算，如果在不同的视角计算出来的相似度，之间不具有可比性. 比如说，通过计算文档中的单词，或通过计算文档的属性来，这两者不具有可比性. 那么，全局的相似度通过给不同视角上的相似度加上权值，即，计算加权平均值.
6. $$\forall x_1 \leq y_1, x_2 \leq y_2 : f(x_1 + x_2, y_1 + y_2) = \frac{y_1}{y_1 + y_2} f(x_1, y_1) + \frac{y_2}{y_1 + y_2} f(x_2, y_2)$$
   这个说明从不同的角度出发，通过加权，计算全局相似度.

1.3 Similarity Theorem

$$sim(A,B) = \frac{\log P(common(A,B))}{\log P(description(A,B))}$$

也就是说， 相似度的衡量可以使用是共性与总描述的比值来衡量． 还句话说， 如果我们知道了共性，它们的相似度表示需要多少信息才决定这两个对象是什么， 区分开对象来．

1.4 Similarity between Ordinal Values

对于顺序的值的计算， 比如说， excellent, good, average, bad, awful这种． 也还是利用各自频率， 如

$$sim(excellent, average) = \frac{2 \times \log P(excellent \vee good \vee average)}{\log P(excellent) + \log P(average)} = \frac{2 \times \log(0.05 + 0.10 + 0.50)}{\log 0.05 + \log 0.50} = 0.23$$

1.5 向量形式

如果某个对象可以表示成向量形式， 那么， 可以使用余弦衡量， 还可以加权． ...

1.6 String Similarity

1. Hamming distance
   It measures the minimum number of substitutions required to change one into the other, or the number of errors that transformed one string into the other. 即只考虑替换字符的最小个数．

```
1 def hamdist(s1, s2):
2     assert len(s1) == len(s2)
3     return sum(ch1 != ch2 for ch1, ch2 in zip(s1, s2))
```

   特殊的性质： 对于固定长度的n来说， 海明距离为n维向量空间中， 单词a和b...For a fixed length n, the Hamming distance is a metric on the vector space of the words of that length, as it obviously fulfills the conditions of non-negativity, identity of indiscernibles and symmetry, and it can be shown easily by complete induction that it satisfies the triangle inequality as well. The Hamming distance between two words a and b can also be seen as the Hamming weight of a-b for an appropriate choice of the - operator.
   对于二进制串的a和b来说， 海明距离就是a XOR b之后的1的个数,这在hypercube graph可以很清楚的看到从a转到b需要几条路径．

   Hamming weight: 字符串中， 非空字符的个数． 如二进制串中为， 1的个数; 单词字符串中， 表示a-z字符的个数．
   应用: An example of an application of Hamming weight is analysis of modular exponentiation by squaring, where the number of modular multiplications required for an exponent e is log2 e + weight(e). This is the reason that the public key value e used in RSA is typically chosen to be a number of low Hamming weight. It also determines path lengths between nodes in Chord distributed hash tables.
   计算二进制串中的1的个数， 可以通过X & (X-1)操作来， 统计X->0经过这个操作的次数．

2. Edit Distance

$$sim_{edit}(x,y) = \frac{1}{1 + editDist(x,y)}$$

   Levenshtein distance: The Levenshtein distance between two strings is given by the minimum number of operations needed to transform one string into the other, where an operation is an insertion, deletion, or

substitution of a single character.

```
int LevenshteinDistance(char s[1..m], char t[1..n])
 // d is a table with m+1 rows and n+1 columns
 declare int d[0..m, 0..n]

 for i from 0 to m
  d[i, 0] := i
 for j from 0 to n
  d[0, j] := j

 for j from 1 to n
  for i from 1 to m {
   if s[i] = t[j] then
     cost := 0
   else
     cost := 1
   d[i, j] := minimum(
                d[i−1, j] + 1,     // insertion
                d[i, j−1] + 1,     // deletion
                d[i−1, j−1] + cost   // substitution
              )
  }

  return d[m, n]
```

时空复杂度都为$O(mn)$

3. Longest common subsequence
Longest common subsequence: to find the longest subsequence common to all sequences in a set of sequences (often just two). It is a classic computer science problem, the basis of diff (a file comparison program that outputs the differences between two files), and has applications in bioinformatics.
NP-hard, 时间复杂度为...没懂.
最长公共子序列和最长公共子串还不一样, 子序列是不连续的, 公共子串是连续的.
最长公共子序列解决定义:

$$LCS(X_i, Y_j) = \begin{cases} \emptyset & \text{if } i = 0 \text{ or } j = 0 \\ (LCS(X_{i-1}, Y_{j-1}), x_i) & \text{if } x_i = y_j \\ longest(LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j)) & \text{if } x_i \neq y_j \end{cases}$$

Computing the length of the LCS

```
function  LCSLength(X[1..m], Y[1..n])
    C = array(0..m, 0..n)
    for i := 0..m
       C[i,0] = 0
    for j := 0..n
       C[0,j] = 0
    for i := 1..m
        for j := 1..n
            if X[i] = Y[j]
                C[i,j] := C[i−1,j−1] + 1
            else:
                C[i,j] := max(C[i,j−1], C[i−1,j])
    return C[m,n]
```

Reading out an LCS

```
function backTrace(C[0..m,0..n], X[1..m], Y[1..n], i, j)
    if i = 0 or j = 0
        return ""
    else if  X[i−1] = Y[j−1]
        return backTrace(C, X, Y, i−1, j−1) + X[i−1]
    else
```

```
        if C[i,j−1] > C[i−1,j]
            return backTrace(C, X, Y, i, j−1)
        else
            return backTrace(C, X, Y, i−1, j)
```

Reading out all LCSs

```
function backTraceAll(C[0..m,0..n], X[1..m], Y[1..n], i, j)
    if i = 0 or j = 0
        return {}
    else if X[i] = Y[j]
        return {Z + X[i] for all Z in backTraceAll(C, X, Y, i−1, j−1)}
    else
        R := {}
        if C[i,j−1] ≥ C[i−1,j]
            R := backTraceAll(C, X, Y, i, j−1)
        if C[i−1,j] ≥ C[i,j−1]
            R := R ∪ backTraceAll(C, X, Y, i−1, j)
        return R
```

Print the diff

```
function printDiff(C[0..m,0..n], X[1..m], Y[1..n], i, j)
    if i > 0 and j > 0 and X[i] = Y[j]
        printDiff(C, X, Y, i−1, j−1)
        print "  " + X[i]
    else
        if j > 0 and (i = 0 or C[i,j−1] ≥ C[i−1,j])
            printDiff(C, X, Y, i, j−1)
            print "+ " + Y[j]
        else if i > 0 and (j = 0 or C[i,j−1] < C[i−1,j])
            printDiff(C, X, Y, i−1, j)
            print "- " + X[i]
```

4. Manhattan distance
   又称Taxicab geometry,, 二维坐标系中，$P_1(x_1, y_1)$，$P_2(x_2, y_2)$，两者之间的距离为$|x_1 - x_2| + |y_1 - y_2|$
5. Euclidean distance
   又称欧式距离，$\sqrt{|x_1 - x_2|^2 + |y_1 - y_2|^2}$
6. Trigrams
   衡量两个字符串相似度，使用字符串的3个子串进行.

$$sim_{tri}(x, y) = \frac{1}{1 + |tri(x)| + |tri(y)| - 2 \times |tri(x) \cap tri(y)|}$$

1.7 Word Similarity

在整个词集中，提取三元组(head, a dependency type, a modifier)，这主要的是单词之间的语法关系，然后根据这个对这些单词提取对应的特征.

$$sim(w_1, w_2) = \frac{2 \times I(F(w_1) \cap F(w_2))}{I(F(w_1)) + I(F(w_2))}, I(S) = -\sum_{f \in S} \log P(f)$$

1.8 Semantic Similarity in a Taxonomy

涉及到语义的话，需要像WordNet这样的语义本体，规定好的词和词之间的关系.

1.8.1 Resnik

$$sim_{Resnik}(A, B) = \frac{1}{2} I(common(A, B))$$

1.8.2 Wu and Palmer

基于当前概念到root概念的层次距离.

$$sim_{Wu\&Palmer}(A,B)=\frac{2\times N_3}{N_1+N_2+2\times N_3}$$

1.8.3 Information Content

公共概念的信息量.

$$sim(x_1,x_2)=\frac{2\times \log P(C_0)}{\log P(C_1)+\log P(C_2)}$$

2 Tversky模型

基于集合理论，Tversky的相似度衡量过程是由各个terms的匹配过程.

$$S(a,b)=\frac{f(B_1\cap B_2)}{f(B_1\cap B_2)+\alpha f(B_1-B_2)+\beta f(B_2-B_2)}$$

B_1,B_2是a，b 的特征，f是相似度的一个衡量方法.

3 An improved Tversky's model

$$S(a,b)=\frac{|B_1\cap B_2|}{|B_1\cap B_2|+\alpha(a,b)|B_1-B_2|+(1-\alpha(a,b))|B_2-B_2|}$$

$$if depth(a)\leq depth(b);\alpha(a,b)=\frac{depth(a)}{depth(a)+depth(b)}$$

$$if depth(a)>depth(b);\alpha(a,b)=1-\frac{depth(a)}{depth(a)+depth(b)}$$

基于Tversky模型的改进还有很多,,, 主要思想就是集合的交/并，最简单的一种方式为，

$$S(a,b)=\frac{|B_1\cap B_2|}{|B_1\cup B_2|}$$

4

5 语义相似度

在Anna Formica的Concept similarity in Formal Concept Analysis: An information content approach中，使用一种基于信息量的方法.
计算名词n在整个文集中出现的频率:

$$p(n)=\frac{freq(n)}{M}$$

如果两个名词n_1和n_2是同义词， 那么$ics(n_1,n_2)=1$ 其他的为: $ics(n_1,n_2)=\frac{2\log p(n')}{\log p(n_1)+\log p(n_2)}$，$-\log p(n')=\max_{n\in S(n_1,n_2)}[-\log p(n)]$， $S(n_1,n_2)$是两个名词概念在wordnet中的所有公共上界.
这样，可得到两个概念的相似度计算公式:

$$Sim((E_1,I_1),(E_2,I_2))=\frac{|(E_1\cap E_2)|}{max\{|E_1|,|E_2|\}}*\omega+\frac{M(I_1,I_2)}{max\{|I_1|,|I_2|\}}*(1-\omega)$$

同时计算了概念的外延，和内涵.$M(I_1,I_2)$为最大值配对，配对权值计算依靠ics计算.