Thesis

Ryan Tanner

February 12, 2012

Contents

1 Introduction

2 Problem Statement

This thesis is an attempt to tackle the problem of extracting facts and connections from written text. Massive quantities of text are produced daily and methods for quickly getting relevant information out of that text are needed. There are many

2.1 Extracting properties defined in a massively large body of text

2.2 Doing so in a time-efficient manner

2.3 On a textual level, the problem of finding connections across concepts and entities in a massive corpora of text

3 Other Approaches

3.1 Training oriented approaches

3.1.1 Manually-annotated training input

3.1.2 More accurate than my proposed solution

3.2 Weakly-linked crowd sourcing

4 Importance of the Problem

4.1 The Problem of Big Data

Google alone processes over twenty petabytes of data per day (**?**).

4.2 Tracing Influence

5 Approach to Solving the Problem

Most approaches to this problem rely on extracting as much information as possible from a given input. This approach comes at the problem from the opposite direction and tries to extract a little bit of information very quickly but over an extremely large input set.

5.1 Treating grammatical dependencies as functions

This approach is based on the premise that dependency grammar relations can be treated as functions and modeled as such. Furthermore, I hypothesize that these functions can be curried, just as in a functional language. Every word in a sentence, save for the head, is dependent upon another word and each of these dependencies has a type. This structure forms a tree. By doing a depth-first traversal of this tree and recursively composing each individual dependency function into a curried function, we end with a function specific to that sentence.

In this approach, dependency functions are short operations which extract properties from the given relation. These functions take two nodes of a tree as input, the governor and the dependent. Based on the types of the tokens in each node a partial or full property is added to the accumulator map and returned up the tree. This map is comprised of entities mapped to properties representing pieces of information extracted from the relationship. More about properties can be found in section **??** on page **??**.

5.2 Mapping the governors and dependents of those dependencies to textual aliases and named entities

5.3 Reducing a set of input documents to find connections between those aliases and entities based on their common properties

5.4 Constructing a graph of these connections where the connections form weighted vertices and entities form nodes

5.5 Visualizing this graph

5.6 Why a functional language?

6 Algorithm in Detail

6.1 Dependency Functions

The grammar dependencies used here are those described in the Stanford typed dependencies manual **?**. Currently 53 grammatical relations are defined for the English language. Each of these has a corresponding function in this algorithm. Though the specifics of each function differ, all follow the same simple pattern. Dependency functions take two parameters, a governor and a dependent, and return a map of tokens to a list of properties. Furthermore, these grammatical relations have a typed hierarchy where relations can inherit from other relations. Each function therefore can use its supertype's own function and only add the minimum processing necessary for its specific relationship.

These functions use the part-of-speech tags of the tokens in the governor and dependent to loosely determine if a full or partial property is defined by this relationship.

6.2 Properties

6.2.1 Full vs. Partial Properties

Given that a property may not be fully defined by a single grammatical relation, this solution provides for "partial properties" to be returned by dependency functions. This can be seen as a loose form of delayed message-passing. Dependency functions which begin a partial property add it to the map entry of the governor's token. Conversely, dependency functions which can use a partial property to create a full property check for the existence of such partial properties on in the map returned by child nodes. As we use a depth-first traversal child nodes are processed before parent nodes, allowing partial properties to be in effect passed from child to parent and then filtered by the parent before returning the result map.

7 Results

8 Future Recommendations

A Some Relevant NLP Concepts

A.1 Dependency Grammars

B Tools Used

C Code Highlights