EVALUATION OF INTER-PARSER AGREEMENT

BIBLIOGRAPHY REPORT

Melpomeni Chioutakou Chioutakakou

Anahita Poulad

Eduardo Vallejo Arguinzoniz

Supervised by - Yannick Parmentier

MSc Natural Language Processing
M1 Supervised Project

IDMC
Academic year 2020-2021

Contents

Chapter 1

Introduction

Parsing of text is a prerequisite for many NLP tasks, such as sentiment analysis [Di Caro and Grella, 2013], machine-translation [Ding and Palmer, 2005] and information extraction [Yakushiji et al., 2006], to mention a few. Over the past decade, parsers have achieved very high performance levels in open-domain data, like the Penn Treebank. However, applying these parsers to real-world data, where assessing performance is complicated due to the lack of a test-set to test against, is a pressing problem. This issue is magnified when dealing with out-of-domain data for which parser performance drops substantially. The most common way to address the evaluation of unannotated data has been to manually annotate a small test-set to test the parser on. An approach that is very costly, as it must be performed for each domain separately and involves human labor. This is why a computerized method to gain meaningful insight about the reliability of a parser in real-world data is highly desirable.

The goal of this project is to explore the possibilities of automatic evaluation of parsers on unnanotated data. We will approach this problem by studying the behaviour of state-of-the-art parsers in out-of-domain data, and trying to correlate the agreement of different parsers with their respective performance. To achieve that, we will look at different tree alignment and parse matching functions and see how these metrics correspond with the actual performance of the parsers.

This report documents the body of knowledge acquired on the field during the inception phase of the project and proceeds to the formalization of an action plan for the following development stages. The rest of this report is organized as follows. Chapter 2 provides a background on Dependency Parsing and its core elements. This is followed by Chapter 3 which focuses on data-driven parsing approaches and techniques. Chapter 4 discusses the challenges of parser evaluation especially on out-of-domain data. Finally, Chapter 5 presents the conclusion and the action plan that will be carried out during the following months.

Chapter 2

Dependency Parsing

In recent years, the field of natural language processing has seen an increasing popularity of dependency-based methods for syntactic parsing. Dependency parsing is an approach inspired by the linguistic theory of dependency grammar, and which performs automatic syntactic analysis of natural language.

There are many reasons why these methods become more and more popular. One reason that justifies this popularity is the fact that dependency-based syntactic representations appear to be useful when it comes to language technology applications, such as machine translation and information extraction, as a result of their transparent encoding of predicate-argument structure. Moreover, they seem to be reliable for a wide range of languages that are typologically different and most importantly, this approach combined with machine learning from syntactically annotated corpora has allowed the development of accurate syntactic parsers.

There are different methods for dealing with dependency parsing. Generally speaking, we can divide these approaches into two classes, knowledge-based and data-driven dependency parsing. Knowledge-based approaches rely on a formal description of the dependency relations within words (so-called grammar), while data-driven ones use annotated data and machine learning algorithms to parse new sentences. We must note however, that it is possible for a parsing method to use both machine learning and formal grammar, making it data-driven and grammar-based at the same time [Kubler et al., 2009].

In the next sections, we will present the notion of dependency grammar that constitutes the basis of dependency parsing, we will formally represent dependency graphs, discuss projectivity and conclude this chapter by introducing annotated data (so-called treebanks), which are used by data-driven dependency parsers.

2.1 Dependency Grammar

Dependency grammars constitute a family of grammar formalisms, used by many current approaches in speech and language processing. In Dependency grammar, a sentence's syntactic structure is described with regard to the words or lemmas in that sentence and the set of directed binary grammatical relations that hold among them. More specifically, the linguistic units, i.e. words, are associated with directed links, the main predicate (verb) becomes the root of the clause structure and every other syntactic unit is connected to the verb with directed links. These links (syntactic units) are called dependencies.

As an illustration, consider Figure 2.1 page 6, where relations are represented by directed, labeled arcs above the sentences, and which connect heads to dependents. This dependency graph serves as a portrayal of each word in a sentence along with its modifiers.

Dependency grammars have been used in particular to describe morphologically rich languages that have a relatively free word order. For example, when dealing with a language whose word order is relatively flexible, such as Czech, where one can observe for instance the occurrence of a grammatical object before or after a location adverbial, a dependency-based approach would just need one link type to represent the particular adverbial relations that occur. This makes it a convenient formalism for a large number of languages, sometimes at the cost of non-projective dependency arcs, that will be discussed later in the chapter.

Moreover, Dependency grammars not only enable us to identify the head-dependent pairs but also to further classify the kinds of grammatical relations or grammatical function, regarding the dependent's role in the sentence with respect to its head. Such relations are familiar notions like subject and direct or indirect object. While in English word order is generally fixed and these notions do not determine the position in a sentence and the constituent type, in more flexible languages where phrase-based constituent syntax does not provide much help, the information that is directly encoded in the aforementioned grammatical relations is of critical importance.

2.2 Formal Representation

Dependency graphs are syntactic structures over sentences. A sentence is denoted by: S= $w_0, w_1 \dots w_n$, with w_0 being the ROOT and each token w_i representing a word.
Let R= $r_1, \dots\dots, r_m$ be a set of permissible arc labels.
Formally, a dependency graph for an input sentence is a labeled directed graph G=(V, A) that consists of a set of nodes V and a set of labeled arcs A, such that for a sentence S= $w_0, w_1 \dots w_n$

and label set R we have:

- V ⊆ $w_0, w_1 \dots w_n$
- A ⊆ V x R x V
- If $(w_i, r, w_j) \in A$ then $(w_i, r', w_j) \notin A$ for all $r' \neq r$

An arc (w_i, r, w_j) A represents a dependency relation from head w_i to dependent w_j labeled with relation type r.

We call dependency trees any well-formed dependency graph G= (V, A) for an input sentence S and dependency relation set R that is a directed tree that originates out of node w_o and has the spanning node set V= Vs.

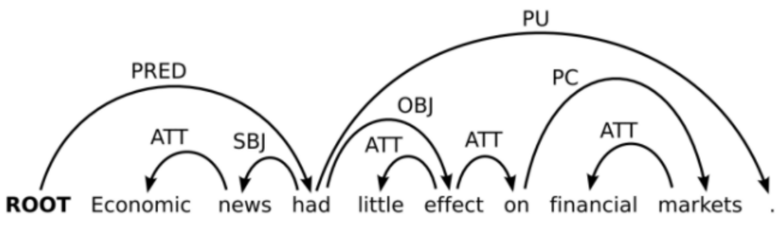

Figure 2.1: Dependency Structure for an English sentence.
[Kubler et al., 2009]

The dependency graph In 2.1 is represented by:

- G = (V, A)
- V= Vs = ROOT, Economic, news, had, little, effect, on, financial, markets,.
- A= (ROOT, PRED, had), (had, SBJ, news), (had,OBJ, effect), (had, PU,.), (news,ATT, Economic), (effect, ATT, little), (effect, ATT, on), (on, PC, markets), (markets, ATT, financial)

2.3 Dependency Trees

More specifically, a dependency tree constitutes a directed graph with the following features:

- There is a single designated root node with no incoming arcs.
- Each vertex has only one incoming arc, with the exception of the root node.

- There is a unique path from the root node to each vertex.

These features ensure that each word will have a single head, that the dependency structure is connected and that there is one and only root node that presents the directed path to every one of the words in the sentence. Furthermore, each word is either modified or modifies another word. The only word that is solely modified and does not modify anything else is the root of the tree [Jurafsky, 2009].

2.4 Projectivity

A dependency tree is considered projective if all the arcs that make it up are projective, meaning that there is a path that connects the head to every word that is found between the head and the dependent in the sentence. However, in languages that present a relatively flexible word order, we observe many valid constructions that lead to non-projective trees. Overall, we know if we are dealing with projectivity or non-projectivity depending on how we are drawing the trees. A dependency tree is projective if there are no crossing edges in its representation [Jurafsky, 2009].

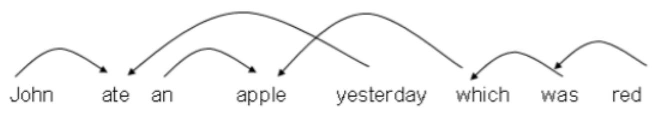

Figure 2.2: An example of a non-projective dependency tree
[Ambati, 2008]

Every sentence in natural language can be analyzed by a division of the dependency tree structures in projective or non-projective trees [Ambati, 2008].

Regarding the English language, the majority of the parse trees are projective but in certain cases, a non-projective tree is preferred. In the sentence “John ate an apple yesterday which was red”, the relative clause “which was red” is separated from the object “an apple” that it modifies by “yesterday”, a modifier of the main verb. In this case, we cannot draw a dependency tree that doesn’t have crossing edges [Ambati, 2008]. This distinction is very important in the research focused on parsing, since the algorithms change based on the kind of the tree we are dealing with [McDonald et al., 2005].

The transition-based parsing approach that is described in the following chapter only produces projective trees in its classical formulation and will therefore contain some errors when dealing with sentences with non-projective structures. This is one of the reasons motivating the more flexible graph-based parsing approach [Jurafsky, 2009].

2.5 Data-driven dependency parsing

This report focuses on the data-driven dependency parsing paradigm. As opposed to grammar-based dependency parsing, data-driven approaches learn to produce dependency graphs from an annotated corpus. One advantage these models present, is that they can be used in any domain that has annotated resources [McDonald and Nivre, 2007].

2.5.1 Dependency treebanks

When it comes to the development and evaluation of such dependency parsers, dependency treebanks are of great importance. They have been created by human annotators that described dependency structures for a particular corpus directly, or via automatic transformation from phrase-structure treebanks [Jurafsky, 2009]. The most well-known such treebank is the Penn Treebank for English [Marcus et al., 1993].

2.5.2 Universal Dependencies

Linguists have succeeded in developing dependency relations that extend further than the familiar notions of subject and object. The Universal Dependencies project [Nivre et al., 2016], provides a list of dependency relations that are linguistically-oriented, computationally useful and cross-linguistically applicable.

Frequently used relations are clausal relations used to describe syntactic roles based on a predicate (usually a verb) and modifier relations used to categorize the ways words modify their heads [Jurafsky, 2009].

Chapter 3

Machine learning approaches for dependency parsing

In this chapter we will talk about ways to tackle the dependency parsing task with a data-driven perspective by using Machine Learning (ML) techniques. We will see the two varieties of algorithms that are most widely recognized by the community: transition-based and graph-based methods. Finally, we will give a short-glimpse of the state-of-the-art in dependency parsing, which mostly boils down to how the deep-learning revolution got integrated into the already existing frameworks.

3.1 Transition-based and graph-based approaches

There are countless ways of approaching dependency parsing, but as the field got more mature most methods started to converge into two varieties of techniques: transition-based and graph-based parsers [Jurafsky, 2009]. In this section we will give a high-level description of these two approaches. The two approaches make use of ML techniques to implement some complex aspect of the algorithm (like decision oracles and dependency arc probabilistic models). For the purpose of brevity we will abstract the specific ML algorithms, and assume that some classical feature based ML technique is used.

3.1.1 Transition-based systems

Transition-based parsers take inspiration from the bottom-up parsers that are common in compilation techniques for programming languages [Aho and Ullman, 1972].

Programming languages are formal languages and are constructed grammar-first instead of the grammar naturally arising as in natural language. The grammars of formal languages are usually constructed to be "unambiguous" and "context-free", which restricts the kinds of rules and productions that can be defined [Aho et al., 1986]. These restrictions are what allows the text to be parsed into the grammar by a computational model as "simple" as *deterministic pushdown automata* [Hopcroft et al., 2001].

A pushdown automaton is just a finite-state automaton that has been augmented with a stack. At each transition the pushdown automaton reads the next item from the input (same as a finite-state automaton), but it also reads the top of the stack and may push or pop a constant amount of symbols to the stack, depending on the read symbols and current state. The behaviour of a pushdown automaton is defined by its transition function. A transition function

$$\delta : Q \times \Sigma \times \Gamma \to Q \times \Gamma^*$$

is an application which maps a combination of an automaton state $q \in Q$, tape symbol $\sigma \in \Sigma$ and stack state $\gamma \in \Gamma$ to a new automaton state $q' \in Q$ and a sequence of symbols to push or pop to the stack $(\gamma_1, \gamma_2, ..., \gamma_n) \in \Gamma^*$. In the case of formal languages the transition function of the pushdown automata is completely defined by the grammar of the formal language.

However, if we want to use this same proven method for natural language text, we are going to need to make some adjustments.

Firstly, we have to assume that the underlying natural language grammar is "context-free". This might seem as too strong an assumption but it turns out that natural language grammar mostly follows context-free patterns. The parses of context-free languages have the property of projectivity, and although dependency parses mostly satisfy this property, it is not always the case. Hopefully there are ways to bypass this limitation, albeit at the cost of computational complexity.

Secondly, we need to think about a transition function for the parser pushdown automaton. The naive solution would be to write a context-free grammar for the language we want to parse, and apply the same technique as in formal languages. Nevertheless, this is highly impractical to do for a natural language, which is huge compared to man-made formal languages.

Since defining a transition function is a very complex task, the common theme is to learn the function using some ML technique. As in the case of these kinds of data-driven techniques, it is necessary to collect a big amount of data to learn the algorithm. To learn this transition function we need examples of the parsing state of a pushdown automaton paired with the action the automaton should take. Although there is no data in such format readily available, it is fairly easy to generate by taking a dependency tree annotated corpus, and automatically parsing it using the reference tree while you annotate the transitions.

A parsing state of a pushdown automaton consists of the contents of the stack (sequence of words), the symbol (word) that is being read from tape and the state of the automaton. There are many ways to encode this information to make it usable for a ML algorithm. The classical way is to represent words as vectors of binary and categorical features (that each word has). The whole state is then represented as a sequence of these vectors.

The idea of action is often simplified from the traditional automata theory notion, instead of describing a transition of the automaton state and an arbitrary modification to the stack, there are three actions that the automaton can do: *SHIFT*, *RIGHTARC* and *LEFTARC*

- SHIFT: removes the next word from the input tape and pushes it onto the stack
- RIGHTARC: produces an arc from the second (from the top) word in the stack to the top word, removes the top word (the dependent)
- LEFTARC: produces an arc from the top word in the stack to the second (from the top) word, removes the second word (the dependent)

This set of operators implements what is known as the **arc-standard** approach to transition-based parsing [Covington, 2001].

To produce labels for the arcs the last two operations must be parametrized with a dependency type label, which grows the number of effective actions considerably. See Figure 3.1 for an unwrapped view of the parsing process.

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	

Figure 3.1: Detailed trace of the parsing process of a transition-based system.

If we generate the appropriate data to train the ML oracle (transition function), and the training is successful, then it is just a matter of simulating a pushdown automaton with the trained ML algorithm as the transition function to have a working parser, see Figure 3.2.

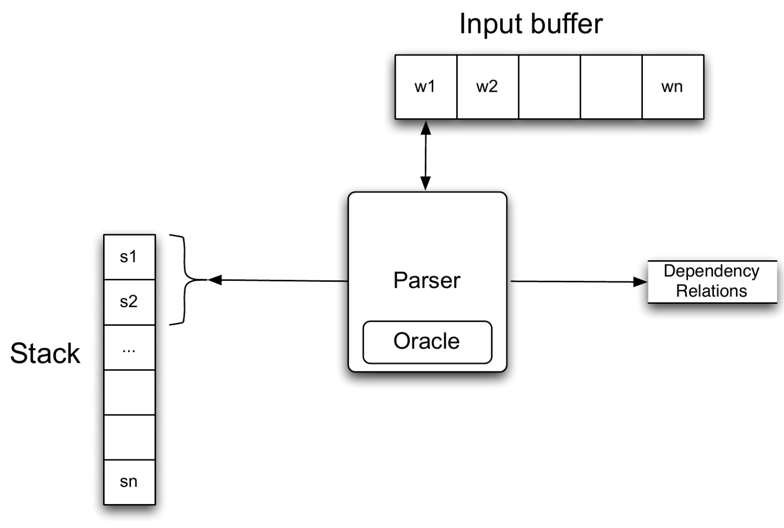

Figure 3.2: Diagram of the operational view of a transition-based parser. It analyzes the next word in the input buffer and the stack, and the oracle decides the action to take.

Before closing this section, let's take this chance to talk about some of the strengths and problems of this approach. To begin with, it is important to appreciate the low resource usage of this method, as it stands, it is linear $O(n)$ in both space usage and time complexity. This is partly because of the greedy nature of this method (i.e. it considers a unique local decision at each step and discards the rest), however this also plays against it. The first problem the greedy aspect arises is that the method only produces projective parses, which greatly limits the performance headroom of the method as it can't get non-projective cases right. There are ways to solve this in exchange for a performance penalty, which keeps the revised methods' time complexity bound at a quadratic level $O(n^2)$ [Nivre, 2009].

The second problem is that the method lacks a global view of the parse. It only operates locally in a decision by decision basis and cannot go back to fix a previous bad decision. This effect can be mitigated somewhat by allowing the parser to keep a set of decision sequences (also called a "beam"), instead of just keeping the decision sequence of the locally best decisions. For this to work, we need to change the oracle, so that it scores all the possible local decisions instead of returning the best ones. With this kind of oracle the procedure would consist in producing all possible next decision sequences given a beam of previous decision sequences, ranking the new sequences according to an aggregate function of all local decisions, trimming the amount of decisions in the beam to a pre-established size K and repeating the procedure until all sequences in the beam are final. This search algorithm is known as *K-best Beam Search* [Russell and Norvig, 2002].

3.1.2 Graph-based systems

Graph-based methods work by first defining a fully-connected directed graph $G(V, E)$, where the vertices V are the words of the sentence to parse and a special "root" vertex; and the edges E are all the possible dependency relations between each word. There is also an edge going from the root to all the words. This graph now contains all the possible dependency trees as subsets of the set of edges, so the current objective is to select the edges from the graph to obtain the subset of edges E' which represents the correct dependency tree.

Not any subset will do though, the subset of edges we select must constitute a valid dependency tree, so we will need the following properties:

1. All vertices (but the root, which has none) must have a single incoming edge $\forall v \in (V - \{v_{root}\})\ deg^{-}(v) = 1$ and $deg^{-}(v_{root}) = 0$
2. There are no cycles

These two properties suffice to ensure that the subgraph is a tree rooted at the "root" vertex, hence all vertices are reachable from the root. It also fixes the amount of edges to $|E'| = |V| - 1$. This kind of tree where all vertices of a graph are reachable from a given root is called a spanning tree of the graph. A graph may have more than one spanning tree, and this graph we defined happens to have a combinatorial amount of them. In fact, each spanning tree corresponds to a dependency tree, so there are as many spanning trees as ways to parse a sentence into a dependency tree.

We are now in need of some criteria to select a single spanning tree as the prediction of the system. With the current setup there is nothing special about some edges over others. In order to measure the quality of a spanning tree, we are going to need to take into account the quality of individual selections of edges and aggregate the scores into an overall score for the whole spanning tree. To do this, we will need to weight each arc with a score that represents how likely a dependency is from the source word to the dependent word. As already mentioned, this a very complex task by itself, so we will turn once again to data-driven ML techniques to learn an approximator for these scores. A example is shown in Figure 3.3

The ML system is very simple in this case. It just needs to regress a numeric value (a probability or log-probability) given the two words in the dependency relation, and the label of the dependency relation. As with the ML methods for transition based systems, the encoding for the words is a topic of its own, but has been classically approached by feature vector representations.

The data we need to learn this function consists in the true weighted graphs, which we can construct by setting the weight to 1 for edges that are in the ground-truth dependency

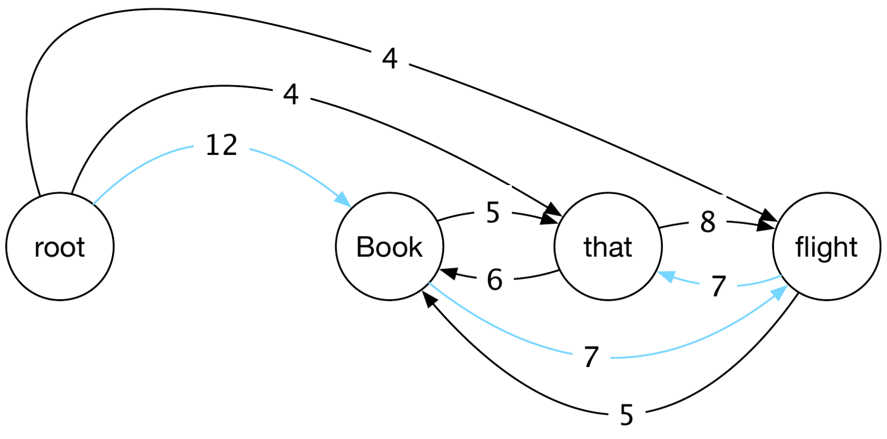

Figure 3.3: Directed weighted graph for the sentence "Book that flight".

trees and 0 for all other edges. Once we have trained our scoring algorithm we can predict weights for all edges in the graph.

Now we need a method to select a spanning tree given the weights of individual edges in the graph. One intuitive way of going about this, is to try to maximize the sum of scores of all edges in the spanning tree. This is a typical problem in graph algorithms. A spanning tree that maximizes the sum of the weights of the edges that constitute it, is called the Maximum Spanning Tree (MST) of a graph. There are very efficient algorithms already developed to tackle this problem, that can solve it in as little as $O(|E|+|V|\log|V|)$ [Gabow et al., 1986], and since the graph is fully connected $|E| \in O(|V|^2)$, the method is quadratic $O(n^2)$ in the number of words $n = |V|$.

Comparing it with the transition based method, this method is both slower and takes more memory ($O(|E|+|V|) = O(n^2)$ at least to store the graph). However it is not a completely fair comparison since this method produces non-projective parses without any modification. If compared with the transition-based methods that can produce non-projective parses, the asymptotic complexities are both quadratic.

In terms of performance, graph-based methods have empirically shown better performance in longer sentences where transition-based methods struggle. Conversely they are usually outperformed in shorter sentences [McDonald and Nivre, 2011]. This difference is attributed to the mismatch between the local and fine-grained nature of the transition-based systems and the high-level coarse-grain view of the solution space of the graph-based method. According to this hypothesis, longer sentences are problematic for transition-based systems because there are more chances for a bad transition to be selected by the oracle, and once a bad action is taken the whole parse process that follows is ill-conditioned; graph methods, on the other hand, do not propagate errors as the parsing process is not sequential, and a small parsing mistake in one place may not have such an impact in the rest of the parse.

3.2 State-of-the-art

In the last decade all the fields of AI have seen a huge progress. This is significantly due to the raise of deep learning, which has allowed to push the performance of the state-of-the-art for many AI tasks, including dependency parsing.

Deep learning refers to a family of machine learning methods that have to do with artificial neural networks and representation learning [Goodfellow et al., 2016]. What sets deep learning apart from previous machine learning methods is that representations are learned directly by the algorithm, bypassing the need to manually test different combinations of feature based representations. Since there is no need to do feature-engineering, the development time for this kind of algorithms is greatly reduced and the need of field experts to do this work is depreciated. Also, features are fixed unlike representations in deep learning, which can adapt during the training process. Because of this deep learning algorithms tend to outperform feature based machine learning algorithms. This is not always the case however, as deep learning techniques are known to need huge amounts of data to form good representations, whereas feature based algorithms are not so data-hungry.

In the case of dependency parsing, the two prevailing methods got revised with deep learning approaches. There are two works that have set the tone for the adoption of deep learning in the field: the one by [Ma et al., 2018] about the stack-pointer architecture for dependency parsing, which implements a deep-learning version of a transition-based dependency parsing system; and the one by [Dozat et al., 2017] about a Graph-based neural dependency parsing, which infuses graph-based methods with deep learning approaches. More recently, the use of pretrained neural language models, such as BERT [Devlin et al., 2019], have enabled additional performance gains and have lowered the training costs significantly. These language models are trained with a very large amount of unannotated text (e.g. the whole Wikipedia) through a self-supervised learning process. In the end these systems are able to produce very rich representations of text that can be used to warm start a deep learning algorithm.

As we have seen, there are various approaches to dependency parsing. In the next chapter, we will discuss parser evaluation, with a focus on out-of-domain data (that is, parsing of data that significantly differ from the data used to train the parser).

Chapter 4

Evaluation of dependency parsing

In this chapter we talk about dependency parser evaluation approaches. We will first discuss how in an ideal setting, a parser's output is evaluated against one or more reference gold standard treebanks. As it is not always the case, we will then contemplate reliability as a challenging aspect of evaluation. Inter Annotator Agreement (IAA) concept and metrics are introduced as a way of tackling the difficulties we might face especially when evaluating parser performance on out of domain data. Different IAA metrics and underlying assumptions are reviewed. Finally, we try to somehow map the idea to the problem of evaluating dependency parser, based on the suggested solutions.

4.1 Dependency Parser Evaluation

The most conventional evaluation of dependency parsers is based on measuring how well they work on a train/dev/test split of the gold standard treebank, meaning how the output parses match the corresponding ones in the gold standard data. Therefore, a dependency parser, could then be evaluated by checking the parsing of the test set against the gold standard annotation of the test set found in the reference treebanks. Many metrics have been used for dependency parser evaluation. Some of them are the following:

4.1.1 Exact Match

This metric represents the percentage of completely matching parsed sentences. This might not be the best choice to measure the parser, as a large number of sentences could be labeled wrong. Although, some of them contain just small deviations from the gold standard data.

4.1.2 Attachment Score

The most common method for evaluating dependency parsers are labeled and unlabeled attachment accuracy. Labeled attachment is the correct assignment of a word to its head along with the correct dependency relation. Unlabeled attachment on the other hand, looks at the correctness of the assigned head, ignoring the dependency relation. Having a parser output and a reference parse, accuracy can be calculated using the percentage of words in the input that are assigned the correct heads with the correct relations. These metrics are usually referred to as the labeled attachment score (LAS) and unlabeled attachment score (UAS).

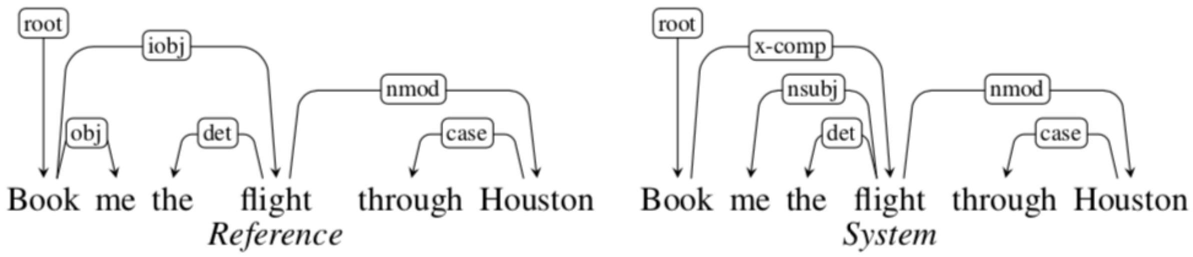
root
iobj
obj
det
nmod
case
Book me the flight through Houston
Reference
root
x-comp
nsubj
det
nmod
case
Book me the flight through Houston
System

Figure 4.1: Comparison of the gold standard parse and the parser result. [Jurafsky, 2009]

In figure 4.1 the total Head-Dependent pairs in 'Reference'/Ground Truth is 6. Total Head-Dependent pairs correctly detected with correct tags is 4 (Book → flight and flight → me are not accounted). Labeled Attachment Score (LAS): The ratio of correctly detected Head-Dependent pairs along with their tag/ total Head-Dependent pairs in the ground truth is $\frac{4}{6}$= 0.666. Total Head-Dependent pairs correctly detected is 5 (irrespective of the tag, only flight → me is a wrong pair and is not counted). Unlabeled Attachment Score (UAS): The ratio of correctly detected Head-Dependent pairs (irrespective of the tag)/ total Head-Dependent pairs in ground truth is $\frac{5}{6}$ = 0.833.

4.2 Parser Evaluation on Out-of-Domain Un-Annotated Data

In parsing literature, it is common to see parsers trained and tested on the same textual domain. Unfortunately, the performance of these systems degrades on sentences drawn from a different domain [McClosky et al., 2010]. Hence, when dealing with out-of-domain data, evaluation of parser's performance seems necessary. This evaluation might turn into a bigger problem once the out of domain data is not annotated and there is no gold standard parse tree available.

Proposed solutions for such a problem would be to either manually annotate test data from the new domain, or to conduct a reliability study. While the former is expensive and not very elegant, the later is based on the assumption that annotation is reliable if parsers seem to agree on the dependencies assigned [Krippendorff, 2004, Craggs and Wood, 2005]. If different parsers produce similar results, then we can infer that they have internalized a similar understanding of the annotation guidelines, and we can expect them to perform consistently under this understanding. Reliability is thus a prerequisite for demonstrating the validity of the parsing [Krippendorff, 2004].

4.3 Inter-Annotator Agreement

The simplest measure for gauging the agreement between two annotators is percentage of observed agreement, defined as "the percentage of judgments on which the two analysts agree when annotating the same data independently" [Scott, 1955]. This is the number of annotations on which the annotators agree divided by the total number of annotations.

Observed agreement enters in the computation of all the measures of agreement we consider, but it does not yield values that can be compared on its own, since some agreement is chance-based, and the amount of chance agreement is affected by two factors. First of all, "[percentage agreement] is biased in favor of dimensions with a small number of categories" [Scott, 1955]. In other words, given two annotation schemes for the same phenomenon, the one with fewer categories will result in higher percentage agreement just by chance.

The second reason why percentage agreement cannot be trusted is that it does not correct for the distribution of items among categories: We expect a higher percentage agreement when one category is much more common than the other [Hsu and Field, 2003]. The conclusion reached in the literature is that in order to get figures that are comparable, observed agreement has to be adjusted for chance agreement [Krippendorff, 2004].

4.3.1 Chance-corrected IAA

All of the coefficients of agreements, correct for chance on the basis of the same idea. First, we find how much agreement is expected by chance: Let us call this value A_e. The value $1 - A_e$ will then measure how much agreement over and above chance is attainable; the value $A_o - A_e$ will tell us how much agreement beyond chance was actually found. The ratio between $A_o - A_e$ and $1 - A_e$ will then tell us which proportion of the possible agreement beyond chance was actually observed. This idea is expressed by the following formula.

$$S, \pi, \kappa = \frac{A_o - A_e}{1 - A_e}$$

The three best-known coefficients, S [Bennett et al., 1954], π [Scott, 1955], and κ [Cohen, 1960], and their generalizations, all use this formula. All three coefficients therefore yield values of agreement between $\frac{-A_e}{1-A_e}$ (no observed agreement) and 1 (observed agreement = 1), with the value 0 signifying chance agreement (observed agreement = expected agreement). Note also, that whenever agreement is less than perfect ($A_o < 1$), chance-corrected agreement will be strictly lower than observed agreement, because some amount of agreement is always expected by chance.

All three coefficients assume independence of the annotators.That is, that the chance of c_1 and c_2 agreeing on any given category k is the product of the chance of each of them assigning an item to that category: $P(k|c_1) \cdot P(k|c_2)$. Expected agreement is then the probability of c_1 and c_2 agreeing on any category, that is, the sum of the product over all categories:

$$A_e^S = A_e^\pi = A_e^\kappa = \sum_{k \in K} P(k|c_1) \cdot P(k|c_2)$$

The difference between S, π, and κ lies in the assumptions leading to the calculation of $P(k|c_i)$, the chance that annotator c_i will assign an arbitrary item to category k [Zwick, 1988, Hsu and Field, 2003].

S: This coefficient is based on the assumption that if annotators were operating by chance alone, we would get a uniform distribution: That is, for any two annotators c_m, c_n and any two categories k_j, k_l, $P(k_j|c_m) = P(k_l|c_n)$.

π: If annotators were operating by chance alone, we would get the same distribution for each annotator: For any two annotators c_m, c_n and any category k, $P(k|c_m) = P(k|c_n)$.

κ: If annotators were operating by chance alone, we would get a separate distribution for each annotator [Krippendorff, 2004].

4.3.1.1 Krippendorff's α - Weighted Agreement Coefficient

Any general similarity metric is necessarily going to suffer from the issue that all parts of the syntactic annotation are not equally important: whether an interjection was misattached in the tree is much less important than if it was the subject of the sentence. Therefore any such similarity metric cannot reliably answer the question how good or bad the result actually is, only how much it is structurally different. [Skjærholt, 2014] Krippendorff's α is not as commonly used as κ and π, but it has the advantage of being expressed in terms of an arbitrary distance function δ. Krippendorff's α is normally expressed in terms of the ratio of observed and expected disagreements: $\alpha = 1 - \frac{D_o}{D_e}$, where D_o is the mean squared distance between annotations of the same item and D_e the mean squared distance between all pairs of annotations:

$$D_o = \sum_{i \in I} \frac{1}{|X_i| - 1} \sum_{c \in C} \sum_{c' \in C} \delta(x_{ic}, x_{ic'})2$$

$$D_e = \frac{1}{\sum_{i \in I} |X_i| - 1} \sum_{i \in I} \sum_{c \in C} \sum_{i' \in I} \sum_{c' \in C} \delta(x_{ic}, x_{i'c'})2$$

Here the function δ can be any metric. function.[Kripendorff, 2004]

4.3.2 Chance-Corrected IAA for Dependency Parsers

There are a lot of IAA metrics for many different kinds of annotation, but no such measure is in widespread use for the task of syntactic annotation.

Most IAA metrics are defined under the assumption that annotations contain little to no internal structure (such as in a label). In fields like syntactic parsing however, where the annotations are trees, structure is the main point of annotation. In this case, the metrics have to be adapted to effectively deal with structure. There are two prevailing methods for IAA: adapted F-score and accuracy metrics, and similarity measure specific for dependency trees(LAS). The former has the issue of being biased in favour of annotation schemes with fewer categories and not accounting for skewed distributions between classes, and the latter is not chance-corrected which has some limitations as pointed out earlier [Skjærholt, 2014].

Instead [Skjærholt, 2014] suggests to use a method based on Krippendorff's α, that works on any items for which a distance function (that defines a valid metric-space) can be defined. The annotation items in syntactic parsing are trees, so a valid distance function must be defined that works with trees. Some works have used the Tree Edit Distance (TED) metric, which is a generalization of the String Edit Distance.

As we just saw, there are various ways of dealing with disagreeing annotations (e.g. dependency parses). What we are looking for is a mean to assess how reliable a dependency parse is, according to its similarity with parses produced by other parsers on the same input sentence. In this context, some of the metrics above may prove useful.

In the next section, we will conclude by sketching an action plan, which aims at implementing an experimental inter-parser agreement evaluation.

Chapter 5

Conclusion and Action Plan

In previous chapters we reviewed the concept of dependency parsing and went through some modern approaches.

Elaborating on the data-driven parsing idea, we discussed the most common methods used in the parsers that are based on machine learning algorithms. We also introduced the state-of-the-art deep-learning parsing. In order to address the parser evaluation on out of domain data, we took a closer look at the parser evaluation process, and how the lack of gold standard data might influence it. The concept and metrics of Inter Annotator Agreement were discussed for the evaluation. Different approaches and methods are offered in articles to measure this agreement as robustly as possible. Lastly, we have reviewed a number of suggestions related to out-of-domain dependency parsing (see Appendix A) and we will start the implementation phase based on the following timeline.

5.1 Data Collection

During the first month we will collect a collection of publicly available data (e.g. Wikipedia), preferably in French.

5.2 Parsing experiments and Annotation

During the second month we will install a number of relevant (in terms of reliability and performance) parsers in order to generate different annotations over the data collected in the previous section.

5.3 Parser Agreement Appraisal

During the third month we will use the annotations generated in 5.2 and evaluate the agreement of the different parsers that we tested over the data set acquired at the first step.

5.4 Structural Adjustment and Re-Inspection

During the fourth month we will implement a suitable structural alignment method and conduct an agreement study, showing how the new calculated agreement is different from the un-adjusted version and how well it represents the actual similarity between the annotations.

5.5 Documentation

The last step will consist in the analysis of the final results, the documentation of the final report's details and the defense.

Bibliography

[Aho et al., 1986] Aho, A. V., Sethi, R., and Ullman, J. D. (1986). Compilers, principles, techniques. *Addison wesley*, 7(8):9.

[Aho and Ullman, 1972] Aho, A. V. and Ullman, J. D. (1972). *The Theory of Parsing, Translation, and Compiling*. Prentice-Hall, Inc., USA.

[Ambati, 2008] Ambati, V. (2008). Dependency structure trees in syntax based machine translation.

[Bennett et al., 1954] Bennett, E. M., Alpert, R., and Goldstein, A. (1954). Communications through limited-response questioning. *Public Opinion Quarterly*, 18(3):303–308.

[Cohen, 1960] Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46.

[Covington, 2001] Covington, M. A. (2001). A fundamental algorithm for dependency parsing. In *Proceedings of the 39th annual ACM southeast conference*, pages 95–102. Citeseer.

[Craggs and Wood, 2005] Craggs, R. and Wood, M. M. (2005). Evaluating discourse and dialogue coding schemes. *Computational Linguistics*, 31(3):289–296.

[Devlin et al., 2019] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding.

[Di Caro and Grella, 2013] Di Caro, L. and Grella, M. (2013). Sentiment analysis via dependency parsing. *Computer Standards & Interfaces*, 35(5):442–453.

[Ding and Palmer, 2005] Ding, Y. and Palmer, M. (2005). Machine translation using probabilistic synchronous dependency insertion grammars. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 541–548.

[Dozat et al., 2017] Dozat, T., Qi, P., and Manning, C. D. (2017). Stanford's graph-based neural dependency parser at the conll 2017 shared task. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 20–30.

[Gabow et al., 1986] Gabow, H., Galil, Z., Spencer, T., and Tarjan, R. (1986). Efficient algorithms for finding minimum spanning tree in undirected and directed graphs. *Combinatorica*, 6:109–122.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

[Hopcroft et al., 2001] Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2001). Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1):60–65.

[Hsu and Field, 2003] Hsu, L. M. and Field, R. (2003). Interrater agreement measures: Comments on kappan, cohen's kappa, scott's π, and aickin's α. *Understanding Statistics*, 2(3):205–219.

[Jurafsky, 2009] Jurafsky, H. M. (2009). Speech and language processing daniel jurafsky and james h. martin (stanford university and university of colorado at boulder) pearson prentice hall, 2009, xxxi+ 988 pp; hardbound, isbn 978-0-13-187321-6, 115.00.

[Kripendorff, 2004] Kripendorff, K. (2004). Content analysis: An introduction to its methodology.

[Krippendorff, 2004] Krippendorff, K. (2004). Reliability in content analysis: Some common misconceptions and recommendations. *Human communication research*, 30(3):411–433.

[Kubler et al., 2009] Kubler, S., McDonald, R., and Nivre, J. (2009). *Dependency Parsing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

[Ma et al., 2018] Ma, X., Hu, Z., Liu, J., Peng, N., Neubig, G., and Hovy, E. (2018). Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1403–1414, Melbourne, Australia. Association for Computational Linguistics.

[Marcus et al., 1993] Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

[McClosky et al., 2010] McClosky, D., Charniak, E., and Johnson, M. (2010). Automatic domain adaptation for parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 28–36. Association for Computational Linguistics.

[McDonald et al., 2005] McDonald, R., Crammer, K., and Pereira, F. (2005). Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 91–98, Ann Arbor, Michigan. Association for Computational Linguistics.

[McDonald and Nivre, 2007] McDonald, R. and Nivre, J. (2007). Characterizing the errors of data-driven dependency parsing models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 122–131, Prague, Czech Republic. Association for Computational Linguistics.

[McDonald and Nivre, 2011] McDonald, R. and Nivre, J. (2011). Analyzing and integrating dependency parsers. *Computational Linguistics*, 37(1):197–230.

[Nivre, 2009] Nivre, J. (2009). Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359, Suntec, Singapore. Association for Computational Linguistics.

[Nivre et al., 2016] Nivre, J., de Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajič, J., Manning, C. D., McDonald, R., Petrov, S., Pyysalo, S., Silveira, N., Tsarfaty, R., and Zeman, D. (2016). Universal Dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1659–1666, Portorož, Slovenia. European Language Resources Association (ELRA).

[Russell and Norvig, 2002] Russell, S. and Norvig, P. (2002). Artificial intelligence: a modern approach.

[Scott, 1955] Scott, W. A. (1955). Reliability of content analysis: The case of nominal scale coding. *Public opinion quarterly*, pages 321–325.

[Skjærholt, 2014] Skjærholt, A. (2014). A chance-corrected measure of inter-annotator agreement for syntax. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 934–944.

[Yakushiji et al., 2006] Yakushiji, A., Miyao, Y., Ohta, T., Tateisi, Y., and Tsujii, J. (2006). Automatic construction of predicate-argument structure patterns for biomedical information extraction. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 284–292.

[Zwick, 1988] Zwick, R. (1988). Another look at interrater agreement. *Psychological Bulletin*, 103(3):374.

Appendix A

Paper summaries

Monolingual Phrase Alignment on Parse Forests

This paper proposes a method to conduct "phrase alignment" on parse trees (and an extension to forests), with the particular application in "paraphrase detection". The proposed method stands out by making use of linguistically motivated grammars to identify "syntactic paraphrases" and because of it being able to align non "non-homographic" phrases. The method shows good performance when compared to the human baseline.

The algorithm has two design aspects:

- A probabilistic aspect where the goal is to estimate the probability of a phrase alignment as supported by some already aligned phrases. In this method this probability estimation is implemented as an expression (softmax) with learnable parameters and feature embeddings. The "linguistically motivated" aspect of the algorithm are the selected features for the computation of this probability.
- An alignment algorithm that constructs sets of aligned phrase pairs and their probabilities using a function that estimates the probability of a phrase alignment. It uses this function in a recursive equation to compute the probabilities of the alignment of the phrases taking into account the probabilities of the supporting alignments. The algorithm works by constructing the alignments in a bottom up fashion by selecting pairs of already aligned phrases (word alignments in the beginning) and considering the alignment of the lowest common ancestors (LCA) of these pairs of phrases, this is repeated until the roots of the parse tree are aligned. The result of the process is a

parse tree alignment (which consists of a set of phrase alignments) and probabilities for these phrase alignments.

This method works on tree-alignment but can be extended with minor changes to forest alignment. The proposed method was trained and evaluated in a new dataset (also part of this work) which consists of a set of manually annotated phrase alignments. The experiment results showed that the methods performance is not very far from human performance.

A chance-corrected measure of inter-annotator agreement for syntax

When annotating a corpus, having a notion for the consistency of annotations between different annotators in a given annotation task, is crucial for understanding the qualities and limitations of the annotated corpus. Inter-annotator agreement (IAA) metrics provide a quantitative measurement (a number) of the agreement between annotations of different annotators, these can be used to give a quantitative comparison between different annotated corpora.

There are a lot of IAA metrics for many different kinds of annotation, each one with its pros and cons: some are sensitive to some kinds of errors and not other kinds, some are very strict (score very low) and some are very lenient (score very high), some are chance-corrected metrics (takes into account the probability of two annotations agreeing by chance) and some are not. All in all there is no perfect formula for assessing IAA, but rather a myriad of metrics that can be used in conjunction to gain an understanding of the quality of the annotation.

Most IAA metrics are defined under the assumption that annotations contain little to no internal structure (such as in a label), in fields like syntactic parsing however, where the annotations are trees, structure is the main point of annotation. In this case the metrics have to be adapted to effectively deal with structure. Previous to the work of this paper there where two prevailing methods for IAA: adapted F-score and accuracy metrics, and a similarity measure specific for dependency trees called LAS. The former has the issue of being biased in favour of annotation schemes with fewer categories and not accounting for skewed distributions between classes, and the latter is not chance-corrected which has some limitations, and only works with dependency trees.

One possibility that this paper observes to find a good IAA chance-corrected metric is to chance-correct LAS. This possibility is soon discarded as it has many inconveniences that make it impractical, for instance it requires a random model of tree annotation which would have to enumerate a number of trees exponential in the length of the annotated sentences. It would also require a probabilistic model of the annotators tree annotations, for which there is no one clear way to define a model, possibly resulting in different definitions by different practitioners, which would make the results in different papers incompatible.

Instead, this paper proposes a method based on Krippendorff's α, which is a chance-corrected measure that works on any items for which a distance function (that defines a valid metric-space) can be defined. The annotation items in syntactic parsing are trees, so a valid distance function must be defined that works with trees. In this work the Tree Edit Distance (TED) metric is used, which is a generalization of the String Edit Distance. They also experiment with two variations of TED which attempt to nullify or normalize the effect that the difference of sentence length has.

The results show that this metric has similar behaviour to LAS while also being chance-corrected and applicable to structural annotation schemes other than dependency parsing (such as constituency parsing), making the metric more sound. Another difference from LAS is that this new metric is more lenient with errors of token tags, and focuses more in the actual structure of the annotated tree, which may or may not be of interest.

Reranking and Self-Training for Parser Adaptation

This paper is focused on parser adaptation whose purpose is to use existing labeled data from a domain and create a parser that is able to parse a different domain. There is a high need for parser portability since there aren't sufficient corpora from different domains that can be used as training data.

McClosky et al. (2006) were able to deliver promising results on parser adaptation without in-domain data. They showed that parser performance can be improved by using two recent techniques for parser improvement, self-training and parse-reranking.

They approached the issue by using a first-stage n-best parser trained on some WSJ labeled out-of-domain data to parse sentences from the LA Times (unlabeled out-of-domain data taken from the North American News Corpus, or NANC).The n-best parse trees that this parser produced were reranked and the highest ranked parse trees were then added to the training corpus and the parser was retrained.

This self-training method not only improved the performance of the WSJ (absolute f-score improvement of 0.8%) but also of test sentences taken from the Brown corpus (absolute f-score improvement of 2.6%). In their experiments, self-training is used on parse trees from the same domain (newspaper articles) as the parser's original training data and perhaps this is the reason why they achieved such results and why they expect that the parser would not perform well on text from another domain, like medical text. However, even though they have not experimented with data from different domains, they speculate that a self-trained parser based on that data might work even better than their standard best.

Learning Paraphrase Identification with Structural Alignment

This paper suggests a new method to calculate Semantic similarity of text, using both local information like lexical semantics and structural information like syntactic structures. They propose a new alignment-based approach that uses an attributed relational graphs, to encode lexical, syntactic and semantic information. Alignment of two such graphs combines local and structural information to support similarity estimation. To improve alignment, they introduced structural constraints inspired by a cognitive theory of similarity and analogy. Because similarity labels are given in training data and the true alignments are unknown, so they address the learning problem using two approaches: alignment as feature extraction and alignment as latent variable. The system is supposed to predicts whether two sentences can be considered semantically equivalent or not. They solve this problem with a pipeline of three components:

- Graph extractor: Given two pieces of text, it uses word embeddings and a set of automatic annotators to extract the tokens, syntactic relations, POS tags and entity mentions to generate the attributed relational graphs.
- Structural aligner: Given two attributed relational graphs, the structural aligner generates an alignment. An alignment of two attributed relational graphs is a set of matches, and each match is a correspondence between two nodes or edges.
- Similarity estimator: Given an alignment, the similarity estimator produces a similarity score between the two graphs or a label indicating whether they are similar enough to be considered equivalent or not.

Their method uses a hybrid representation, attributed relational graphs (directed graphs with attributes attached to the nodes and edges). The attributes store local information about a unit/node or a relation/edge, which will later be used to extract features for each match between two nodes or two edges. They use tokens as units/nodes and dependency arcs as relations/edges. For attributes, they used dependency label, token, lemma, POS tag, NER tag and word embedding. There are two advantages of this representation. First, it is expressive enough to encode heterogeneous structural information in the same graph; second, local information can be easily encoded as attributes.

The two core components of their approach are the structural aligner and similarity estimator. Given two input graphs, the structural aligner finds the best alignment between them, which can be seen as a structured prediction problem. Based on the best alignment, the similarity estimator produces a score indicating the degree of similarity or a binary output indicating whether the two sentences are semantically equivalent or not, which can be seen as a regression or classification task depending on which output is produced. An alignment is a set of matches. Each match is a pair of nodes or edges from the two graphs. The structural alignment has two steps:

First, given two graphs, the structural aligner generates all the possible matches that pass some criteria. In this work, the criteria they used are that (1) two matched dependency arcs must have the same dependency label; (2) the cosine similarity between word embeddings of two matched tokens must be greater than come threshold (value is chosen from pilot experiments with a subset of the data). Second, it selects the subset of matches that optimizes a function mapping input graphs and a candidate alignment to a feature vector. They use beam search to find the alignment.

Formalized as a regression or classification problem, the similarity estimator takes the pair of graphs and the predicted alignment as input, and uses another feature function to map them to a feature vector. They use SVM to learn the set of parameters for the similarity estimator.

Learning the set of parameters for the structural aligner is more challenging because the true alignment is usually latent. They address this problem using two approaches, alignment as feature extraction and alignment as latent variable.

In the first approach, they consider the structural aligner as a feature extractor and the set of parameters for the similarity estimator as hyperparameters, and use grid search over a validation set to select the best parameters. But the problem is that the number of runs needed in grid search grows exponentially with the number of hyperparameters, So they have to restrict the mapping function of similarity estimator to include just a small set of features. In addition, the grid search cannot be very fine-grained due to its computational cost. To overcome these problems and utilize more features in the structural aligner, the second approach jointly trains the structural aligner and the similarity estimator through an iterative process.

First, they initialize the parameters to some value obtained from the first approach. Then they repeat two steps: Keep set of parameters for the structural aligner fixed, for each input, assume the alignments produced by the structural aligner is “correct” and learn the set of parameters for the similarity estimator. Keep the set of parameters for the similarity estimator fixed, for each input, hallucinate the “correct” alignment.

Features used in feature functions:

Unary features are used to estimate how similar two matched tokens or dependency arcs are, and also how important they are in their sentences. These features are used to compute how much this match will contribute to the alignment score or overall similarity.

Pairwise features are introduced to improve alignment by encoding the structural constraints between matches. These structural constraints ensure that the final alignment is structurally consistent. They are inspired by the structural consistency principle of Structure Mapping theory. The principle states that two constraints are used by human when aligning predicate-argument structures:

(1) one-to-one mapping that one entity or predicate should only match to one entity or predicate; (2) parallel connectivity that if a predicate matches another predicate, their roles and arguments should also match correspondingly. In this work, they adapted these constraints to work on tokens and syntactic relations.

So if the heads of two dependency arcs match, the two dependency arcs should also be more likely to match. If two dependency arcs match, the dependent of the dependency arcs should be more likely to match as well.

The alignment and rich features enabled the system to learn which part of the sentences are more important to its semantic rather than treating them all the same. Structural alignment further eliminates false positives because it helps constrain the lexical matches.In the experiment, this approach achieved results competitive with other state-of-the-art models on the MSRP corpus. Further analysis showed the strength of this hybrid approach and confirmed contributions of structural alignment using structural constraints and joint learning.

The CoNLL 2007 Shared Task on Dependency Parsing

The idea is to explore data-driven methods for multilingual dependency parsing, and the problem of domain adaptation. The domain adaptation task was to use machine learning to adapt a parser for a single language to a new domain. The official evaluation metric in both tracks was the labeled attachment score (LAS), i.e., the percentage of tokens for which a system has predicted the correct HEAD and DEPREL, but results were also reported for unlabeled attachment score (UAS), i.e., the percentage of tokens with correct HEAD, and the label accuracy (LA), i.e., the percentage of tokens with correct DEPREL

The multilingual track: with annotated training and test data from a wide range of languages to be processed with one and the same parsing system. This system must therefore be able to learn from training data, to generalize to unseen test data, and to handle multiple languages, possibly by adjusting a number of hyper-parameters.

Domain adaptation track: For this shared-task there is no annotated resources in the target domain. Participants were provided with a large annotated corpus from the source domain, in this case sentences from the Wall Street Journal. They were also provided with data from three different target domains: biomedical abstracts (development data), chemical abstracts (test data 1), and parent-child dialogues (test data 2). Additionally, a large unlabeled corpus for each data set (training, development, test) was provided. The goal of the task was to use the annotated source data, plus any unlabeled data, to produce a parser that is accurate for each of the test sets from the target domains.

In multilingual track there were two main paradigms for learning and inference: transition-based parsers and graph-based parsers.

Transition-based parsers build dependency graphs by performing sequences of actions, or transitions. Both learning and inference is conceptualized in terms of predicting the correct transition based on the current parser state and/or history. We can further subclassify parsers with respect to the model (or transition system) they adopt, the inference method they use, and the learning method they employ.

Graph-Based Parsers While transition-based parsers use training data to learn a process for deriving dependency graphs, graph-based parsers learn a model of what it means to be a good dependency graph given an input sentence. They define a scoring or probability function over the set of possible parses. At learning time, they estimate parameters of this function; at parsing time they search for the graph that maximizes this function. These parsers mainly differ in the type and structure of the scoring function (model), the search algorithm that finds the best parse (inference), and the method to estimate the function's parameters (learning).

Domain Adaptation

Feature-Based Approaches: One way of adapting a learner to a new domain without using any unlabeled data is to only include features that are expected to transfer well. In structural correspondence learning a transformation from features in the source domain to features of the target domain is learnt. The original source features along with their transformed versions are then used to train a discriminative parser.

Ensemble-Based Approaches: One system trained a diverse set of parsers in order to improve cross-domain performance by incorporating their predictions as features for another classifier. Similarly, two parsers trained with different learners and search directions were used in the co-learning approach. Unlabeled target data was processed with both parsers. Sentences that both parsers agreed on were then added to the original training data. This combined data set served as training data for one of the original parsers to produce the final system. In a similar fashion, another system used a variant of self-training to make use of the unlabeled target data.

Other Approaches: One system learnt tree revision rules for the target domain by first parsing unlabeled target data using a strong parser; this data was then combined with labeled source data; a weak parser was applied to this new dataset; finally tree correction rules are collected based on the mistakes of the weak parser with respect to the gold data and the output of the strong parser. Another technique used was to filter sentences of the out-of-domain corpus based on their similarity to the target domain, as predicted by a classifier. Only if a sentence was judged similar to target domain sentences was it included in the training set. Another system used a hybrid approach, where a data-driven parser trained on the labeled training data was given access to the output of a Constraint Grammar parser for English run on the same data. Finally, another system learnt collocations and relational nouns from the unlabeled target data and used these in their parsing algorithm.

System Combination: To combine the outputs of each parser in multilingual track they assign to each possible labeled dependency a weight that is equal to the number of systems that included the dependency in their output. This can be viewed as an arc-based voting scheme. Using these weights, it is possible to search the space of possible dependency trees using directed maximum spanning tree algorithms. The maximum spanning tree in this case is equal to the tree that on average contains the labeled dependencies that most systems voted for. It is done by sorting the systems based on their average labeled accuracy scores over all languages, and then incrementally adding each system in descending order. both labeled and unlabeled accuracy are significantly increased, even when just the top three systems are included. Accuracy begins to degrade gracefully after about ten different parsers have been added. Furthermore, the accuracy never falls below the performance of the top three systems.

In domain adaptation track four closed system outperformed the best scoring open system. Considering that approximately one third of the words of the chemical test set are new, the results are noteworthy. The next surprise is to be found in the relatively low UAS for the CHILDES data. One major reason for this is that auxiliary and main verb dependencies are annotated differently in the CHILDES data than in the WSJ training set. In this domain, it seems more feasible to use general language resources than for the chemical domain. However, the results prove that the extra effort may be unnecessary (small difference between top closed system and top open system)