Dependency Parsing

Shubham Gupta

January 11, 2020

1 Introduction

1.1 Phrase structure

- words
- phrases
- Bigger phrases(sentences lol)
- Also called Context Free Grammer(CFG). Building structure for the language using Noun(N), Verb(V), Verb Phrase(VP), etc.

1.2 Dependency structure

- Words depend on other words.
- Why do we need all this?
  - Understand language structure
  - Need to know wat is connected is to what.

2 Ambiguities

2.1 Prepositional phrase attachment ambiguity

- Occurs when you have PP before a Noun or Noun Phrase or a Verb.

2.2 Coordination phrase ambiguity

- Example: Doctor: No heart, cognitive issues.

2.3 Adjectival Modifier Ambiguity

- Example: Students get first hand job experience(looool)

2.4 Verb Phrase attachment ambiguity

- Example: Mutilated body washes up rio beach to be used for olympics

3 Dependency Grammar

- Syntactic structure between lexical terms (normally arrows called *depedencies*)
- Arrows have the relationship type between the two words. They connect *head* and the *dependant* of the dependency

4 Depedency conditioning preferences

- Bilexical affinities $\implies$ discussion to issues is possible
- Depedency distance $\implies$ mostly with nearby words
- Intervening material $\implies$
- Valency of heads

4.1 How to do it?

- Choose for each word what the other word(including the ROOT)is it a dependant of
- Only one word is a dependent of the ROOT
- No cycles i.e A -¿ B, B -¿ A

5 How to do dependency parsing

- **Dynamic Programming** $O(n^3)$
- **Graph algorithms** Create a minimum spanning tree
- **Constraint satisfaction** Apply constraints to edges. Maybe this could be done with constraint programming?
- **Transition-based or deterministic dependecy parsing**: Use greedy algo. Most used.

5.1 Greedy based depedency parsing

- Sequence of bottom-up actions
- Has the following parts:
  - stack σ written with top to the right. Starts with root symbol
  - buffer β written with top to the left. Starts with input sentence
  - Finish if $\sigma = \beta$
- For this algo, you would need to explore all possible paths before finding the optimal path. Not very efficient

5.2 MaltParser

- Use maseen learninggggg lol
- Use a discrimative classifier to predict next action to take.
  - Max 3 untyped choices: —R— x 2 + 1
  - Features: top of stack word, POS, first buffer word, POS
- Super fast linear parsing with mad performance
- Just below current SOTA

5.3 Evaluation of depedency parsers

- Compare with labelled data

6 Neural depedency parsers

- Problems with previous methods
  - Conventional method has sparse matrices
  - incomplete
  - Expensive
- Instead of specifying the sparse matrix, train neural network to learn the representation automatically
- Represent each word as a vector of dimension d
- Also label POS and dependency labels as vectors of dimension d
- greater accuracy and speed comparsed to maltparser