Working Paper Series
ISSN 1177-777X

Natural Language Processing Algorithm Descriptions

Daniel McEnnis

Working Paper: xx/2009
August 23, 2009

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

Natural Language Processing Algorithm Descriptions

Daniel McEnnis
University of Waikato, Hamilton, New Zealand
dm75@waikato.ac.nz

1. Introduction

The following set of algorithms deal with term extraction using natural language processing of flat documents. Some algorithms use a knowledge base (Wikipedia), while others are designed to use the documents as the knowledge base. Wiki Fast Disambiguation performs term disambiguation over a set of previously extracted terms where all meanings of the terms are known in advance. The concept pruning algorithm eliminates peripheral terms, forming a hierarchal set description of a documents key concepts from a set of concepts.

2. Definitions

1. Let $Term_ij$ be the jth sense of the ith term.
2. Let $TermSet$ be the set of $Term_ij \forall i, j$
3. Let $TermSet - k$ be the set of $Term_ij$ excluding $Term_ij$ where $i = k$
4. Let Node be a TermSet with a parent Node reference and potentially unordered Children node references

3. Wiki Fast Disambiguation

The original algorithm, initially declared by Anna Huang on 17-8-2009, is an exponential algorithm over the number of terms, but provably maximizes AverageRelatedness. The average relatedness presented here is a reconstruction of the function presumed to be underlying the brute force method. Trivially, this brute force method achieves an output that maximizes average relatedness. When performing term extraction on a short document, this exponential algorithm has acceptable performance. However, over larger documents such as books, this algorithm is computationally infeasible.

Wiki Fast Disambiguation is designed to for use in the processing large documents. It performs a near cubic comparison of possible term definitions. The initial loop is trivially cubic, however, it is unproven how many iterations of the second loop occur, or even if the second loop converges to a solution.

Also, it is unproven if the output is equivalent to the brute force method or what conditions it fails to converge.

Note: this method and the brute force method makes the implicit assumption that no more than one sense of a word is applicable for any given docuement. While this is generally correct, performing pruning on a term set expanded using polysemy without disambiguation (see Concept Pruning) eliminates this problem.

double AverageRelated(TermSet termSet){
$doublesum = 0.0$
$\forall Term_ijtermintermSet\{$
$\forall Term_ijterm2 \in termSetwheretermterm2! = term\{$
sum += Distance(term,term2)
}
}
return $sum/Count(Term_ij)$
}
TermSet PolysemyDetection(TermSet termSet){
TermSet ret = termSet
double max = $NegativeInfinity$
Term disambiguated
// perform initial guesses
$\forall i \in ret\{$
TermSet-i termSet-i = ret - $Term_i$
$\forall j \in termSet_i\{$
TermSet local = termSet-i + $Term_ij$
if(AverageRelated(local) > max){
disambiguated=$Term_ij$
max = AverageRelated(local)
}
}
ret = termSet-i $\cup$ disambiguated
}
// check the rest are still disambiguated correctly
boolean done = false
while(!done){
done = true
$\forall i \in termSet\{$
TermSet-i termSet-i = ret - $Term_i$
max = AverageRelated(ret)
disambiguated = ret_i
}
$\forall j \in termSet_i! = ret_i\{$
TermSet local = termSet-i + $Term_ij$
if(AverageRelated(local) > max){
disambiguated=$Term_ij$
max = AverageRelated(local)

done = false
}
}
ret = termSet-i + disambiguated
}
return ret
}

4. Layered Clustering

This form of clustering, originally postulated by Jardine and Sibsone , involves the creation of hierarchies of sets involving a sequence of divisions of the original set that terminate with a set size greater than one and which typically involve divisions that are greater than binary. This a generalization of the 'Strong Connected Link Betweeness' graph clustering algorithm, implemented in Graph-RAT 0.5.1 [1] with two sub-languages for stop conditions and graph acceptance criteria.

5. Concept Pruning

Concept Pruning involves a layered clustering algorithm taking a set of concept descriptions and outputting a tree of sets that describes the core terms of a document. While this algorithm is initially created for use with disambiguated terms, the algorithm can also perform term disambiguation implicitly, trimming unneeded meanings, permitting multiple senses of a term to co-exist in a document.

The algorithm splits into two phases, the creation of the next generation of nodes and the decision on which of these new nodes are to be added to the parent set. The first iteratively removes a term from the set. If the result is more internally cohesive, the process is repeated with removing another term in a depth first manner. If the result is a local minimum, it is added to the set of potential sets. The list is pruned of all non-significant subsets. The end result is then pruned again with the greatest average relatedness member of not-significantly-different subsets made the parent node of a recursive call, then added to the parent and returned. The algorithm is at least $O(n^4)$, however the formal proof of time complexity has not been completed.

List{Node} Reduce (Node parent, Node current){
List{Node} ret
$\forall Termt \in current\{$
Node child = current - term
if(AverageRelated(child) < AverageRelated(current)){
List{Node} set = Reduce(parent,child)

[1] `http://graph-rat.sourceforge.net`

if(set = {}){
ret = ret ∪ term
}else{
ret = ret ∪ set
}
}
$\forall Noden \in ret\{$
if(T-Test(AverageRelated(parent),AverageRelated(child)) is not significant){
ret = ret - n
}
}
return ret
}
List{Node} SplitTerms(Node parent){
List{Node} children
List{Node} ret
ret = Reduce(parent,parent)
List{List{Node}} equivelance = GetSignificantlyDifferentNodes(children)
$\forall List < Node > nodeList \in equivelance\{$
double max = NegativeInfinity
Node representative
$\forall Nodeleaf \in nodeList\{$
if(AverageRelated(leaf) > max){
representative = leaf
max = AverageRelated(leaf)
}
}
representative.addAll(SplitTerms(representative,representative))
ret.add(representative)
}
return ret
}