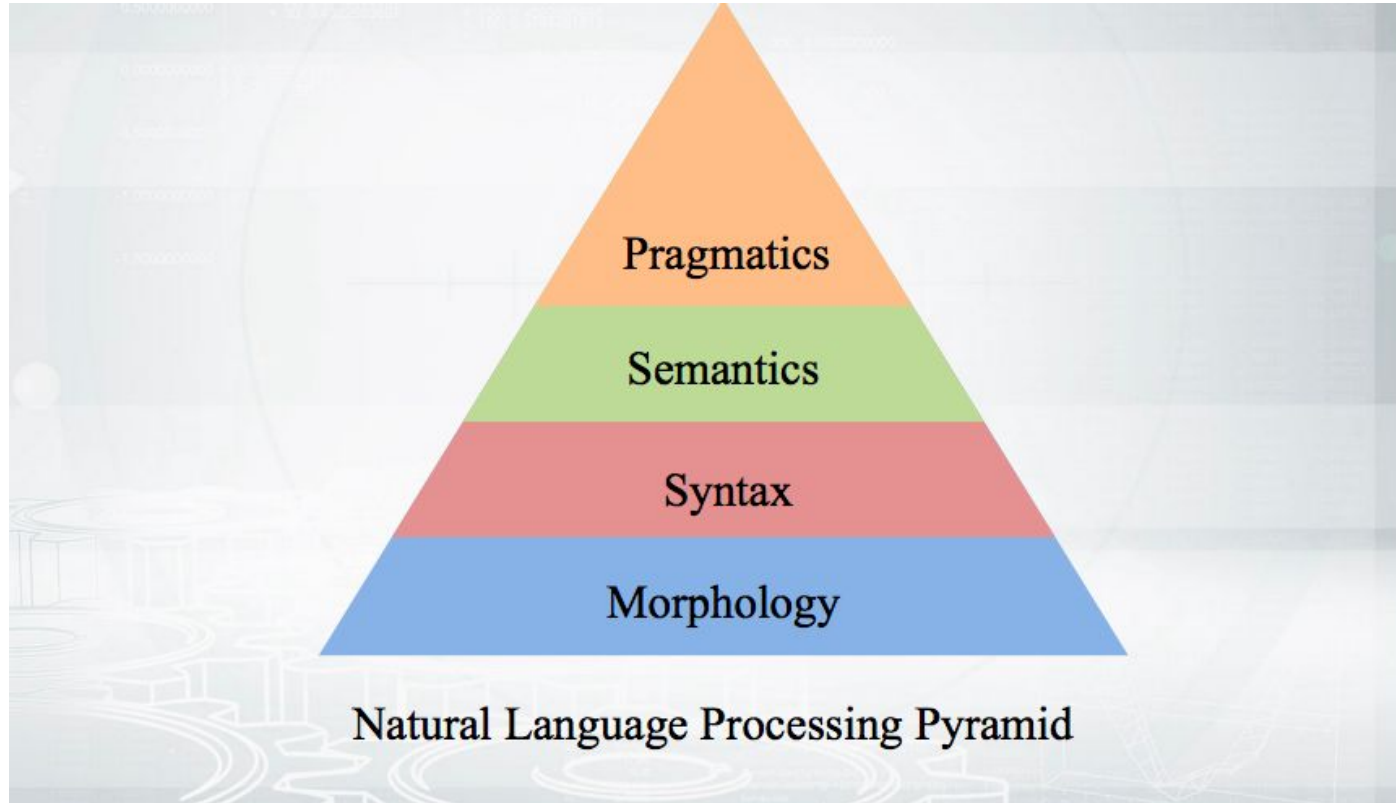


Содержание

1. **Что такое синтаксис и зачем он нужен**
2. Теоретические фреймворки
3. Dependency parsing
4. Метрики и соревнования
5. Инструменты
6. Varia

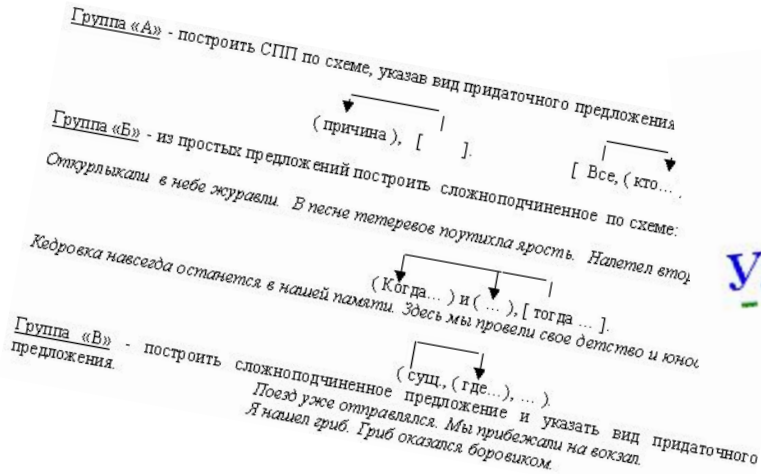
Где мы вообще?



Где мы вообще?

машинный анализ структуры текста, в первую очередь структуры предложения

Мы все это делали в школе, и машине это зачастую тоже под силу



Зачем это нужно?

- «Банкомат съел карту» vs «карта съела банкомат»
- Определение правильности грамматики фразы (при порождении речи)
- Question answering, в частности Knowledge-based QA
- Машинный перевод
- Information extraction
- Синтаксическая **роль** токена как метрика его важности (подлежащее важнее определения), использование весов в классификаторе
- BERT кое-что знает о синтаксисе. А уж его знания лишними явно не бывают :)

Содержание

1. Что такое синтаксис и зачем он нужен
- 2. Теоретические фреймворки**
3. Dependency parsing
4. Метрики и соревнования
5. Инструменты
6. Varia

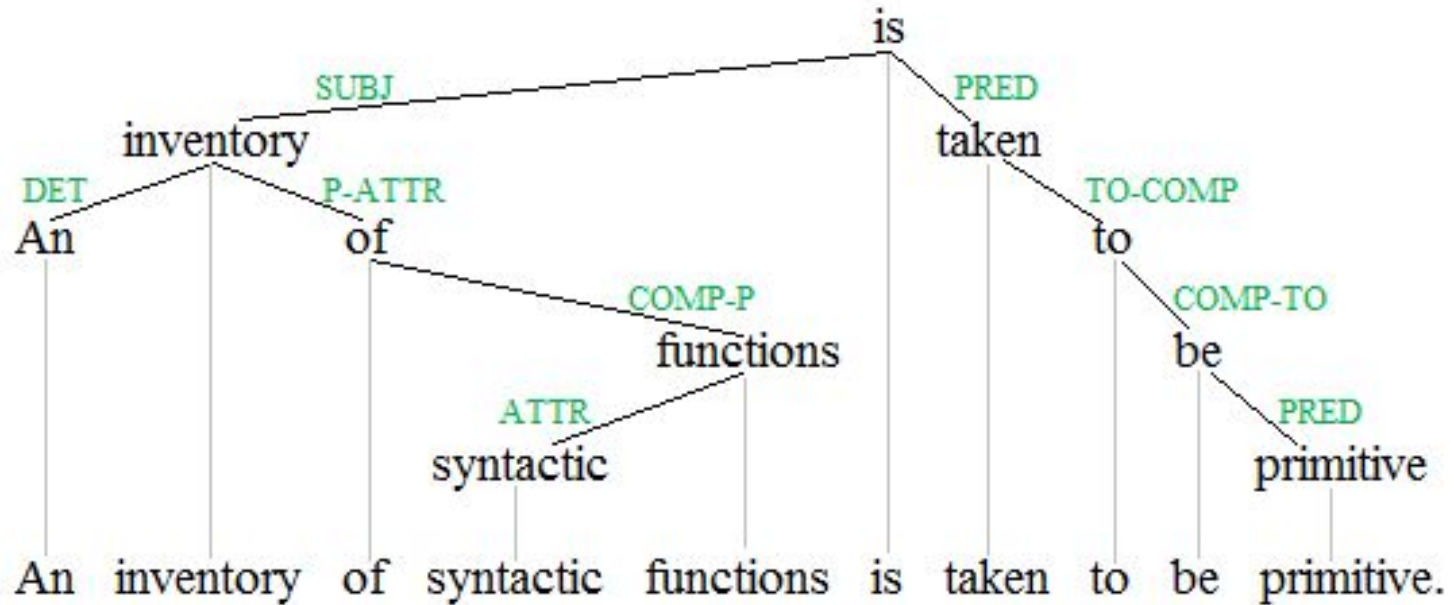
Грамматика непосредственно составляющих (constituency) и грамматика зависимостей (dependency)



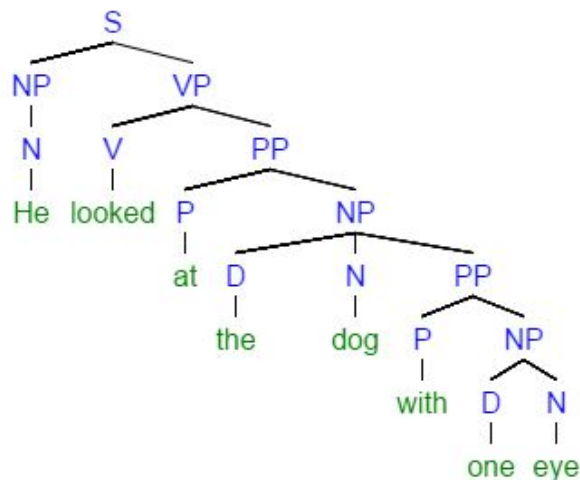
- [[моя мама] [мыла] [грязную раму]]

- В автоматическом парсинге для русского языка нет phrase-based парсеров (проблемы со “свободным” порядком слов)
- У многих других языков [получше](#)

Пример разбора: зависимости



Пример разбора: непосредственно составляющие



<https://i.imgur.com/ShMtNEy.png>

VP - глагольная группа, verb phrase

(грубо: состоит из глагола и зависимых;
но не подлежащее)

NP - именная группа, noun phrase

(грубо: вершина — существительное)

PP - предложная группа, prepositional phrase

AP - группа прилагательного, adjective phrase

D (Det) - детерминативы: артикли, указ., притяж.,
определятельные местоимения, квантификаторы,
числительные, вопросительные слова

...

Constituency vs Dependencies

- Хорошо разработанные в лингвистике теории
- Отчасти (!) формально (!) взаимозаменяемые — грубо говоря, обе основаны на правилах взаимодействия частей речи
- Нет главной и нет вторичной (хотя ГЗ слегка устарела)
- Как водится, много проблем на периферии у обеих, см. ([Тестелец 2001](#))
- [SoTA работа](#) (2019) по автоматическому парсингу умеет за раз в оба фреймворка (правда, только на некоторых языках) :)

А в чем проблемы?

- Эллипсис (aka пропуски)
- Синтаксическая омонимия

Он увидел их семью своими глазами

Он сам увидел их семью

Он увидел их при помощи своих семи глаз

ср. хрестоматийное “Эти типы стали есть в цехе”, “подпись руководителя группы или командированного лица”

- Непроективность

(Не)проективность (формализуя “нехорошее”)

Предложение называется **проективным**, если $\langle \dots \rangle$:

а) ни одна из стрелок не пересекает другую стрелку

б) никакая стрелка не накрывает корневую (\sim сказуемое \rightarrow подлежащее)

[[Тестелец 2001](#): 95]

Все мечтают выиграть кубок.

← Проективное

б.

Кубок все выиграть мечтают.

← НЕпроективное — нарушен принцип **пересечения**

в.

Кубок все мечтают выиграть.

← НЕпроективное — нарушен принцип **обрамления**

- Non-projectivity in natural languages

Language	Trees	Arcs
Arabic [Hajič et al. 2004]	11.2%	0.4%
Basque [Aduriz et al. 2003]	26.2%	2.9%
Czech [Hajič et al. 2001]	23.2%	1.9%
Danish [Kromann 2003]	15.6%	1.0%
Greek [Prokopidis et al. 2005]	20.3%	1.1%
Russian [Boguslavsky et al. 2000]	10.6%	0.9%
Slovene [Džeroski et al. 2006]	22.2%	1.9%
Turkish [Oflazer et al. 2003]	11.6%	1.5%

Table from invited talks by Prof. Joakim Niver: [Beyond MaltParser - Advances in Transition-Based Dependency Parsing](#)

Грамматика зависимостей

- Бурное развитие в computational сфере
- Лучше применима к парсингу русского языка
- Всё не успеть за одну пару
- Субъективный выбор лектора
(про CP немножко теории обещано и на семинаре)
- Constituency parsing освещен в литературе, особенно см. [две главы учебника Журафского и Мартина](#)

Содержание

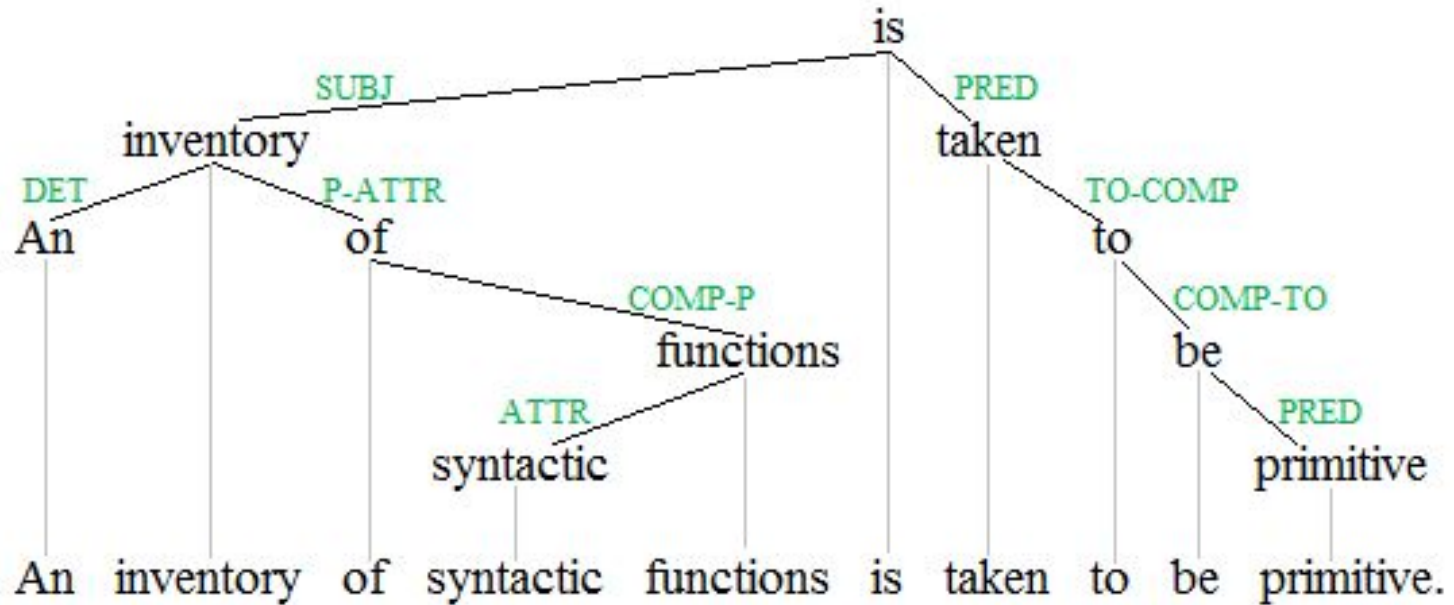
1. Что такое синтаксис и зачем он нужен
2. Теоретические фреймворки
3. **Dependency parsing**
4. Метрики и соревнования
5. Инструменты
6. Varia

Дерево зависимостей [[Jurafsky & Martin 2017](#)]

“Dependency tree is a directed graph that satisfies the following constraints:

- There is a single designated **root node that has no incoming arcs.**
- With the exception of the root node, **each vertex has exactly one incoming arc.**
- There is a **unique path from the root node to each vertex in V .**”

Пример разбора: зависимости



Dependency parsing: алгоритмы

Построение дерева зависимостей по предложению

Два основных подхода (supervised learning; еще есть их микс, но это довольно маргинально); “нейронная революция” кардинально не поменяла основные алгоритмы, хотя улучшила их качество

- **transition-based**

жадно набираем дерево (см. далее)

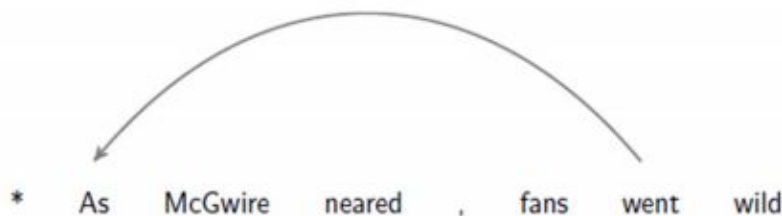
- **graph-based**

ищем минимальное остовное дерево (minimum spanning tree; MST)

в полном графе всех возможных связей

(быстрее работает при непроективности + обычно лучше работает с длинными предложениями)

Features for one dependency

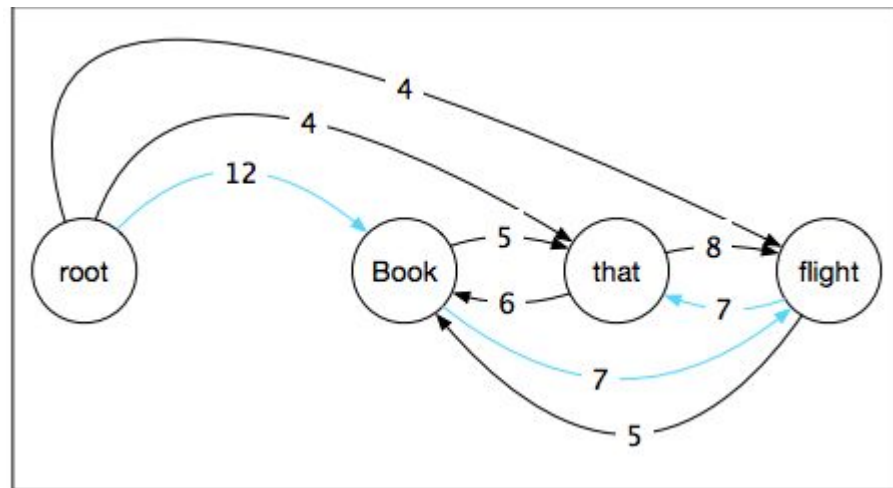


Example from slides of Rush and Petrov (2012)

[went]	[VBD]	[As]	[ADP]	[went]
[VERB]	[As]	[IN]	[went, VBD]	[As, ADP]
[went, As]	[VBD, ADP]	[went, VERB]	[As, IN]	[went, As]
[VERB, IN]	[VBD, As, ADP]	[went, As, ADP]	[went, VBD, ADP]	[went, VBD, As]
[ADJ, *, ADP]	[VBD, *, ADP]	[VBD, ADJ, ADP]	[VBD, ADJ, *]	[NNS, *, ADP]
[NNS, VBD, ADP]	[NNS, VBD, *]	[ADJ, ADP, NNP]	[VBD, ADP, NNP]	[VBD, ADJ, NNP]
[NNS, ADP, NNP]	[NNS, VBD, NNP]	[went, left, 5]	[VBD, left, 5]	[As, left, 5]
[ADP, left, 5]	[VERB, As, IN]	[went, As, IN]	[went, VERB, IN]	[went, VERB, As]
[JJ, *, IN]	[VERB, *, IN]	[VERB, JJ, IN]	[VERB, JJ, *]	[NOUN, *, IN]
[NOUN, VERB, IN]	[NOUN, VERB, *]	[JJ, IN, NOUN]	[VERB, IN, NOUN]	[VERB, JJ, NOUN]
[NOUN, IN, NOUN]	[NOUN, VERB, NOUN]	[went, left, 5]	[VERB, left, 5]	[As, left, 5]
[IN, left, 5]	[went, VBD, As, ADP]	[VBD, ADJ, *, ADP]	[NNS, VBD, *, ADP]	[VBD, ADJ, ADP, NNP]
[NNS, VBD, ADP, NNP]	[went, VBD, left, 5]	[As, ADP, left, 5]	[went, As, left, 5]	[VBD, ADP, left, 5]
[went, VERB, As, IN]	[VERB, JJ, *, IN]	[NOUN, VERB, *, IN]	[VERB, JJ, IN, NOUN]	[NOUN, VERB, IN, NOUN]
[went, VERB, left, 5]	[As, IN, left, 5]	[went, As, left, 5]	[VERB, IN, left, 5]	[VBD, As, ADP, left, 5]
[went, As, ADP, left, 5]	[went, VBD, ADP, left, 5]	[went, VBD, As, left, 5]	[ADJ, *, ADP, left, 5]	[VBD, *, ADP, left, 5]
[VBD, ADJ, ADP, left, 5]	[VBD, ADJ, *, left, 5]	[NNS, *, ADP, left, 5]	[NNS, VBD, ADP, left, 5]	[NNS, VBD, *, left, 5]
[ADJ, ADP, NNP, left, 5]	[VBD, ADP, NNP, left, 5]	[VBD, ADJ, NNP, left, 5]	[NNS, ADP, NNP, left, 5]	[NNS, VBD, NNP, left, 5]
[VERB, As, IN, left, 5]	[went, As, IN, left, 5]	[went, VERB, IN, left, 5]	[went, VERB, As, left, 5]	[JJ, *, IN, left, 5]
[VERB, *, IN, left, 5]	[VERB, JJ, IN, left, 5]	[VERB, JJ, *, left, 5]	[NOUN, *, IN, left, 5]	[NOUN, VERB, IN, left, 5]

Graph-based dependency parsing

- Изначально имеем полный орграф
- Все ребра и все типы связей
- При обучении учимся скорить связи
- Фичи примерно с предыдущего слайда
- + могут быть фичи про порядок слов
- Постпроцессинг: фильтр на циклы



Проще справляться с непроективностью: без этого ограничения нам нужно меньше постпроцессинга, просто берем топ-кандидата, не отбирая именно топ-проективного

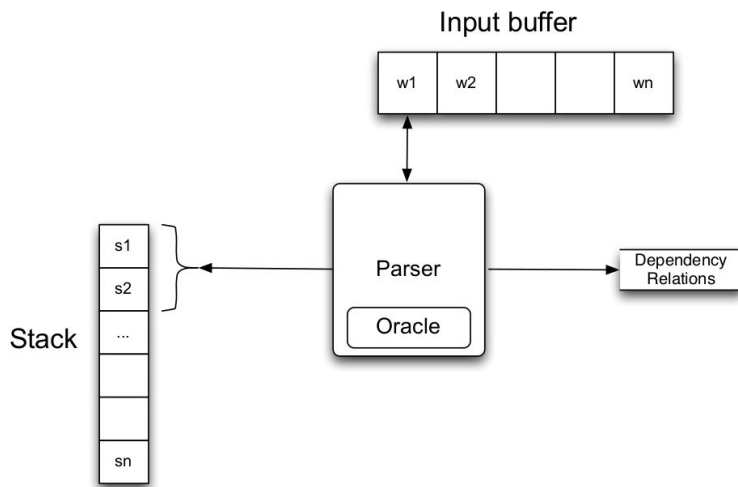
Transition-based (arc-standard) dependency parsing

Есть список токенов, стек (изначально содержит только root) и конфигурация (изначально пустая). Три дефолтных способа изменить конфигурацию:

- **LeftArc** [применим, если второй элемент стека не ROOT]
проводим зависимость между токеном на верхушке стека и вторым + выкидываем второй из стека
- **RightArc**
то же, но зависимость в другую сторону, и выкидываем верхушку стека
- **Shift**
переносим очередное слово из буфера в стек

Transition-based dependency parsing

Ключевое понятие: “**конфигурация**” = **состояние** процесса разбора:
входящие токены, верхушка стека и набор уже построенных отношений
(то, что мы “держали в уме”, когда играли; да, аналогия не вполне точна)



Потому и transition-based -- мы сейчас будем **переходить** из состояния в состояние системы по правилам

Псевдокод

function DEPENDENCYPARSE(*words*) **returns** dependency tree

state \leftarrow { [root], [*words*], [] } ; initial configuration

while *state* **not** **final**

 t \leftarrow ORACLE(*state*) ; choose a transition operator to apply

 state \leftarrow APPLY(*t*, *state*) ; apply it, creating a new state

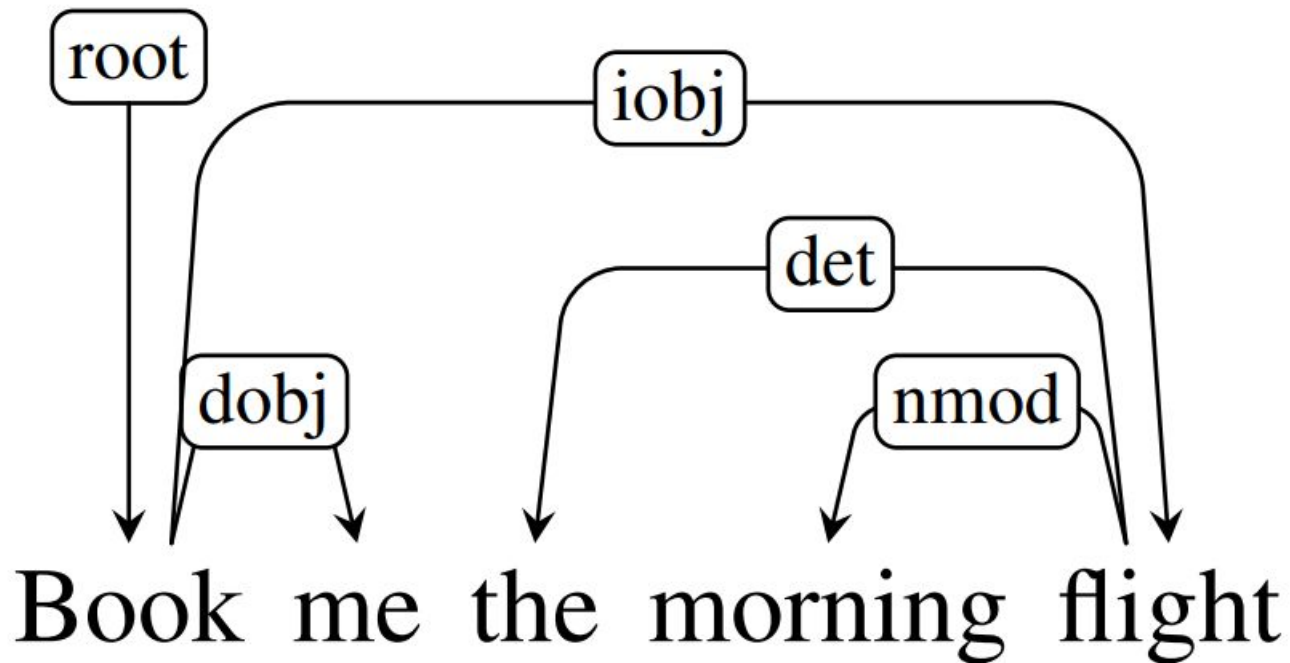
return *state*

Пример работы

Немножко игра в пьяницу между стеком и буфером...

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

Пример работы [[Jurafsky & Martin 2017](#)]



А что тут с непроективностью?

- Есть математическое доказательство, что сферический в вакууме transition based-parser может разобрать любое проективное предложение
- А непроективное?



Нарушен принцип пересечения — предложение непроективное.

Базовый ТВ-parser и непроективность

Я мальчика вижу красивого

Шаг	Стэк	Буфер
0	[root]	[я, мальчика, вижу, красивого]
1	[root, я]	[мальчика, вижу, красивого]
2	[root, я, мальчика]	[вижу, красивого]
3	[root, я, мальчика, вижу]	[красивого]
4	[root, я, мальчика, вижу, красивого]	[]

Цугцванг? Или можно соединить “вижу” и “мальчика”?..

Базовый TV-parser и непроективность

Шаг	Стэк	Буфер	Операция
0	[root]	[я, мальчика, вижу, красивого]	SHIFT
1	[root, я]	[мальчика, вижу, красивого]	SHIFT
2	[root, я, мальчика]	[вижу, красивого]	SHIFT
3	[root, я, мальчика, вижу]	[красивого]	LeftArc
4	[root, я, вижу]	[красивого]	LeftArc
5	[root, вижу]	[красивого]	SHIFT
6	[root, вижу, красивого]	[]	RightArc
7	[root, вижу]	[]	RightArc
8	[root]	[]	

Модификации для непроективности

via [[Kuhlmann, Nivre 2010](#)]

- Pseudo-projective parsing [[Nivre, Nilsson 2005](#)]
- Non-adjacent arc-transitions [[Attardi 2006](#)]
(и далее [[Cohen et al. 2011](#)], [[Gómez-Rodríguez et al. 2014](#)])
- Online reordering [[Nivre 2009](#)]

Сравнивались на корпусах немецкого, чешского и английского языков

Спойлер: последнее чуть лучше остальных...

Online reordering

- Концепция: добавить **четвертую операцию SWAP** — вернуть второй элемент стека обратно в буфер, тем самым изменив порядок токенов в буфере (т.е. исходный порядок слов!)
- Лучше ее применять с пониженным весом (только если других вариантов больше не осталось), см. [[Nivre et al. 2009](#)]

Online reordering

- Пусть есть предложение “A B”
- Тогда я могу вернуть A в буфер, и если я потом его верну снова в стек, то как бы получу строку “B A”
- Поэтому online **reordering**: де-факто я могу рассматривать строку с другим порядком токенов.
- Хочется из “я мальчика вижу красивого” получить хотя бы “я мальчика красивого вижу”...

Online reordering

Я мальчика вижу красивого

Шаг	Стэк	Буфер	Операция
0	[root]	[я, мальчика, вижу, красивого]	SHIFT
1	[root, я]	[мальчика, вижу, красивого]	SHIFT
2	[root, я, мальчика]	[вижу, красивого]	SHIFT
3	[root, я, мальчика, вижу]	[красивого]	SHIFT
4	[root, я, мальчика, вижу, красивого]	[]	SWAP
5	[root, я, мальчика, красивого]	[вижу]	RightArc

Online reordering

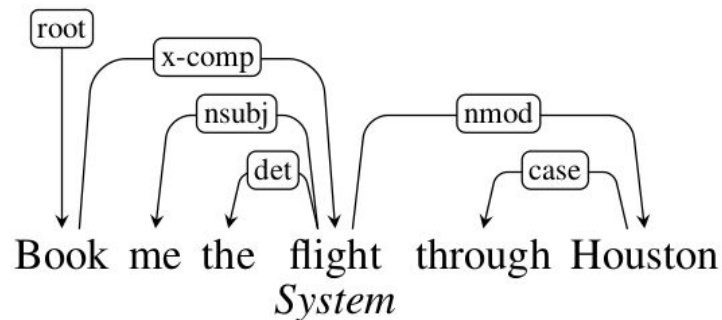
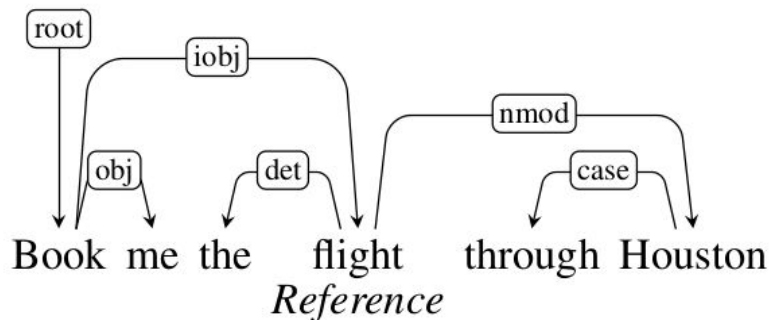
Я мальчика вижу красивого

Шаг	Стек	Буфер	Операция
6	[root, я, мальчика]	[вижу]	SHIFT
7	[root, я, мальчика, вижу]	[]	LeftArc
8	[root, я, вижу]	[]	LeftArc
9	[root, вижу]	[]	RightArc
10	[root]	[]	

Содержание

1. Что такое синтаксис и зачем он нужен
2. Теоретические фреймворки
3. Dependency parsing
4. **Метрики и соревнования**
5. Инструменты
6. Varia

Оценка качества



Unlabeled Attachment Score (UAS) = 5/6

(правильно приписана вершина)

Labeled Attachment Score (LAS) = 4/6

(правильно приписаны вершина **И** тип метки)

+ macro-averaged vs micro-averaged (по предложениям vs независимо от предложений)

И что, берем accuracy?

Проект Universal dependencies

- Лингвистическая проблема: несоответствие терминов и правил из грамматик зависимостей разных языков
- Computational challenge: обучить синтаксический парсер для многих языков, включая low-resource languages
⇒ <http://universaldependencies.org/>
- > 100 версионированных трибанков (размеченных корпусов) для > 70 языков, теги зависимостей [унифицированы](#)

Соревнование пайплайнов

- [CONLL 2017 Shared Task](#) “from raw text to dependencies”
- 81 трибанк, 49 языков (82 трибанка, 57 языков в 2018)
- Парсинг сырого текста **vs** брать бейзлайн токенизацию и/или морфологию
- Один из лучших результатов среди всех команд и всех трибанков был показан на корпусе русского языка: 94% UAS и 92,6% LAS (чуть меньше в 2018)

Соревнование пайплайнов: к дискуссии о метриках

- [CONLL 2017 Shared Task](#)
- F-мера!
- **Точность** = количество точных попаданий/количество предсказаний
- **Полнота** = количество точных попаданий/количество связей в размеченных данных
- От чего зависит полнота?

Соревнование пайплайнов: к дискуссии о метриках

- При идеальной токенизации точность совпадает с полнотой, а значит:

$$F = 2PR/(P+R) = 2*x*x/(x+x) = x \text{ (=точность=полнота=accuracy)}$$

- При неидеальной токенизации значение F-меры меняется (точность != полнота)
- “Синтаксическая” метрика зависит от токенизации

Соревнование пайплайнов: к дискуссии о метриках





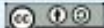


- Не очень честное сравнение пайплайнов (кто-то мог делать свою токенизацию)
- Отсутствие метрики, позволяющей чисто теоретически выделить идеальный сферический парсер в вакууме
- Отсутствие единой метрики, позволяющей сравнить весь пайплайн целиком (from raw text to dependencies)
- Но как делать иначе — не очень понятно...

Соревнование пайплайнов: [CONLLI 2018 Shared Task](#)

- **LAS** (labeled attachment score) will be computed the same way as in the 2017 task so that results of the two tasks can be compared
- **MLAS** (morphology-aware labeled attachment score) is inspired by the CLAS metric computed in 2017, and extended with evaluation of POS tags and morphological features
- **BLEX** (bi-lexical dependency score) combines content-word relations with lemmatization (but not with tags and features)

Данные: русский язык

Russian treebanks

▶	GSD	98K	(L)(F)	W		★★★★★
▶	SynTagRus	1,107K	(L)(F)(D)			★★★★★
▶	Taiga	38K	(L)(F)			★★★★★
▶	PUD	19K	(L)(F)			★★★★★

See [here](#) for comparative statistics of Russian treebanks.

⇒ Практически все эксперименты проводятся на SynTagRus
(корпуса для всех языков [лежат на гитхабе](#))

Содержание

1. Что такое синтаксис и зачем он нужен
2. Теоретические фреймворки
3. Dependency parsing
4. Метрики и соревнования
- 5. Инструменты**
6. Varia

Инструменты

- См. результаты дорожек [Conll-17](#) и [Conll-18](#)
- Осторожно: есть академические и закрытые разработки
- UDPipe
- Syntaxnet (оба изначально transition-based, но не обманывайтесь: например, в дорожках-17 и -18 их обошел [graph-based](#) парсер)
- UDPipe с 2018 пересел на graph-based архитектуру, но полноценного релиза еще не было

UDPipe vs Syntaxnet

	UDPipe (Future)	Syntaxnet (parseysaurus-17)
UAS (russian, syntagrus)	92.96%	92.67%
LAS (russian, syntagrus)	91.46%	88.68%
Время парсинга одного предложения	~ 3 ms	~ 100 ms
Возможность “распилить” пайплайн	+	-
Запуск напрямую без докера и др.	+	-

UDPipe

- UDPipe — пайплайн, обучаемый токенизации, лемматизации, морфологическому тэггингу и парсингу, основанному на грамматике зависимостей
- [Статья об архитектуре](#), [репозиторий с кодом обучения](#), [мануал](#)
- Есть готовые [модели](#) (в том числе и для РЯ)
- Подобранные для каждого корпуса параметры обучения [зарелизены](#)

UDPipe: архитектура

- **Совместное деление на слова и предложения:**
однослойная двухсторонняя GRU, для каждого символа предсказывающая, последний ли он в предложении и/или токене
- **Теггер:** по последним четырем символам каждого слова генерируем триплеты (UPOS, XPOS, FEATS), при помощи перцептрона выбираем лучшего кандидата

UDPipe: архитектура

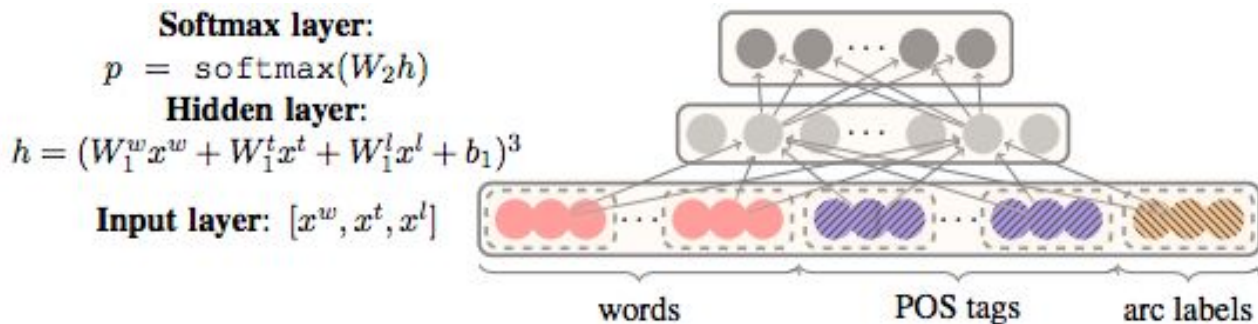
- **Лемматизатор:** генерируем пары (lemma rule, UPOS), лемму предсказываем, отрезая префиксы и суффиксы и генерируя новые на их место; перцептрон выбирает наилучшего кандидата
- **Раздельное предсказание тегов и лемм**
(2 модели, но можно соединить в одну)
- + можно подключить свой список лемм

UDPipe: архитектура

- Dependency parsing ([[Straka et al. 2015](#)]): transition-based arc-standard dependency parser
- Один скрытый слой, нет рекуррентности, см. картинку
- Mini-batched SGD при обучении
- До 18 источников фич на вход: 3 элемента на вершине стека, 3 элемента на вершине буфера, первый и второй левый и правый потомки 2 элементов на вершине стека, и самый левый и самый правый потомок 2 элементов на вершине стека
- Можно загрузить pre-trained семантические эмбединги форм или лемм (см. *мышь ест стол*)

UDPipe: фичи для парсера

- Each node is represented using distributed representations of its form, its POS tag and its arc label; the latter only if it has already been assigned
- word2vec-like training
- Network like in [\[Chen 2014\]](#):



Содержание

1. Что такое синтаксис и зачем он нужен
2. Теоретические фреймворки
3. Dependency parsing
4. Метрики и соревнования
5. Инструменты
6. **Varia**

Varia

- А кто там щас SoTA?

- Все очень зависит от языка, SoTA из [таблички Рудера](#) проверялась только на английском и китайском (в обоих “жесткий” порядок слов)
- [Китайцы поверх XLNet](#), но там довольно хитрый препроцессинг дерева в сторону хитрого формализма
- Они ближе к графовой архитектуре, но на самом деле там что-то третье

- А можно unsupervised?

- [Можно](#), уже лет 10 [пытаются](#)! Но пока там своя лига...

Varia

- А что там BERT?

- На ранних слоях относительно компактно сосредоточено “знание” о синтаксисе
- Сырой BERT показывает очень неплохой (82,5%) скор по UUAS
- Правда, пока считали только на английском...
- Еще смотрели на влияние аттеншн хэдов, специальной “синтаксической” головы вроде нет. И это кажется логичным: разные синтаксические отношения выглядят по-разному и “весят” по-разному, поэтому попали в разные головы.

Полезные ссылки

- [Об архитектуре парсера в Spacy + библиография](#)
- [Программа воркшопа на EMNLP-18](#)
- [Материалы курса на ESSLI-18](#)
- [J. Nivre's workshop at EACL-2014](#)
- [Краткое саммари на сайте Себастиана Рудера про результаты в Dependency Parsing](#)
- [SyntaxRuEval-2012](#)

Заключение

- **Теоретические фреймворки:** ГЗ vs ГНС
- **Dependency parsing:** graph-based vs transition-based подходы
- **Метрики:** UAS, LAS + более сложные для from raw text to dependencies
- **Соревнования:** SyntaxRuEval-12, CoNLL-17, CoNLL-18
- **Данные:** корпуса с дорожек CoNLL (в формате conllu)
- **Инструменты:** Syntaxnet, UDPipe