Методы обработки естественного языка

Cortana

Dialogflow

Hey Siri

Можно ли научить машины извлекать из текстов важные данные?

Этой проблемой занимается особое направление искусственного интеллекта: обработка естественного языка, или **NLP (Natural Language Processing)**.

Компьютеры не могут в полной мере понимать человеческий язык, однако они на многое способны. NLP может делать по-настоящему волшебные вещи и экономить огромное количество времени.

Natural Language Processing (далее – NLP) – обработка естественного языка – подраздел информатики и AI, посвященный тому, как компьютеры анализируют естественные (человеческие) языки.
NLP позволяет применять алгоритмы машинного обучения для текста и речи.

Например, можно использовать NLP, чтобы создавать системы вроде распознавания речи, обобщения документов, машинного перевода, выявления спама, распознавания именованных сущностей, ответов на вопросы, автокомплита, предиктивного ввода текста и т.д.

Задачи NLP

- Машинный перевод. *Задача получения полностью автоматического перевода высокого качества пока остается нерешенной.*
- Анализ тональности текста (*классификация отзывов на положительные, отрицательные и нейтральные*)
- Спам фильтрация
- Определение фейковых новостей (здесь анализ текста выступает как часть общего алгоритма)
- Многоклассовая классификация новостей по категориям (рубрикация)
- Извлечение именованных сущностей, NER
- Диалоговые системы, чат-боты
- Саммаризация (на вход система принимает текст большого размера, а выходом служит текст меньшего размера: пересказ, аннотация)
- Argumentation mining, поиск обоснования в тексте (даны факт и текст, нужно найти обоснование данного факта в тексте)

Почему решать задачи NLP сложно?

- **Полисемия** (многозначные слова имеют общий исходный смысл)
- **Омонимия** (разные по смыслу слова произносятся и пишутся одинаково)

- Данные, «хранящиеся» в форме текста на естественном языке, обладают неопределенностью –человек понимает значение того или иного слова, словосочетания, предложения **исходя из контекста**.

С точки зрения анализа текста набор данных часто называют **корпусом** (corpus) и каждая точка данных, представленная в виде отдельного текста, называется **документом** (document).

NLP пайплайн шаг за шагом

NLP – это весело! Обработка естественного языка на Python (proglib.io)

Реализация какой-либо сложной комплексной задачи в машинном обучении обычно означает построение **пайплайна** (конвейера). Смысл этого подхода в том, чтобы разбить проблему на очень маленькие части и решать их отдельно. Соединив несколько таких моделей, поставляющих друг другу данные, вы можете получать замечательные результаты.

Итак, разобьем процесс языкового анализа на стадии: **NLP пайплайн шаг за шагом**

London is the capital and most populous city of England and the United Kingdom. Standing on the River Thames in the south east of the island of Great Britain, London has been a major settlement for two millennia. It was founded by the Romans, who named it Londinium.

В этом параграфе содержится несколько полезных фактов. Хотелось бы, чтобы компьютер смог понять, что Лондон – это город, что он расположен в Англии, был основан римлянами и т. д. Но прежде всего мы должны научить его самым базовым концепциям письменного языка.

Шаг 1. Выделение предложений

Первый этап пайплайна – разбить текст на отдельные предложения. В результате получим следующее:

- London is the capital and most populous city of England and the United Kingdom.
- Standing on the River Thames in the south east of the island of Great Britain, London has been a major settlement for two millennia.
- It was founded by the Romans, who named it Londinium.

Можно предположить, что каждое предложение – это самостоятельная мысль или идея.

Шаг 2. Токенизация, или выделение слов

Следующий шаг конвейера – выделение отдельных слов или токенов – токенизация. Результат токенизации первого предложения выглядит так:

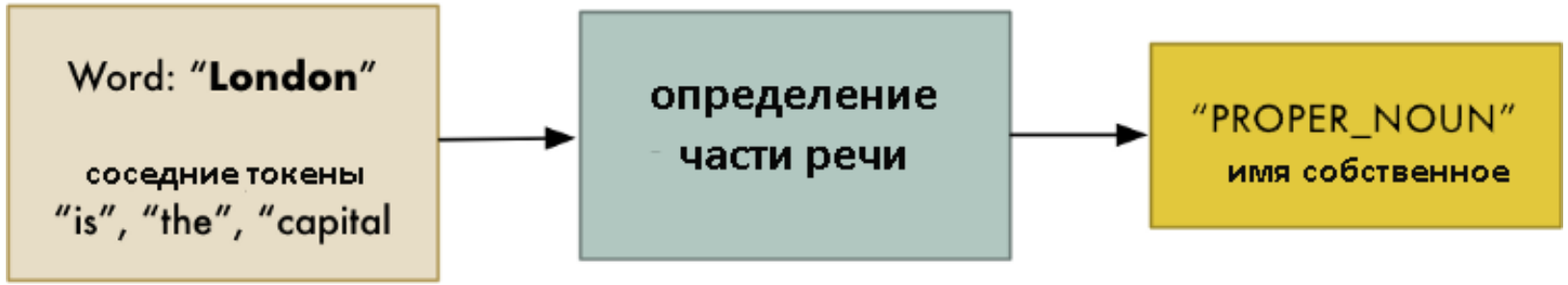
«London», «is», «the», «capital», «and», «most», «populous», «city», «of», «England», «and», «the», «United», «Kingdom», «.»

В английском языке это несложно. Мы просто отделяем фрагмент текста каждый раз, когда встречаем пробел. Знаки препинания тоже являются токенами, поскольку могут иметь важное значение.

Шаг 3. Определение частей речи

Теперь посмотрим на каждый токен и постараемся угадать, какой частью речи он является: существительным, глаголом, прилагательным или чем-то другим. Зная роль каждого слова в предложении, можно понять его общий смысл.

На этом шаге мы будем анализировать каждое слово вместе с его ближайшим окружением с помощью предварительно подготовленной **классификационной модели**:



Эта модель была обучена на миллионах английских предложений с уже обозначенными частями речи и теперь способна их распознавать. Этот анализ основан на статистике – на самом деле модель не понимает смысла слов, вложенного в них человеком. Она просто знает, как угадать часть речи, основываясь на похожей структуре предложений и ранее изученных токенах.

После обработки получаем следующий результат:

С этой информацией уже можно начинать анализировать смысл. Например, мы видим существительные «London» и «capital», вероятно, в предложении говорится о Лондоне.

Шаг 4. Нормализация токенов

Нормализовать токены можно с помощью **стемминга и лемматизации**

Stemming: Porter vs Lancaster

Porter stemmer

- Published in 1979
- Base starting option

Snowball stemmer (Porter 2)

- Based on Porter
- More aggressive
- Most popular option now

Lancaster stemmer

- Published in 1990
- The most aggressive
- Easy adding of your own rules

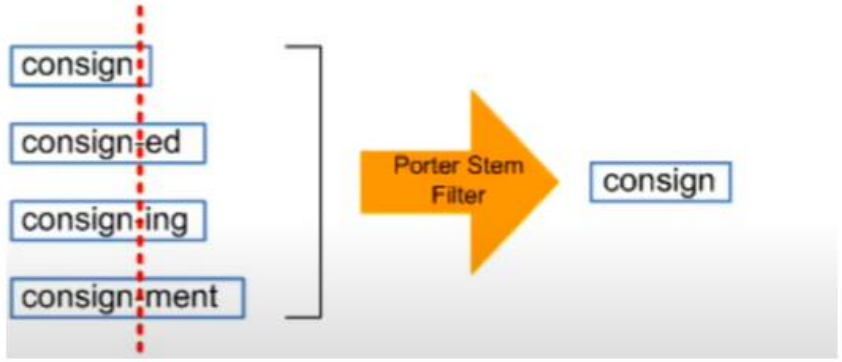
17

Стемминг

Porter's stemmer:

- **Heuristics, applied one-by-one:**
  - SSES - SS (dresses - dress)
  - IES - I (ponies - poni)
  - S - <empty> (dogs - dog)
- **What's wrong?**
  - **Overstemming and understemming**

Stemming: removing and replacing suffixes to get to the root of the word (**stem**)



Проблеммы стемминга

Overstemming

- University
- Universal
- Universities
- Universe

Univers

Understemming

Data → dat

Datum → datu

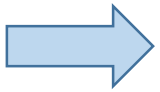
Лемматизация

В английском и большинстве других языков слова могут иметь различные формы:

- I had a pony.
- I had two ponies.

*Оба предложения содержат существительное «pony», но с разными окончаниями. Компьютер должен **знать основную форму каждого слова**, чтобы понимать, что речь идет об одной и той же концепции пони. Иначе токены «pony» и «ponies» будут восприняты как совершенно разные.*

В NLP этот процесс называется **лемматизацией** – нахождением основной формы (леммы) каждого слова в предложении. *Лемматизация обычно выполняется простым поиском форм в таблице.*



глаз	глаза	тут	тута
глаз	глазу	тут	туту
глаз	глазом	тут	тутом
глаз	глазе	тут	туте
глаз	глазам	тут	туты
глаз	глазами	тут	тутов
глаз	глазах	тут	тутам

Глаголы можно привести к неопределенной форме.
Таким образом, предложение «I had two ponies» превращается в «I [have] two [pony]».
Вот так выглядит наше предложение после обработки:

Шаг 5. Определение стоп-слов

В английском очень много вспомогательных слов, например, «and», «the», «a». При статистическом анализе текста эти токены создают много шума, так как появляются чаще, чем остальные. *Некоторые NLP пайплайны отмечают их как стоп-слова и отсеивают перед подсчетом количества.*

Теперь наше предложение выглядит следующим образом:

Для обнаружения стоп-слов обычно используются готовые таблицы. Однако нет единого стандартного списка, подходящего в любой ситуации. Игнорируемые токены могут меняться, все зависит от особенностей проекта.

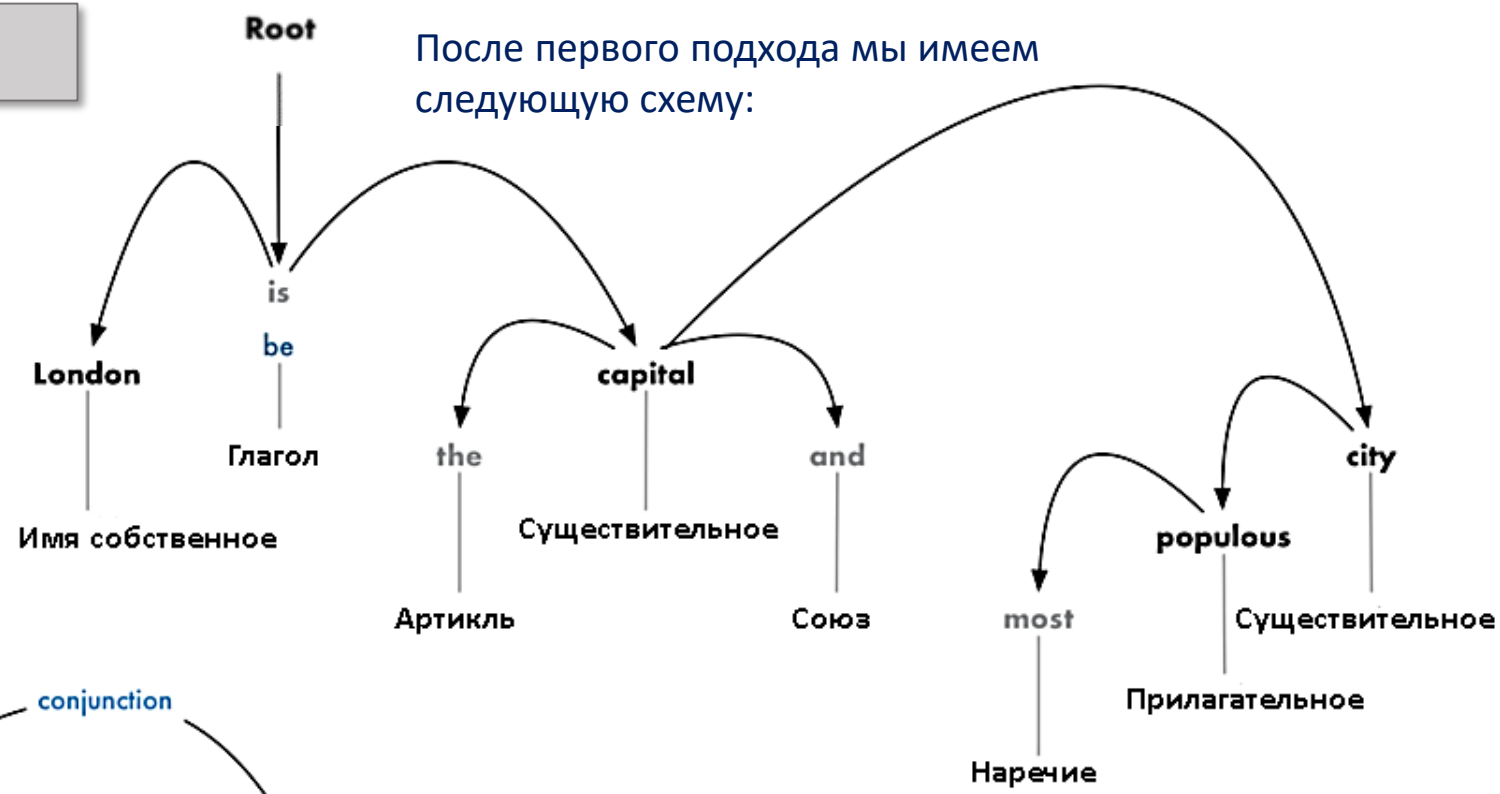
Например, если вы решите создать движок для поиска рок-групп, вероятно, вы не станете игнорировать артикль «the». Он встречается в названии множества коллективов, а одна известная группа 80-х даже называется «The The!».

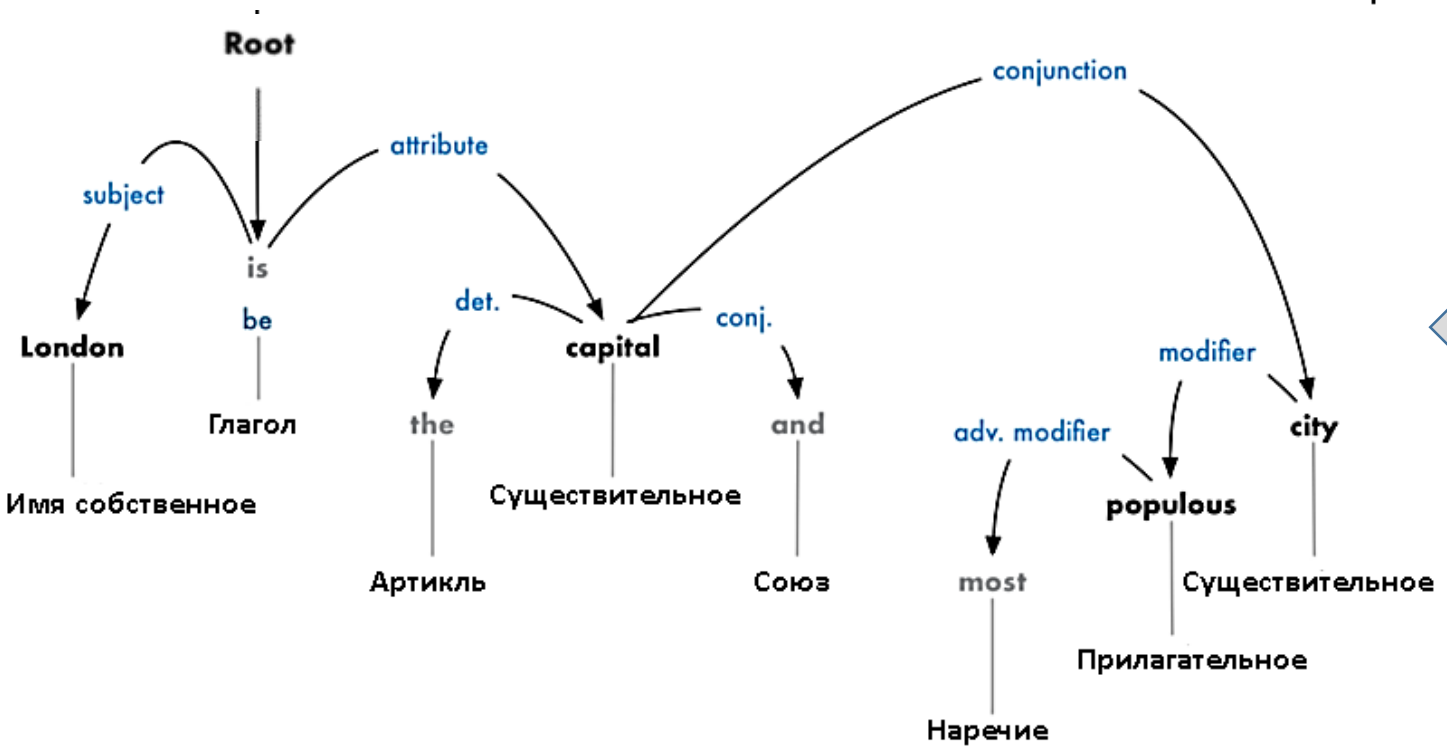
Шаг 6. Парсинг зависимостей

Теперь необходимо установить взаимосвязь между словами в предложении. Это называется **парсингом зависимостей**.

Конечная цель этого шага – построение дерева, в котором каждый токен имеет единственного родителя. Корнем может быть главный глагол.

После первого подхода мы имеем следующую схему:





Но нужно не только определить родителя, но и установить тип связи между двумя словами.

*Это дерево парсинга демонстрирует, что главный субъект предложения – это существительное **«London»**. Между ним и **«capital»** существует связь **«be»**. Вот так мы узнаем, что Лондон – это столица!*

Шаг 6b. Поиск групп существительных (необязательный)

Сейчас мы рассматриваем каждое слово в нашем предложении как отдельную сущность. Но иногда имеет смысл сгруппировать токены, которые относятся к одной и той же идее или вещи. Мы можем использовать полученное дерево парсинга, чтобы автоматически объединить такие слова.

Например, вместо этого:

Можно получить такой результат:

Это необязательный шаг. Но часто это быстрый и удобный способ упростить предложение, если вместо максимально подробной информации о словах мы стремимся извлечь законченные идеи.

Шаг 7. Распознавание именованных сущностей (Named Entity Recognition, NER)

В нашем предложении присутствуют следующие существительные:

London is the capital and most populous city of England and the United Kingdom.

Некоторые из них обозначают реальные вещи. Например, **«London»**, **«England»** и **«United Kingdom»** – это географические объекты. При помощи NLP можно автоматически извлекать из документа список реальных объектов.

Цель распознавания именованных сущностей – обнаружить такие существительные и связать их с реальными концепциями. После обработки каждого токена **NER-моделью** наше предложение будет выглядеть вот так:

London is the capital and most populous city of England and the United Kingdom.

Географическая сущность

Географическая сущность

Географическая сущность

Большинство NER-моделей распознают следующие типы объектов:

- имена людей;
- названия компаний;
- географические обозначения
- физические обозначения
- политические обозначения;
- продукты;
- даты и время;
- денежные суммы;
- события.

Так как NER модели позволяют легко извлекать из сплошного текста структурированные данные, они очень активно используются в разных областях.

Шаг 8. Разрешение кореференции

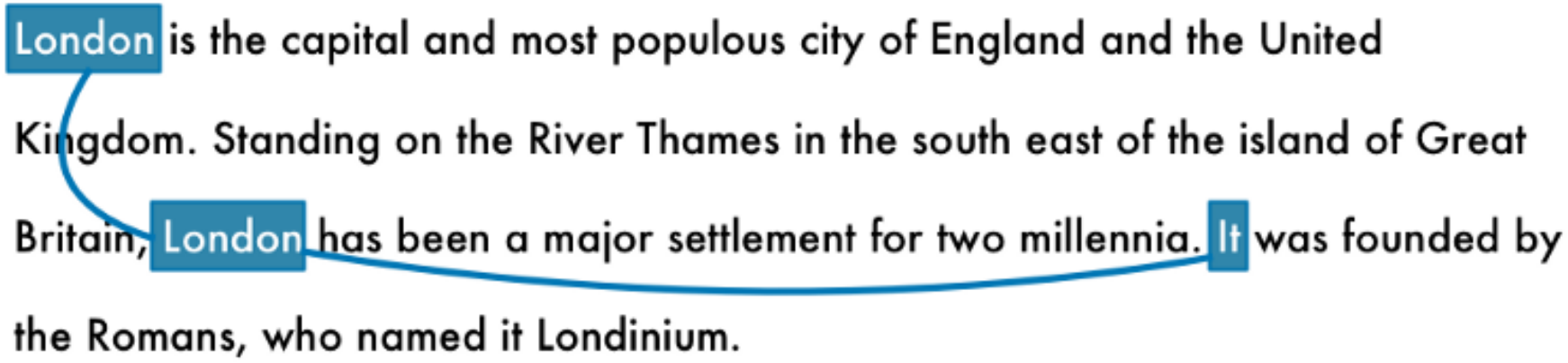
В английском очень много местоимений – слов вроде he, she, it. Это сокращения, которыми мы заменяем на письме настоящие имена и названия. Человек может проследить взаимосвязь этих слов от предложения к предложению, основываясь на контексте. Но NLP-модель не знает о том, что означают местоимения, ведь она рассматривает всего одно предложение за раз.

КОРЕФЕРЕНЦИЯ (англ. coreference) – вид текстовой или синтаксической связности, при которой две или более номинативных (именных) групп называют один и тот же объект (референт).

Давайте посмотрим на третью фразу в нашем документе:

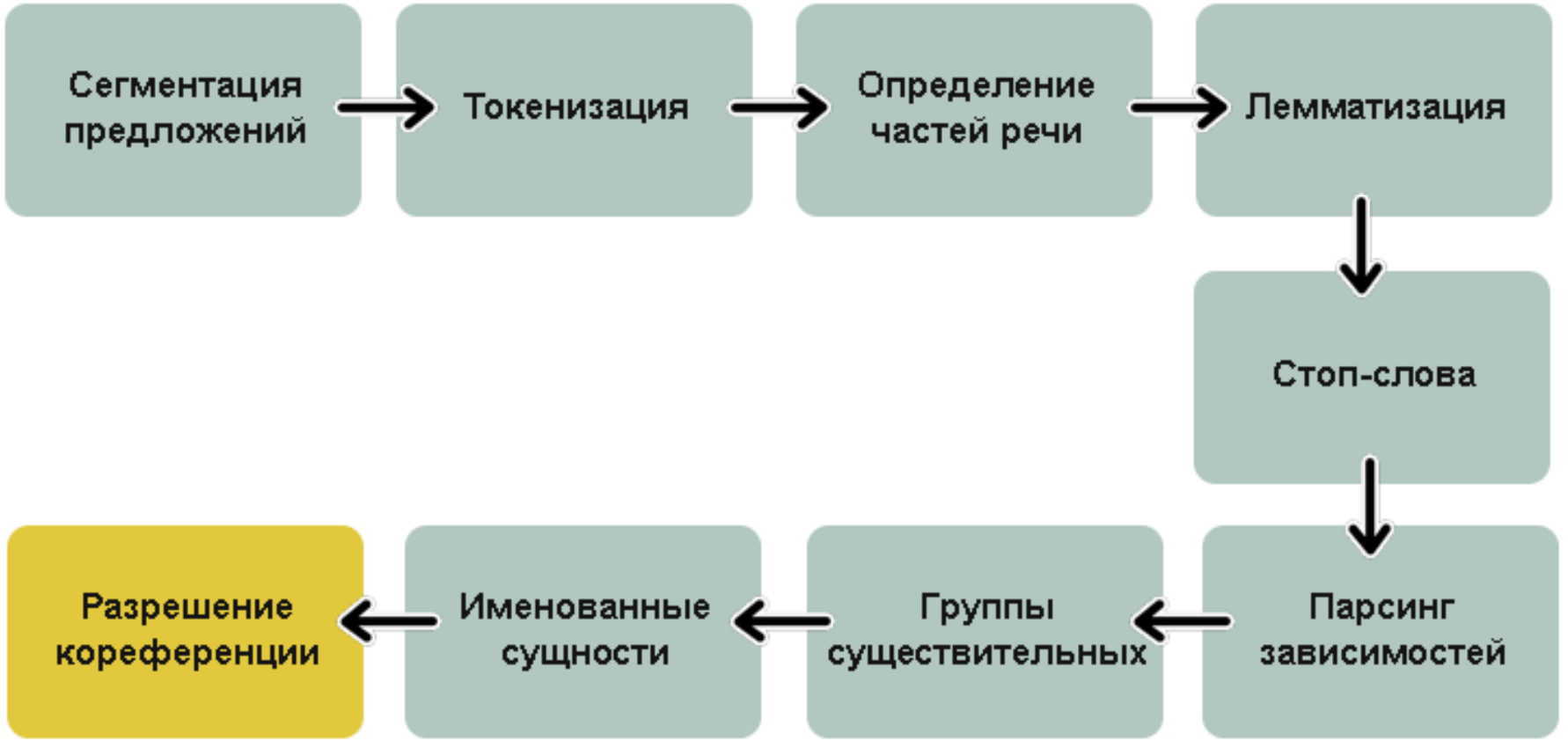
It was founded by the Romans, who named it Londinium.

В процессе чтения мы догадаемся, что «это» не что иное как Лондон. Разрешением кореференции называется отслеживание местоимений в предложениях с целью выбрать все слова, относящиеся к одной сущности.
Вот результат обработки документа для слова **«London»**:



Разрешение кореференции – один из самых трудных шагов в нашем пайплайне. В области глубокого обучения уже появились способы его реализации, они достаточно точны, но все еще не совершенны.

Резюмирующая схема рассмотренного паплайна выглядит так:



Это стандартные этапы обычного NLP-паплайна, но в зависимости от конечной цели проекта и особенностей реализации модели, некоторые из них можно пропускать или менять местами.

Например, **spaCy** производит сегментацию позже, используя для нее результаты парсинга зависимостей.

Выделение именованных сущностей (Named Entity Recognition, NER)

Задача NER – выделить спаны сущностей в тексте (спан – непрерывный фрагмент текста).
Допустим, есть новостной текст, и мы хотим выделить в нем сущности (некоторый заранее зафиксированный набор — например, персоны, локации, организации, даты и так далее). Задача NER – понять, что участок текста “1 января 1997 года” является датой, а “Кофи Аннан” – персоной.

NER-системы не просто просматривают словари. Они анализируют контекст токена в предложении и используют статистические модели, чтобы угадать какой объект он представляет. *Хорошие NER-системы способны отличить актрису Brooklyn Decker от города Brooklyn.*

Так как NER модели позволяют легко извлекать из сплошного текста структурированные данные, они очень активно используются в разных областях (след. слайд).

Извлечение именованных сущностей (Named Entity Recognition, NER) — это шаг в сторону “понимания” текста. Это может как иметь самостоятельную ценность, так и помочь лучше решать другие задачи NLP.

- **Структуризация неструктурированных данных**. *Например, есть какой-то текст (или набор текстов), и данные из него нужно ввести в базу данных (таблицу). Классические именованные сущности могут соответствовать строкам такой таблицы или же служить содержанием каких-то ячеек.*
- **Локализация важных фрагментов текста**. *Например, можем выделить только те абзацы, где встречаются сущности какого-то определенного типа, а потом работать только с ними.*
- **Сущности – это жесткие коллокации** и действия с ними нужно совершать как с единым блоком. *Например, переводить название сущности единым куском, а не разбить на несколько не связанных фрагментов*.
- Определение коллокации полезно для **синтаксического парсинга**.
- **Разрешение кореференции**. *Например, пусть мы хотим проанализировать текст “Прискакал Чарминг на белом коне. Принцесса выбежала ему навстречу и поцеловала его”. Если мы выделили на слове “Чарминг” сущность Персона, то машина сможет намного легче понять, что принцесса, скорее всего, поцеловала не коня, а принца Чарминга.*
- **При построении вопросно-ответных систем**. *Если задать в вашем любимом поисковике вопрос «Кто играл роль Дарта Вейдера в фильме “Империя наносит ответный удар”», то с большой вероятностью вы получите верный ответ. Это делается как раз с помощью выделения именованных сущностей: выделяем сущности (фильм, роль и т. п.), понимаем, что нас спрашивают, и дальше ищем ответ в базе данных.*

Коллокация — словосочетание, имеющее признаки синтаксически и семантически целостной единицы

https://habr.com/ru/company/abbyy/blog/449514/

Важная особенность, благодаря которому задача NER так популярна: постановка задачи очень гибкая. Другими словами, никто не заставляет нас выделять именно локации, персоны и организации. **Мы можем выделять любые нужные нам непрерывные фрагменты текста, которые чем-то отличаются от остального текста.** В результате можно подобрать свой набор сущностей для конкретной практической задачи, приходящей от заказчика, разметить корпус текстов этим набором и обучить модель. Такой сценарий встречается повсеместно, и это делает NER одной из самых часто решаемых задач NLP в индустрии.

Примеры таких юзкейсов от конкретных заказчиков:

Вот первый из них: пусть у вас есть набор инвойсов (денежных переводов). Каждый инвойс имеет текстовое описание, где содержится необходимая информация о переводе (кто, кому, когда, что и по какой причине отправил). Например, компания X перевела 10 долларов компании Y в такую-то дату таким-то образом за то-то. Текст довольно формальный, но пишется живым языком.

Мы можем выбрать набор сущностей, которые соответствуют столбцам таблицы в базе данных (названия компаний, сумма перевода, его дата, тип перевода и т. п.) и научиться автоматически их выделять. После этого остается только занести выделенные сущности в таблицу.

Второй юзкейс такой: нужно анализировать письма с заказами из интернет-магазинов. Для этого необходимо знать номер заказа (чтобы все письма, относящиеся к данному заказу, помечать или складывать в отдельную папку), а также другую полезную информацию — название магазина, список товаров, которые были заказаны, сумму по чеку и т. п. Все это — номера заказа, названия магазинов и т. п. — можно считать именованными сущностями и научиться выделять.

https://habr.com/ru/company/abbyy/blog/449514/

Инструменты для NLP

NLTK (Natural Language ToolKit) — самая известная NLP библиотека, созданная исследователями в данной области. Она популярна в академических кругах и в основном используется для обучения или создания различных методов обработки используя базовые инструменты, которые NLTK предоставляет в огромном количестве и не всегда самые лучшие. Так же она довольно медленная в силу того, что написана полностью на Python и работает со строками.

SpaCy — в каком-то смысле противоположность NLTK. Она значительно быстрее, так как она написана на Cython и работает с объектами, об этом дальше. SpaCy предоставляет в основном лучшие инструменты для решения конкретной задачи. Она — numpy из мира NLP.

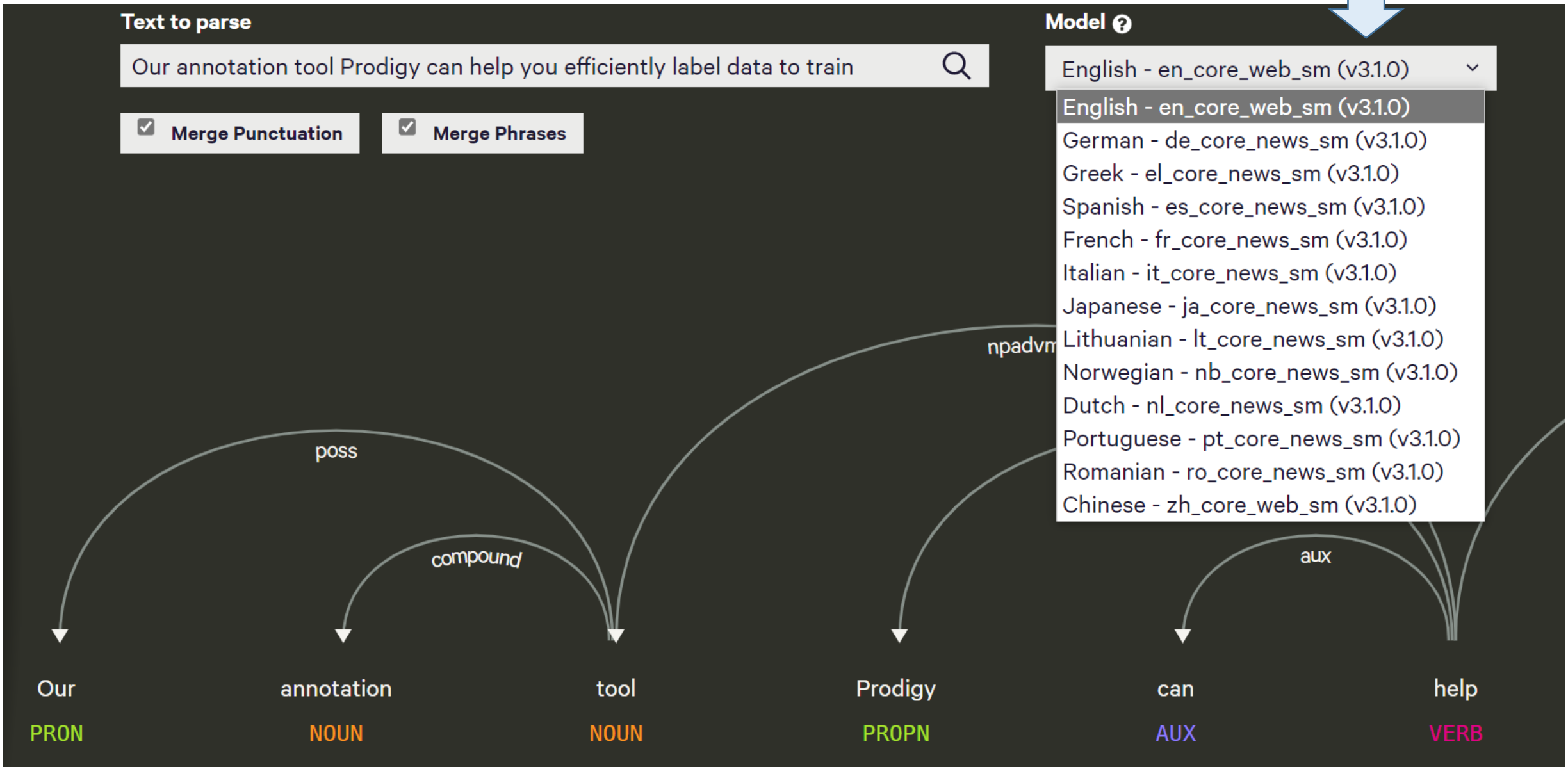
В целом, ***SpaCy*** *с её предобученными моделями, скоростью, удобным API и абстракцией гораздо* ***лучше подходит для разработчиков, создающих готовые решения****, а* ***NLTK*** *с огромным числом инструментов и возможностью городить любые огороды —* ***для исследователей и студентов****. В любом случае для создания собственных моделей ни та, ни другая библиотека не подходит. Для этого всего существуют Tensorflow, PyTorch и прочие.*

Если кратко, то SpaCy может примерно всё то же самое, что и NLTK и их аналоги, но быстрее и точнее.

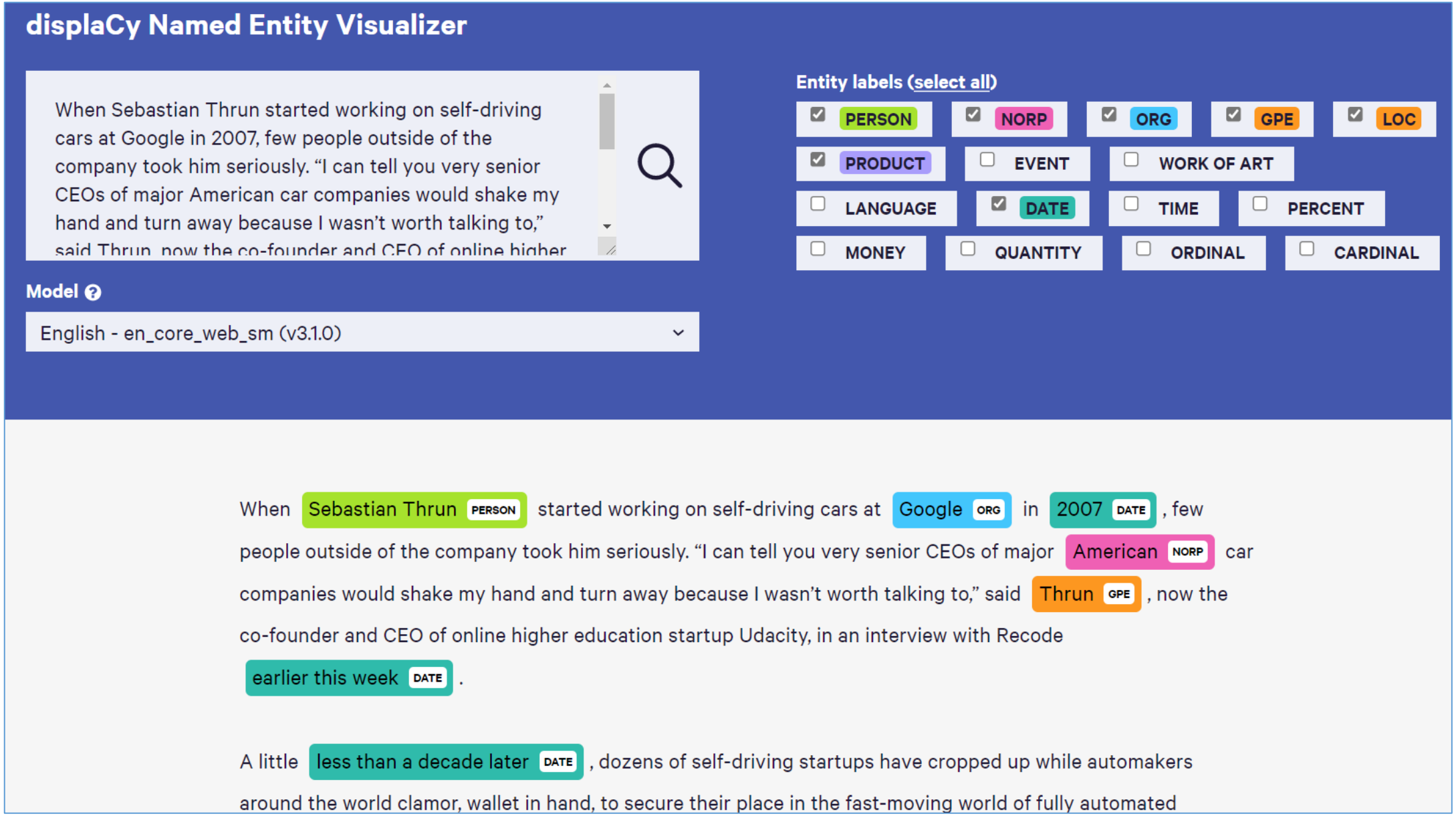
Онлайн визуализатор

displaCy Dependency Visualizer · Explosion

Модели для различных языков



[displaCy Named Entity Visualizer · Explosion]



Пример кода

```python
import spacy
from spacy.lang.en.examples import sentences

# Загрузка английской NLP-модели
nlp = spacy.load("en_core_web_sm")

# Текст для анализа
text = "London is the capital and most populous city of England and the United Kingdom."

# Парсинг текста с помощью spaCy. Эта команда запускает целый конвейер
doc = nlp(text)

print(doc.text)

# текст токена, начальная форма, часть речи, является ли стоп-словом
for token in doc:
    print(token.text, token.lemma_, token.pos_, token.is_stop)
```

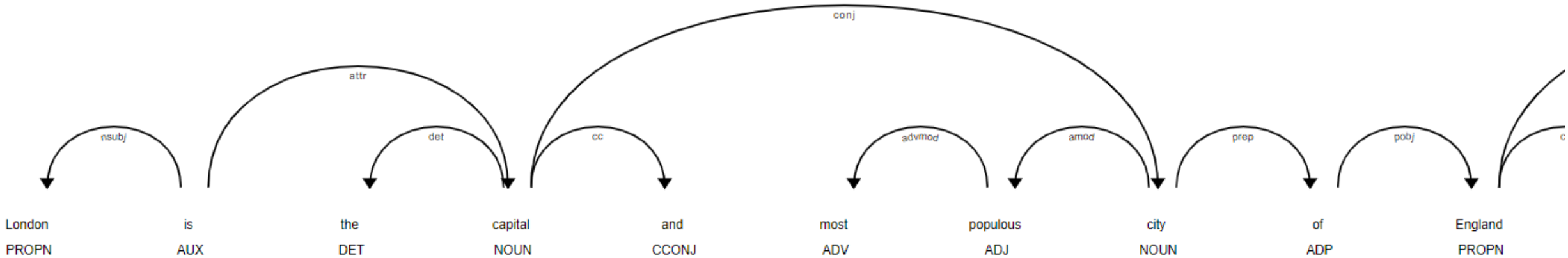
```
London London PROPN False
is be AUX True
the the DET True
capital capital NOUN False
and and CCONJ True
most most ADV True
populous populous ADJ False
city city NOUN False
of of ADP True
England England PROPN False
and and CCONJ True
the the DET True
United United PROPN False
Kingdom Kingdom PROPN False
```

```python
# Построение дерева зависимостей
#(текст токена, тип зависимости(согласно Universal Dependency), корневое слово)

for token in doc:
    print(token.text, token.dep_, token.head)
```

```
London nsubj is
is ROOT is
the det capital
capital attr is
and cc capital
most advmod populous
populous amod city
city conj capital
of prep city
England pobj of
and cc England
the det Kingdom
United compound Kingdom
Kingdom conj England
. punct is
```

```python
# визуализации дерева зависимостей, а также распознанных сущностей
from spacy import displacy
displacy.render(doc, style='dep', jupyter=True) # первые 10 doc[:10]
```




```python
# для расшифровки названий тегов можно воспользоваться функций explain:
print(spacy.explain("AUX"))
print(spacy.explain("CCONJ"))
```

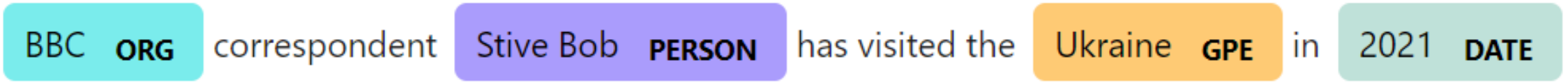
```
auxiliary
coordinating conjunction
```

Распознавание именованных сущностей (Named Entity Recognition, NER)

```
 doc2=nlp("BBC correspondent Stive Bob has visited the Ukraine in 2021.")

for ent in doc2.ents:
    print(ent.text, ent.label_)
displacy.render(doc2, style='ent', jupyter=True)
```

```
BBC ORG
Stive Bob PERSON
Ukraine GPE
2021 DATE
```



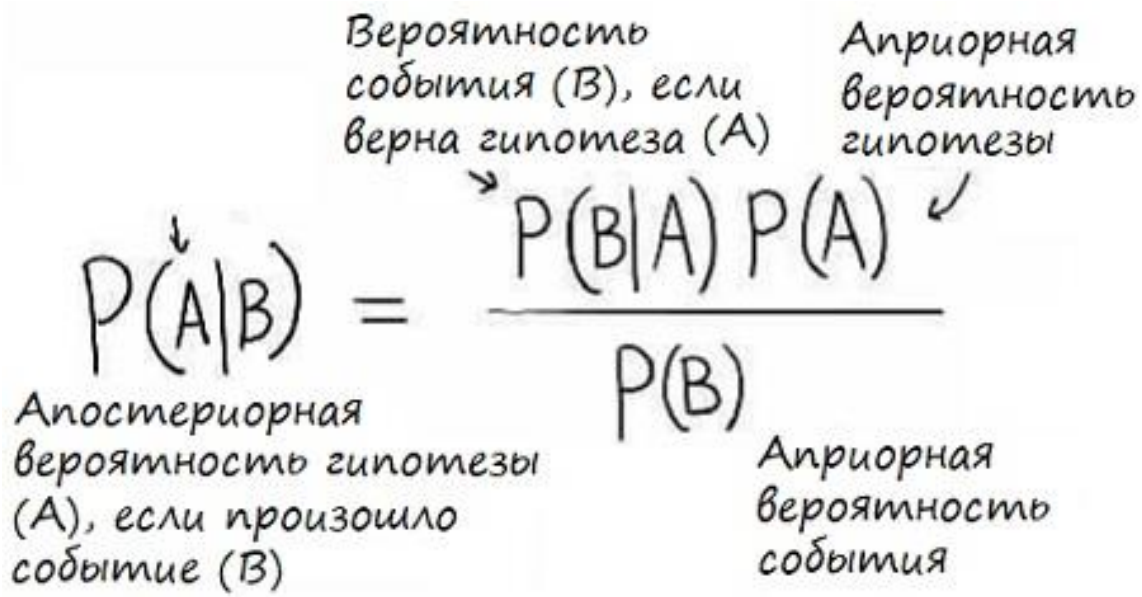

```
print(spacy.explain("GPE"))
```

```
Countries, cities, states
```

https://baguzin.ru/wp/den-morris-teorema-bajesa-vizualnoe-vvedenie-dlya-nachinayushhih/

Баесовские спам фильтры

Фильтрация спама значительно улучшилась за последнее десятилетие до такой степени, что большинство из нас больше не думают о спаме. В 1998 году Microsoft подала заявку на патент на спам-фильтр, который использовал Байесовский подход. Конкуренты вскоре присоединились к MS, **и теорема Байеса быстро стала основой фильтрации спама.**



Байесовские фильтры определяют, является ли письмо спамом или нет на основе содержимого сообщения. Для каждого слова определяется вероятность того, что оно является спамом или нет. Что отличает Байесовские фильтры от других фильтров электронной почты? Первые учатся и адаптируются к каждому отдельному пользователю. Вот почему они настолько эффективны.

https://baguzin.ru/wp/den-morris-teorema-bajesa-vizualnoe-vvedenie-dlya-nachinayushhih/

Спам-фильтры, построенные на Байесовской сети, как правило, предварительно заполняются списком потенциальных слов и характеристик, которые содержит спам. *Вспомните, какие слова всплывают у вас в памяти, когда вы думаете о спаме? Обычно, все, что связано с сексом, сделками, секретами, выигрышами… Этот список можно продолжить.* **Фильтр анализирует слова в теле сообщения, в заголовке, в метаданных**. Этот список постоянно обновляется по мере получения каждого письма, и фильтр узнает все больше и больше о том, что искать.

Фильтр учится двумя разными способами:

- на основе собственных решений;
- на основе решений пользователя (вы проверяете электронную почту, и отправляете часть писем в спам).

Например, если слово спорт часто появляется в вашей электронной почте, фильтр может заключить, что оно имеет очень низкую вероятность быть спамом (по шкале от 0 до 1, может быть. 0,1). Однако, если вы не интересуетесь спортом, фильтр может заключить, что письма с этим словом имеют высокую вероятность быть спамом (например, 0,65). Вот как теорема Байеса может быть использована при обнаружении спама:

P (спам | определенное слово) = P (опр. слово | спам) * P (спам) / P (опр. слово)

Почтовые байесовские фильтры основываются на теореме Байеса. Теорема Байеса используется несколько раз в контексте спама:

- в первый раз, чтобы вычислить вероятность, что сообщение — спам, зная, что данное слово появляется в этом сообщении;
- во второй раз, чтобы вычислить вероятность, что сообщение — спам, учитывая все его слова (или соответствующие их подмножества);
- иногда в третий раз, когда встречаются сообщения с редкими словами.

Данный метод прост (алгоритмы элементарны), **удобен** (позволяет обходиться без «чёрных списков» и подобных искусственных приёмов), **эффективен** (после обучения на достаточно большой выборке отсекает до 95—97 % спама), причём в **случае любых ошибок его можно дообучать**. В общем, есть все показания для его повсеместного использования, что и имеет место на практике — на его основе построены практически все современные спам-фильтры.

Но есть и принципиальный недостаток: он базируется на предположении, что одни слова чаще встречаются в спаме, а другие — в обычных письмах, и неэффективен, если данное предположение неверно. Впрочем, как показывает практика, такой спам даже человек не в состоянии определить «на глаз» — только прочтя письмо и поняв его смысл.

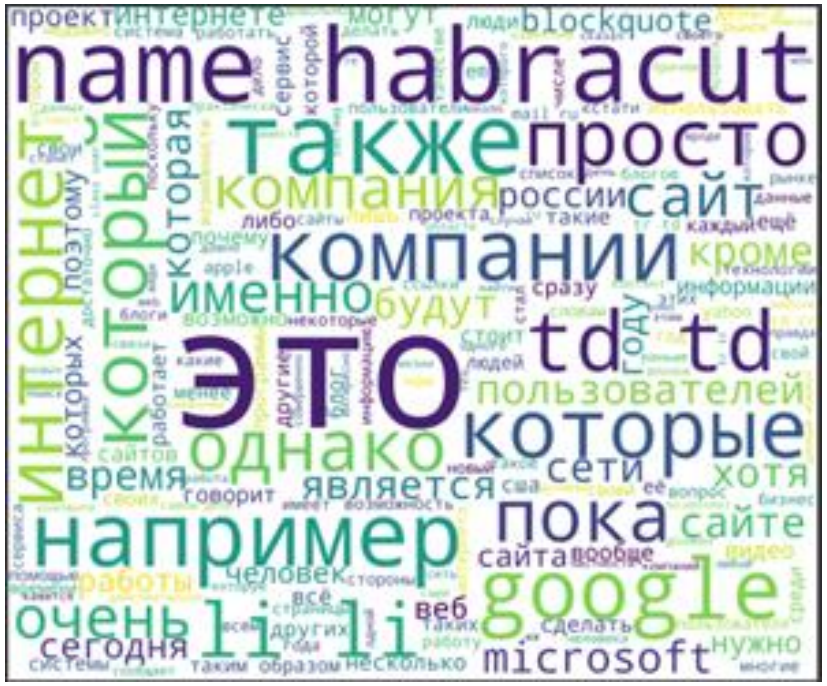
Ещё один не принципиальный недостаток, связанный с реализацией — метод работает только с текстом. Зная об этом ограничении, спамеры стали вкладывать рекламную информацию в картинку.

РЕШАЕМ NLP-ЗАДАЧУ – КЛАССИФИКАЦИЯ ТЕКСТОВ ПО ТЕМАМ

https://newtechaudit.ru/reshaem-nlp-zadachu-klassifikacziya-tekstov-po-temam/

```
from wordcloud import WordCloud
```

Для необработанного набора данных «облако слов» содержит 243024 уникальных слова и выглядит так:



После очистки текстов от «стоп-слов» и знаков пунктуации количество уникальных слов снизилось до 142253, а «облако слов» стало более осмысленным: