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.

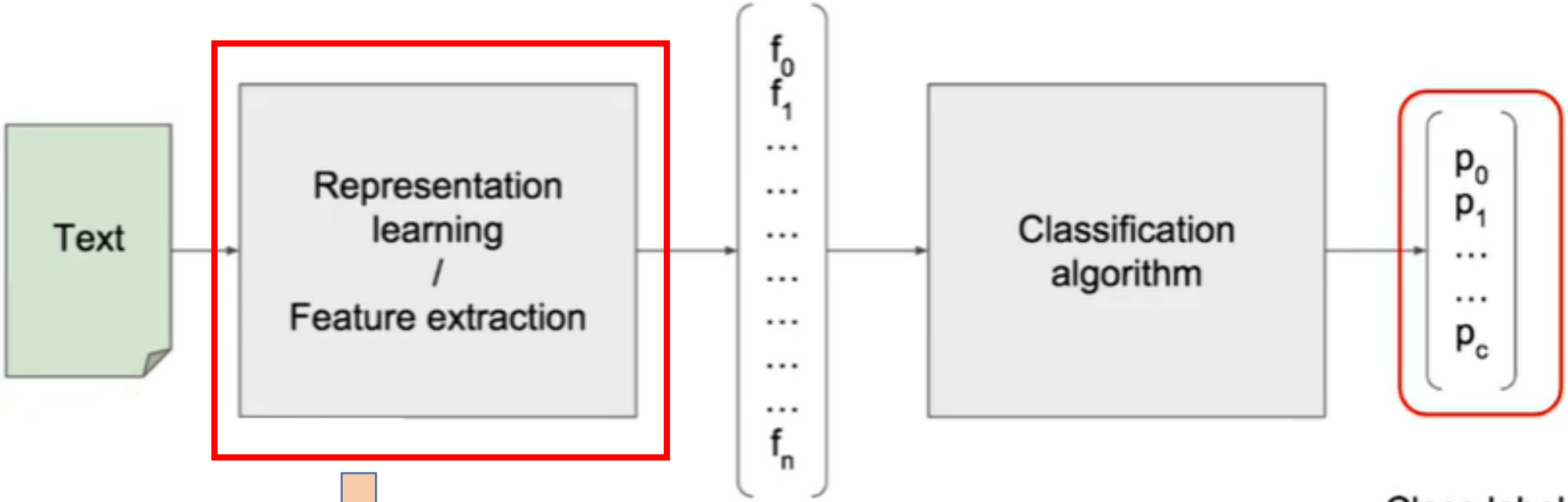
Natural Language Processing (далее – NLP) – обработка естественного языка – подраздел информатики и AI, посвященный тому, как компьютеры анализируют естественные (человеческие) языки.
NLP позволяет применять алгоритмы машинного обучения для текста и речи.

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.

Классификация намерений

узнать_имя	как тебя зовут? скажи свое имя с кем я общаюсь? я Ксюша, а ты? ...
откуда_номер	откуда вы мой номер узнали вы где мой телефон взяли? кто вам дал этот номер? откуда у вас этот номер телефона ...

Text classification in general



Буквы ?
Слова ?
Словосочетания ?

- Discrete labels:
  - Binary
    - spam filtering, sentiment analysis
  - Multi-class
    - categorization of items by its description
  - Multi-label
    - #hashtag prediction

Векторное представление текста

Пример векторизации: подсчет количества слов из словаря

	вы	откуда	телефон	номер	...
откуда вы мой номер узнали	1	1	0	1	...
вы где мой телефон взяли?	1	0	1	0	...
откуда у вас этот номер телефона	1	1	1	1	...
...					...
как вас зовут?	1	0	0	0	...

Bag of words (BOW)

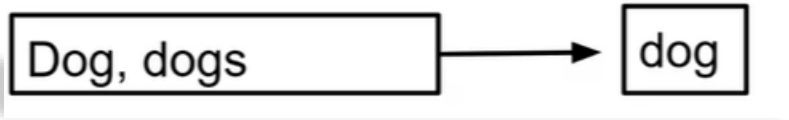
	the	red	dog	cat	eats	food
1. the red dog →	1	1	1	0	0	0
2. cat eats dog →	0	0	1	1	1	0
3. dog eats food →	0	0	1	0	1	1
4. red cat eats →	0	1	0	1	1	0

Problems:

- No information about words order
- Word vectors are huge and very sparse
- Word vectors are not normalized
- Same words can take different forms

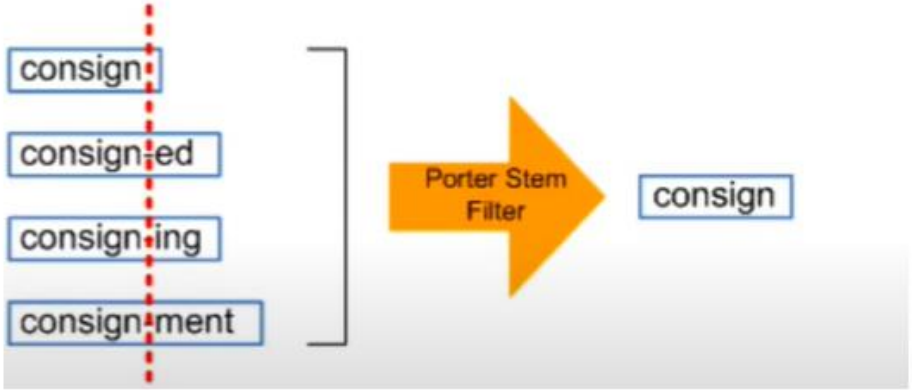
Предобработка текста (Preprocessing)

Token normalization



Нормализовать токены (слова) можно с помощью стемминга и лемматизации

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



Проблемма стемминга

Overstemming

- University
- Universal
- Universities
- Universe

Univers

- **Lemmatization**: to get base or dictionary form of a word (**lemma**) Based on **WordNet** database

список лемматизации

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

Потерян порядок слов
Как исправить?

Bag-of-Words

- How to improve BOW?
  - Use n-gramms instead of words!

The brown dog plays with a little cat →

The brown
brown dog
dog plays
plays with
with a
a little
little cat

Удаляем n-gramms по правилам

The brown dog plays with a little cat →

brown dog
dog plays
little cat

Meaningful n-gramms

Meaningful n-gramms are often called **collocations**

Delete:
- High-frequency n-gramms
  - Articles, prepositions
  - Auxiliary verbs (to be, to have, etc.)
  - General vocabulary
- Low-frequency n-gramms
  - Typos
  - Combinations that occur 1-2 times in a text

TF-IDF (от англ. TF — term frequency, IDF — inverse document frequency) – **статистическая мера, используемая для оценки важности слова в контексте документа**, являющегося частью коллекции документов. Вес некоторого слова пропорционален частоте употребления этого слова в документе и обратно пропорционален частоте употребления слова во всех документах коллекции.

*Мера TF-IDF часто используется в задачах анализа текстов и информационного поиска, например, как **один из критериев релевантности документа поисковому запросу, при расчёте меры близости документов при кластеризации.***

TF (*term frequency* — частота слова) — отношение числа вхождений некоторого слова к общему числу слов документа. Таким образом, оценивается важность слова t_i в пределах отдельного документа.

$$\mathrm{tf}(t,d) = \frac{n_t}{\sum_k n_k},$$

где n_t есть число вхождений слова t в документ, а в знаменателе — общее число слов в данном документе.

IDF (inverse document frequency — обратная частота документа) — инверсия частоты, с которой некоторое слово встречается в документах коллекции.

Учёт **IDF уменьшает вес широкоупотребительных слов.** Для каждого уникального слова в пределах конкретной коллекции документов существует только одно значение IDF.

$$\operatorname{idf}(t, D) = \log \frac{|D|}{|\{ d_i \in D \mid t \in d_i \}|}$$,[2]

где

- |D| — число документов в коллекции;
- $|\{ d_i \in D \mid t \in d_i \}|$ — число документов из коллекции D, в которых встречается t (когда $n_t \neq 0$).

Таким образом, мера TF-IDF является произведением двух сомножителей:

$$\text{tf-idf}(t, d, D) = \operatorname{tf}(t, d) \times \operatorname{idf}(t, D)$$

Большой вес в TF-IDF получат слова с высокой частотой в пределах конкретного документа и с низкой частотой употреблений в других документах.

Sentence A: The car is driven on the road.

Sentence B: The truck is driven on the highway.

(each sentence is a separate document)

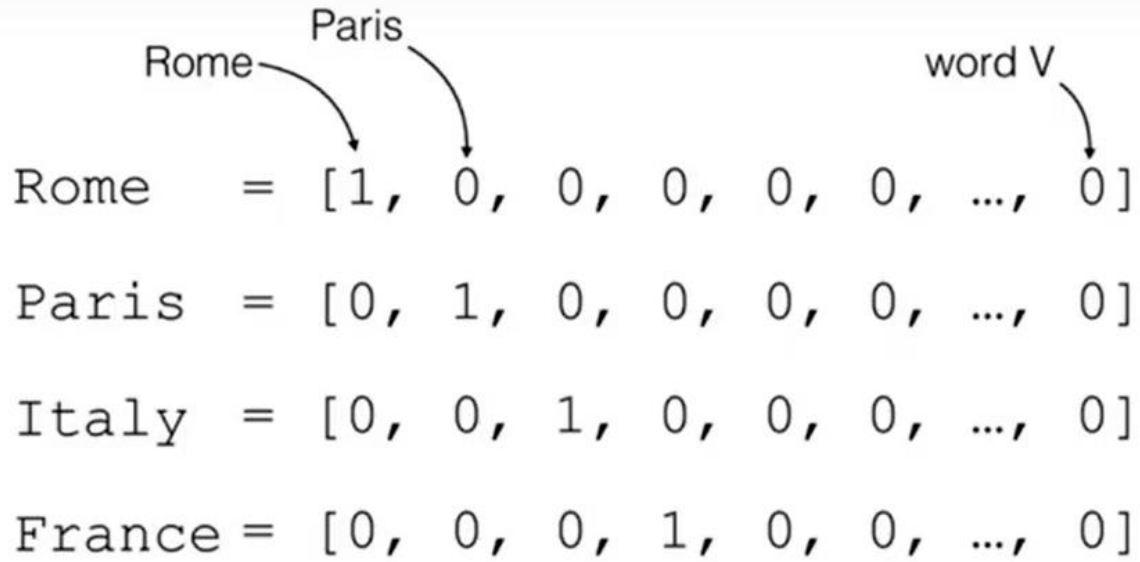
```
from sklearn.feature_extraction.text

import TfidfVectorizer
```

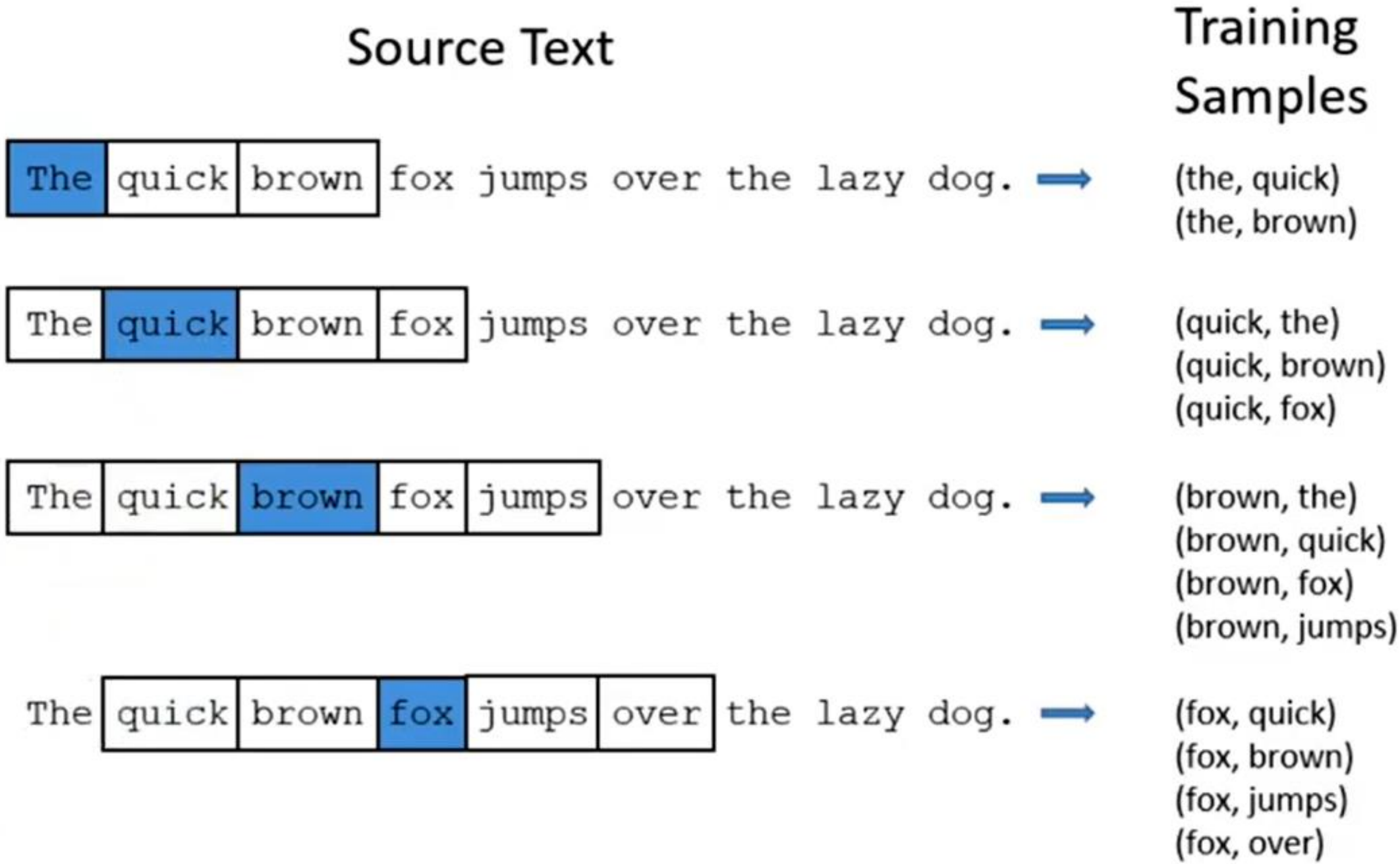
Word	TF		IDF	TF * IDF	
	A	B		A	B
The	1/7	1/7	log(2/2)=0	0	0
Car	1/7	0	log(2/1)=0.3	0.043	0
Truck	0	1/7	log(2/1)=0.3	0	0.043
Is	1/7	1/7	log(2/2)=0	0	0
Driven	1/7	1/7	log(2/2)=0	0	0
On	1/7				
The	1/7				
Road	1/7				
Highway	0				

Вес слова

Мера TF-IDF часто используется для представления документов коллекции в виде числовых векторов, отражающих важность использования каждого слова из некоторого набора слов (количество слов набора определяет размерность вектора) в каждом документе. Подобная модель называется векторной моделью и **даёт возможность сравнивать тексты, сравнивая представляющие их вектора в какой-либо метрике**

One-hot vectors:



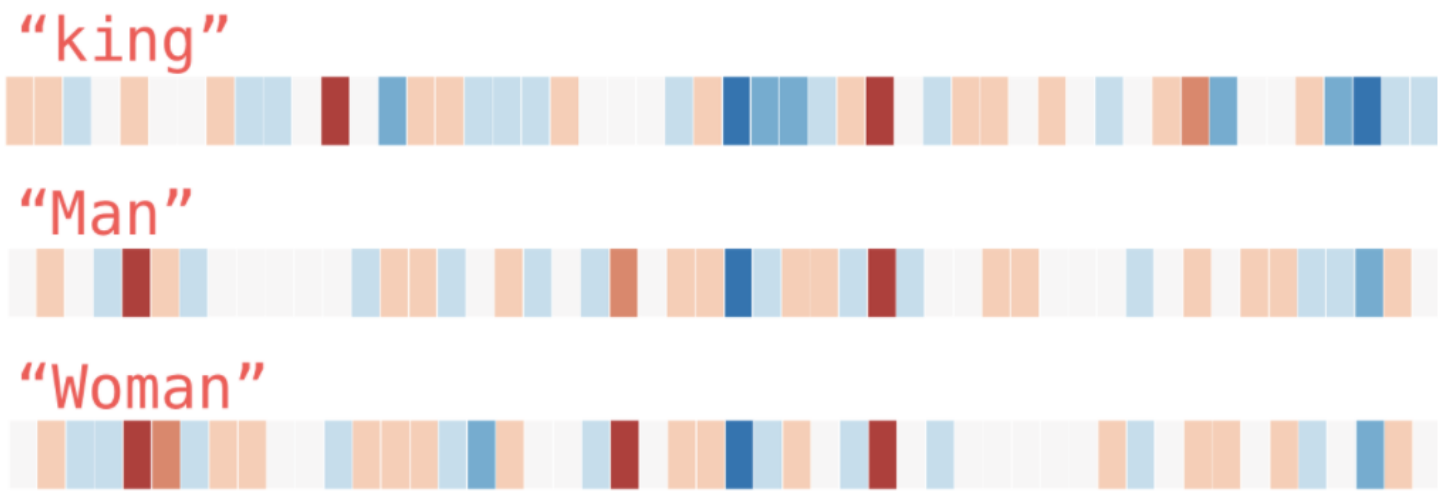

Source Text
Training Samples
The quick brown fox jumps over the lazy dog.
(the, quick)
(the, brown)
The quick brown fox jumps over the lazy dog.
(quick, the)
(quick, brown)
(quick, fox)
The quick brown fox jumps over the lazy dog.
(brown, the)
(brown, quick)
(brown, fox)
(brown, jumps)
The quick brown fox jumps over the lazy dog.
(fox, quick)
(fox, brown)
(fox, jumps)
(fox, over)

Word2vec - совокупности моделей на основе искусственных нейронных сетей, предназначенных для получения векторных представлений слов на естественном языке.

Вот вложение для слова «король» (вектор GloVe, обученный на Википедии):

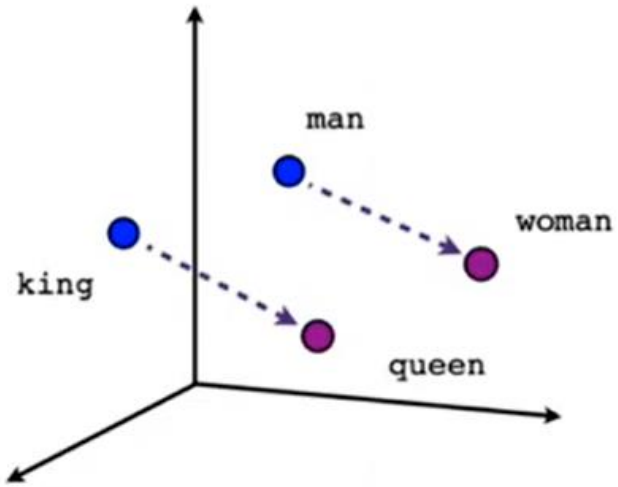
```
[ 0.50451 , 0.68607 , -0.59517 , -0.022801, 0.60046 , -0.13498 , -0.08813 , 0.47377
, -0.61798 , -0.31012 , -0.076666, 1.493 , -0.034189, -0.98173 , 0.68229 , 0.81722 ,
-0.51874 , -0.31503 , -0.55809 , 0.66421 , 0.1961 , -0.13495 , -0.11476 , -0.30344 ,
0.41177 , -2.223 , -1.0756 , -1.0783 , -0.34354 , 0.33505 , 1.9927 , -0.04234 ,
-0.64319 , 0.71125 , 0.49159 , 0.16754 , 0.34344 , -0.25663 , -0.8523 , 0.1661 ,
0.40102 , 1.1685 , -1.0137 , -0.21585 , -0.15155 , 0.78321 , -0.91241 , -1.6106 ,
-0.64426 , -0.51042 ]
```

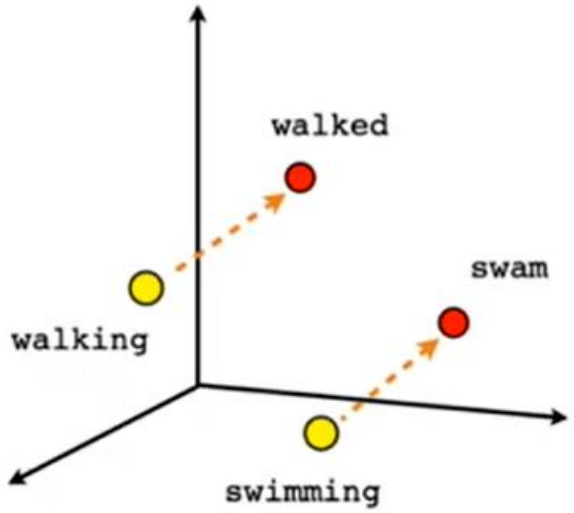
Мы видим список из 50 чисел, но по ним трудно что-то сказать. Давайте их визуализируем, чтобы сравнить с другими векторами

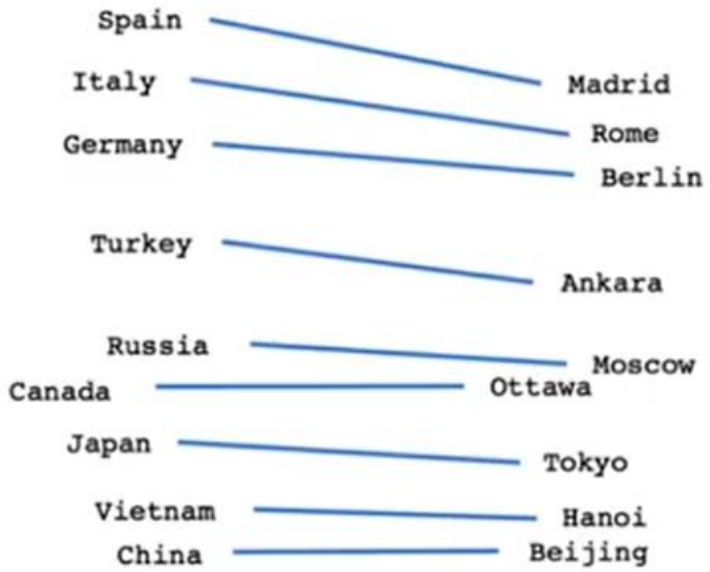


Векторные представления слов, полученным в результате обучения называют **вложениями (embeddings)**.

word embeddings







Male-Female Verb tense Country-Capital

Методы word embeddings реализованы в Gensim. Это библиотека обработки естественного языка предназначения для «Тематического моделирования».
С ее помощью можно обрабатывать тексты, работать с векторными моделями слов (такими как Word2Vec, FastText и т. д.) и создавать тематические модели текстов.

NLP пайплайн шаг за шагом

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:

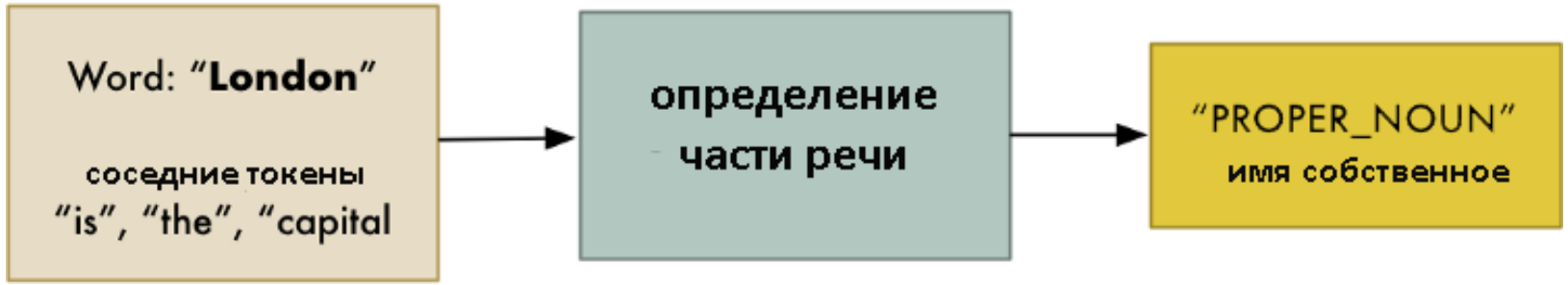
«London», «is», «the», «capital», «and», «most», «populous», «city», «of», «England», «and», «the», «United», «Kingdom», «.»

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. 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.

То же самое относится к глаголам. Мы можем привести их к неопределенной форме. Таким образом, предложение «I had two ponies» превращается в «I [have] two [pony]».

Лемматизация обычно выполняется простым поиском форм в таблице. Кроме того, можно добавить некоторые пользовательские правила для анализа слов.

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.

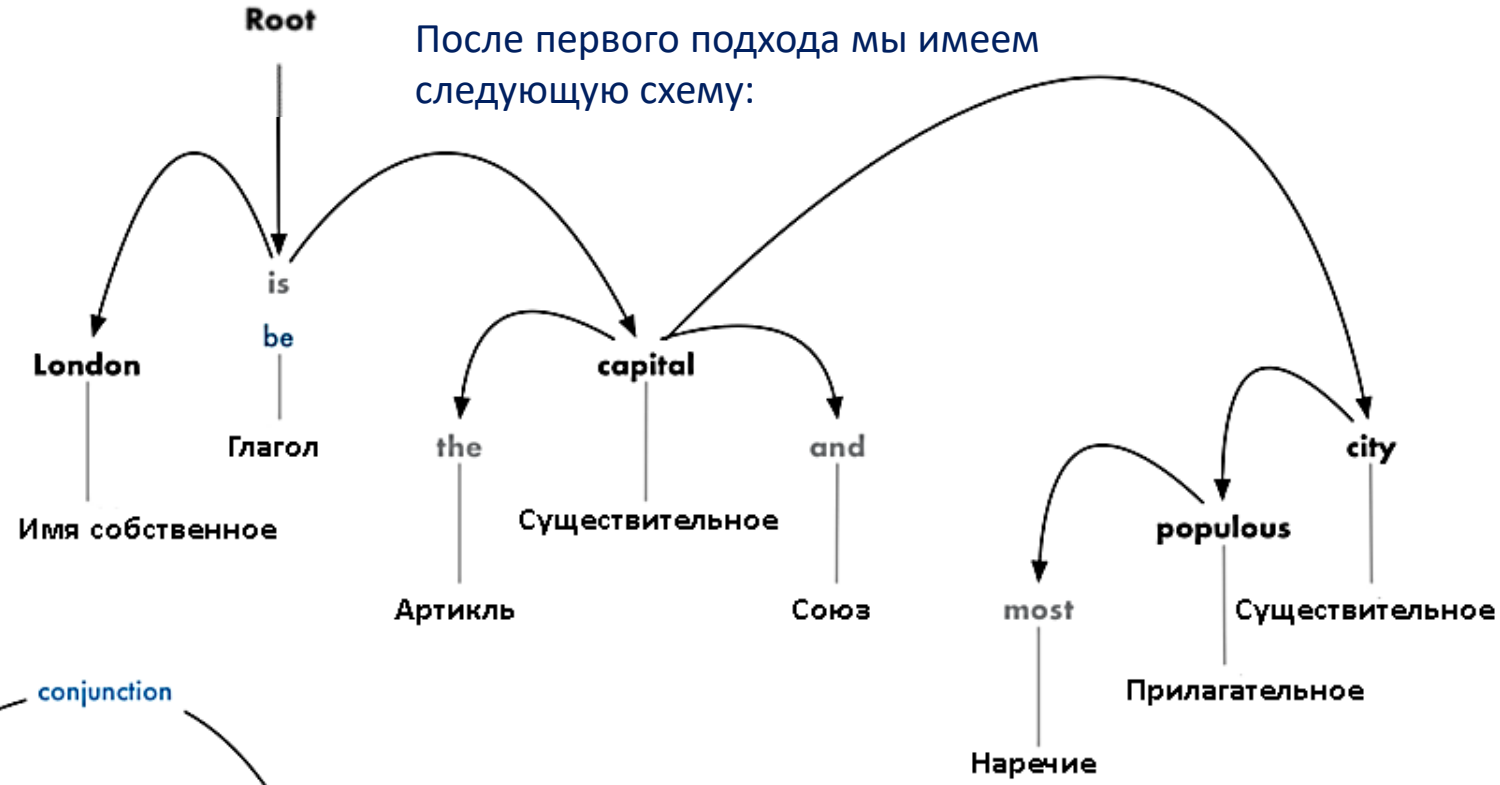
Например, если вы решите создать движок для поиска рок-групп, вероятно, вы не станете игнорировать артикль «the». Он встречается в названии множества коллективов, а одна известная группа 80-х даже называется «The The!».

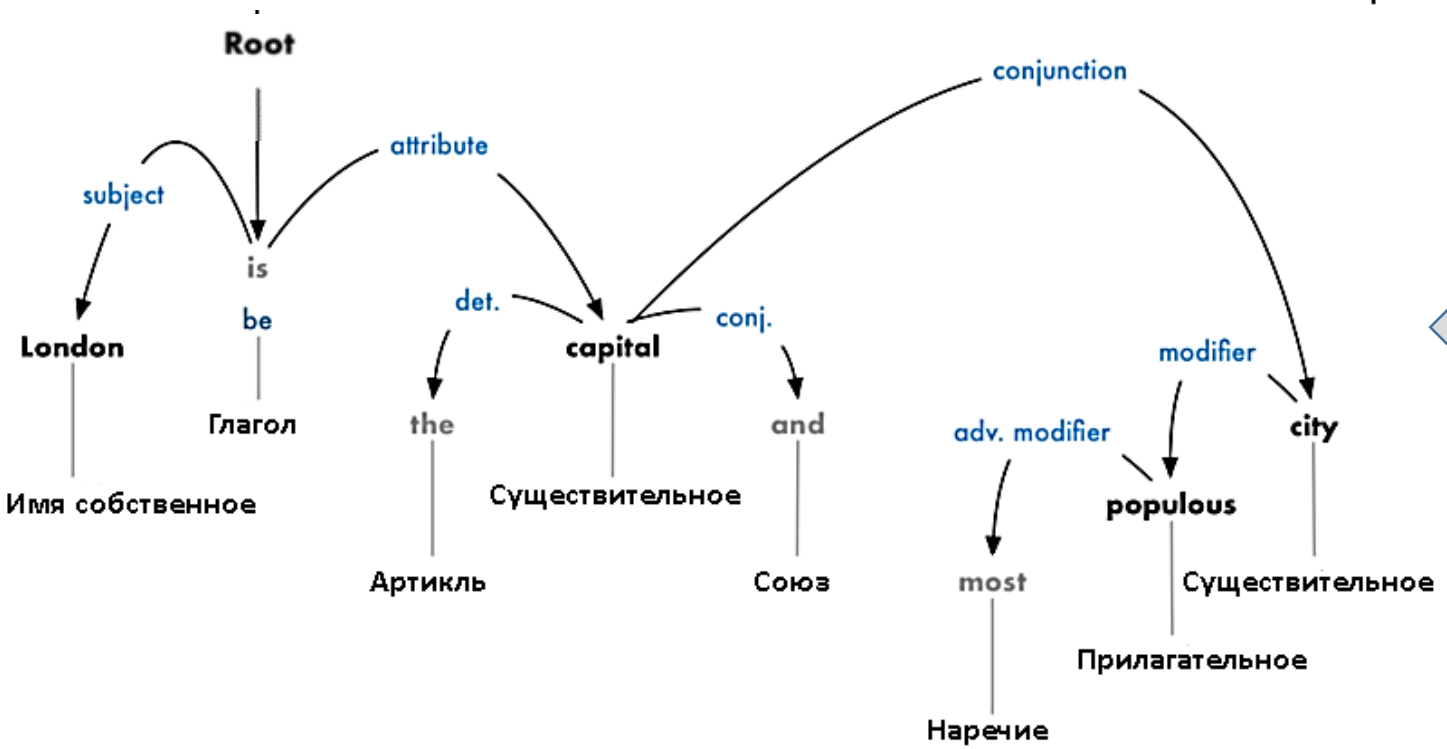
Шаг 6. Парсинг зависимостей

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:

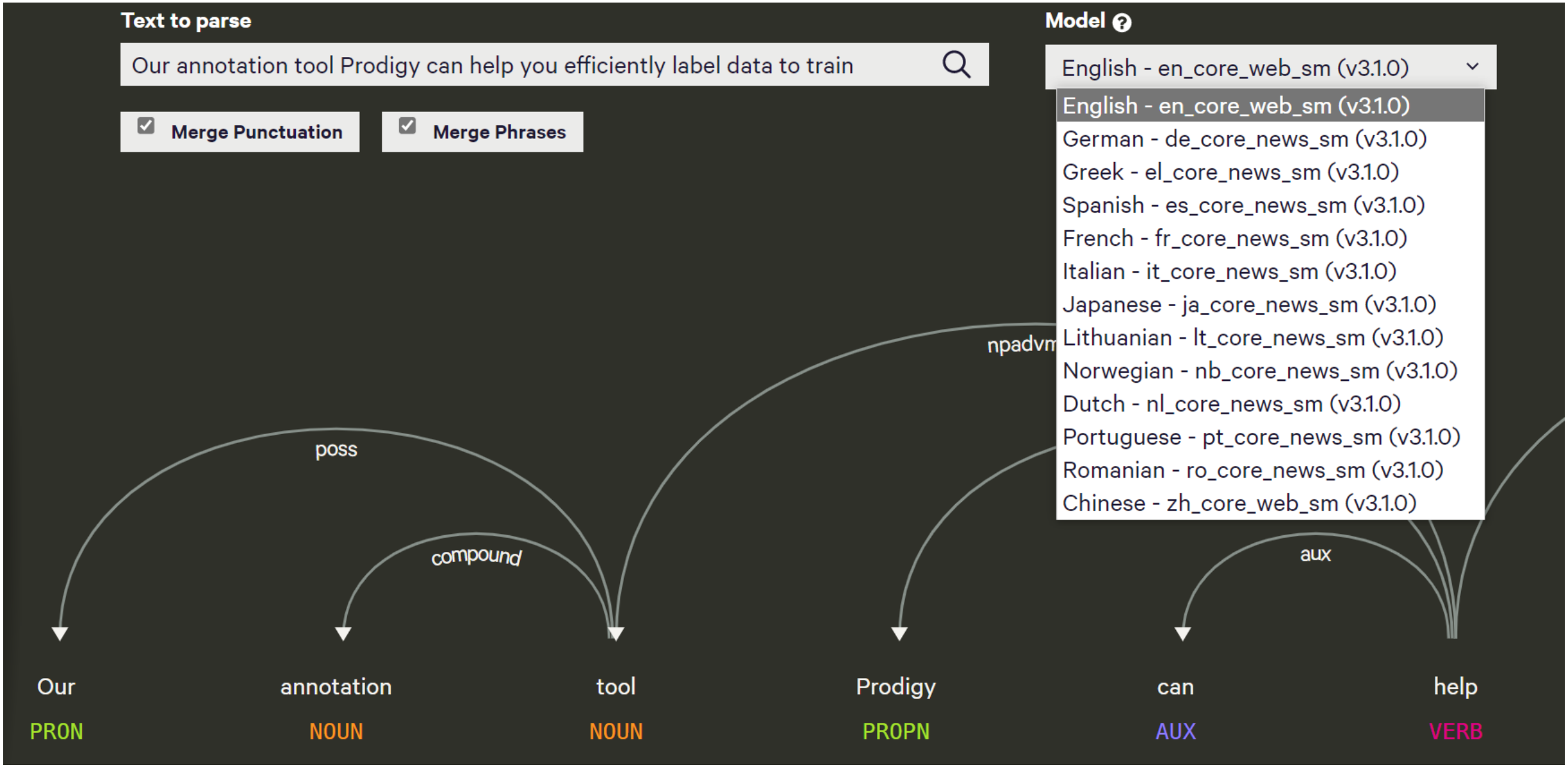




Но нужно не только определить родителя, но и установить тип связи между двумя словами.

*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!*

[displaCy Dependency Visualizer · Explosion]



Шаг 6б. 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.

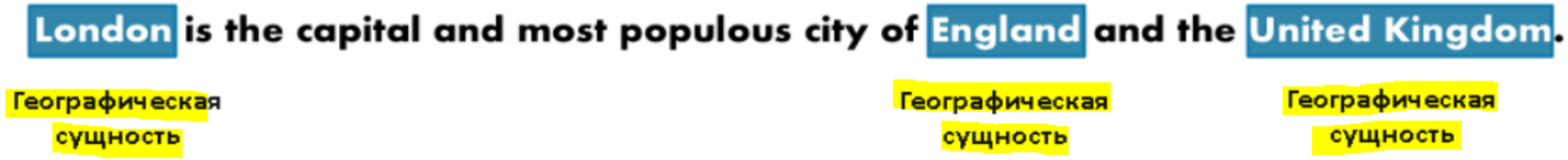
Шаг 7. Распознавание именованных сущностей (Named Entity Recognition, NER)

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:



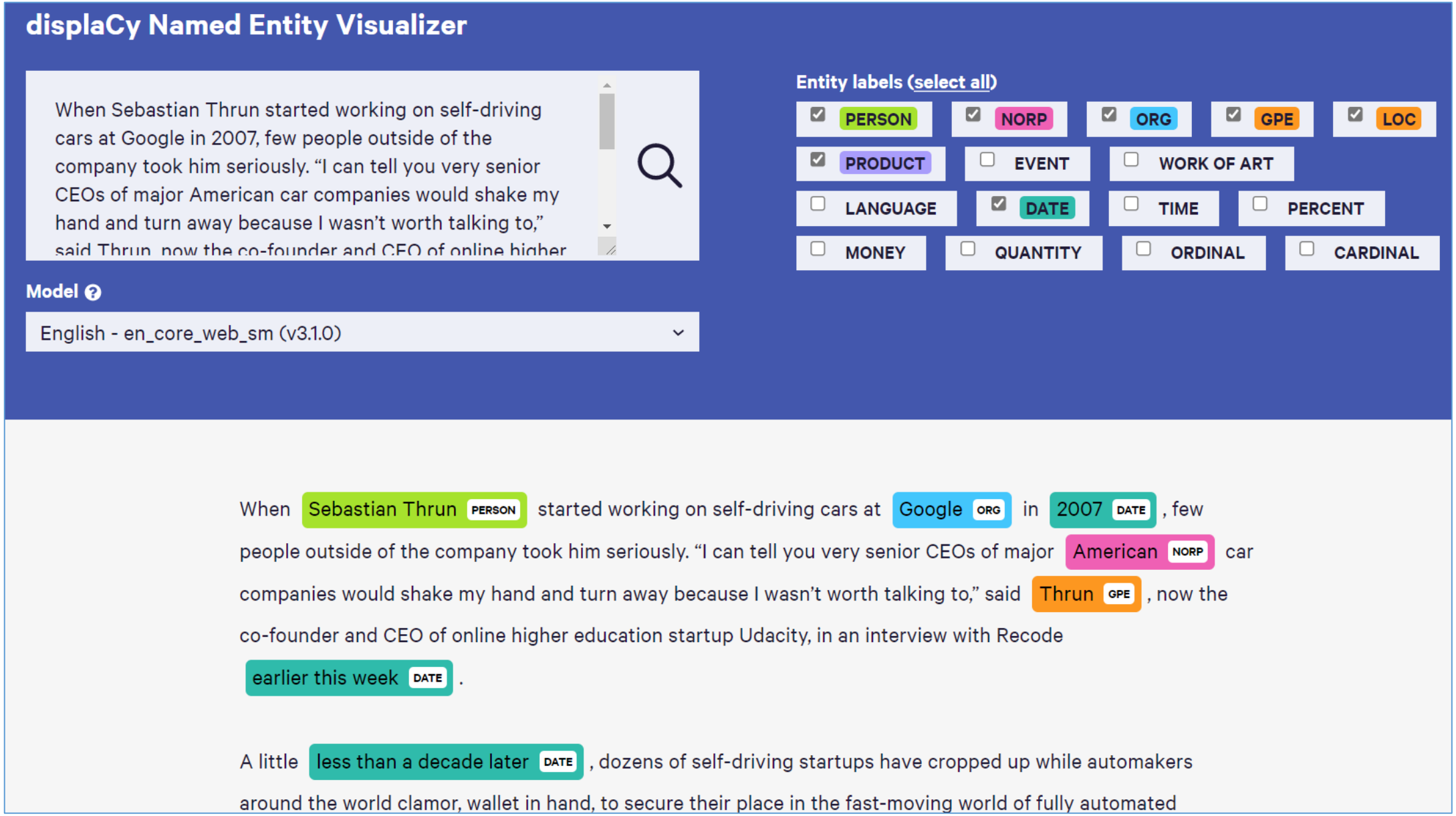
NER-системы не просто просматривают словари. Они анализируют контекст токена в предложении и используют статистические модели, чтобы угадать какой объект он представляет. *Хорошие NER-системы способны отличить актрису Brooklyn Decker от города Brooklyn.*

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.

[displaCy Named Entity Visualizer · Explosion]



Шаг 8. Разрешение кореференции

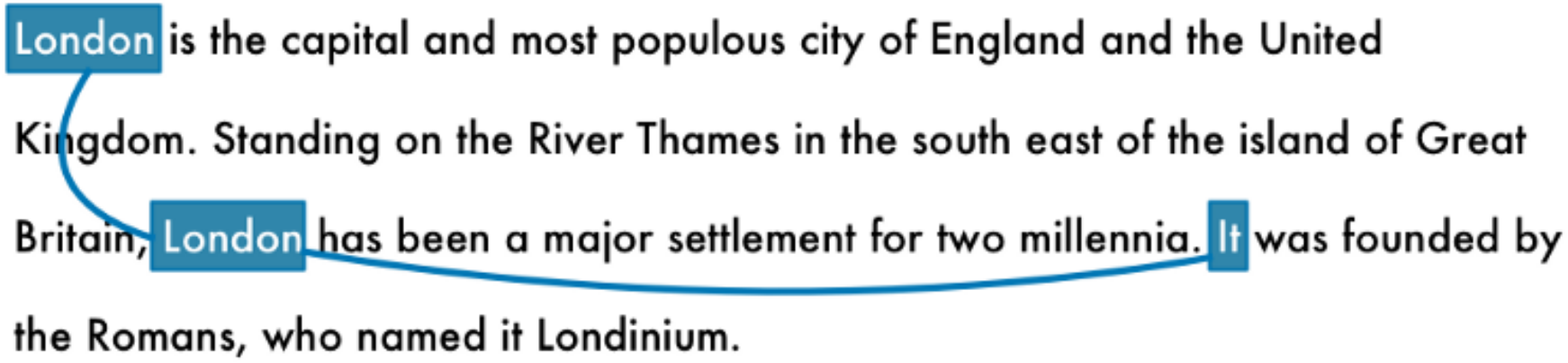
В английском очень много местоимений – слов вроде he, she, it. Это сокращения, которыми мы заменяем на письме настоящие имена и названия. Человек может проследить взаимосвязь этих слов от предложения к предложению, основываясь на контексте. Но NLP-модель не знает о том, что означают местоимения, ведь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

Давайте посмотрим на третью фразу в нашем документе:

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

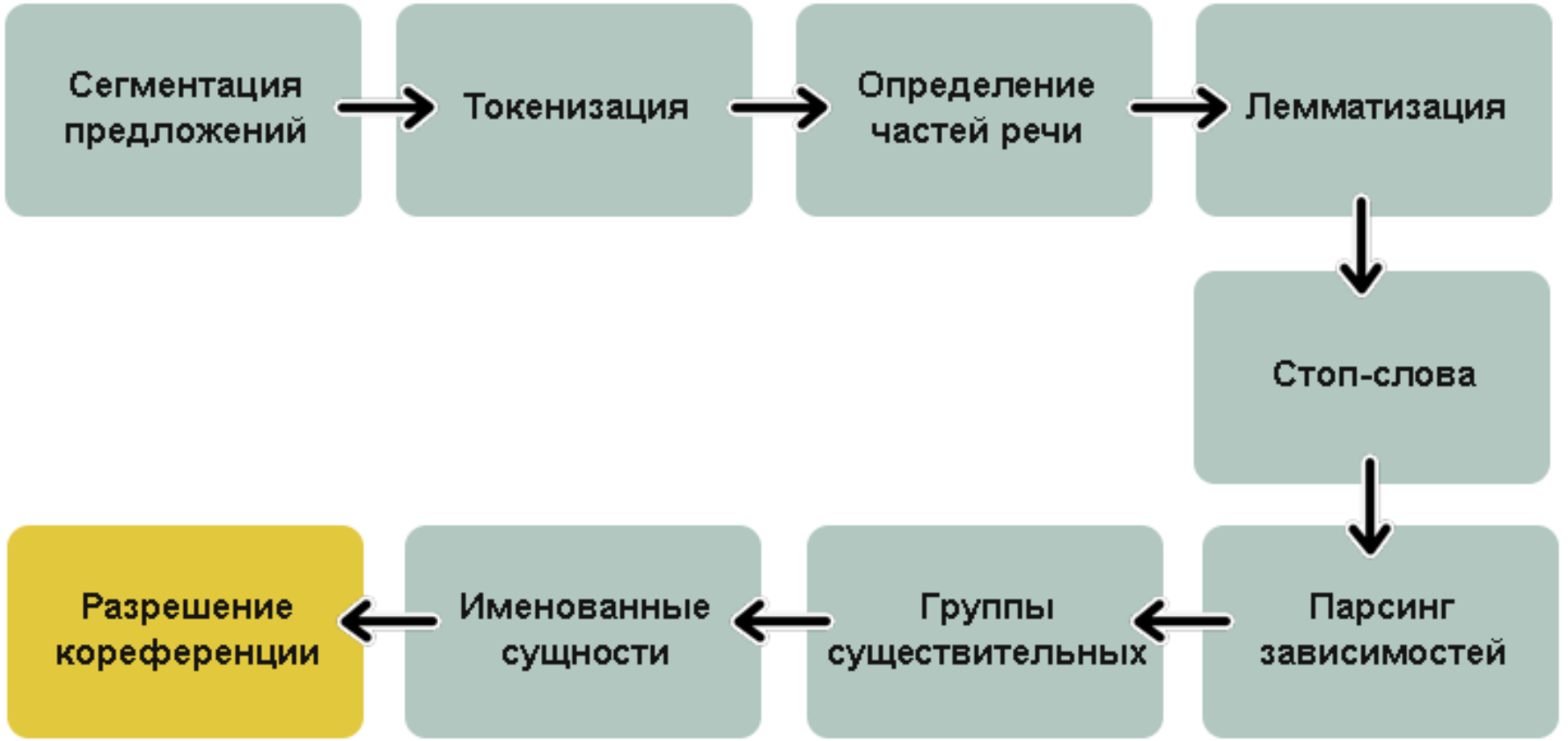
В процессе чтения мы догадаемся, что «это» не что иное как Лондон. Разрешением кореференции называется отслеживание местоимений в предложениях с целью выбрать все слова, относящиеся к одной сущности.

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.

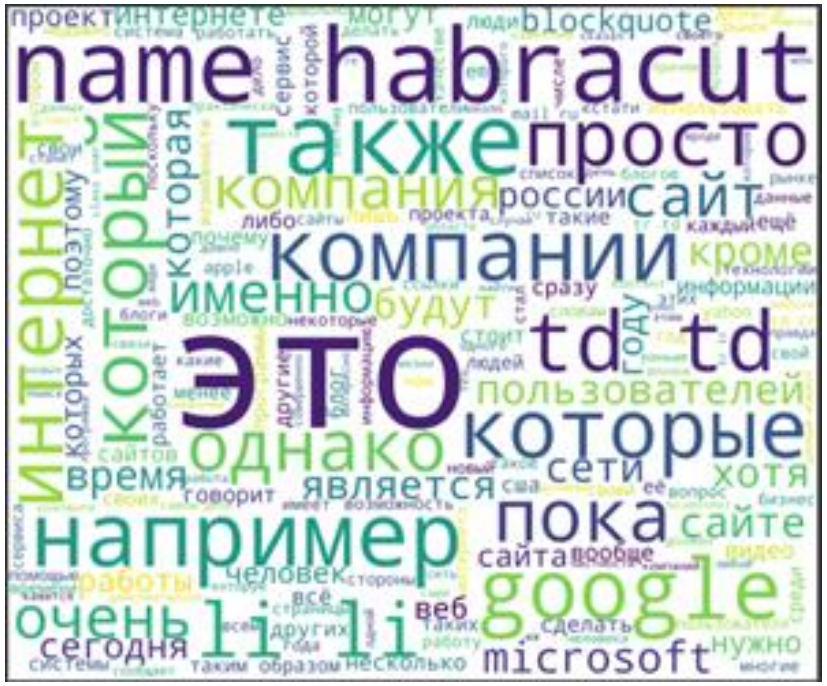
Например, **spaCy** производит сегментацию позже, используя для нее результаты парсинга зависимостей.

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:

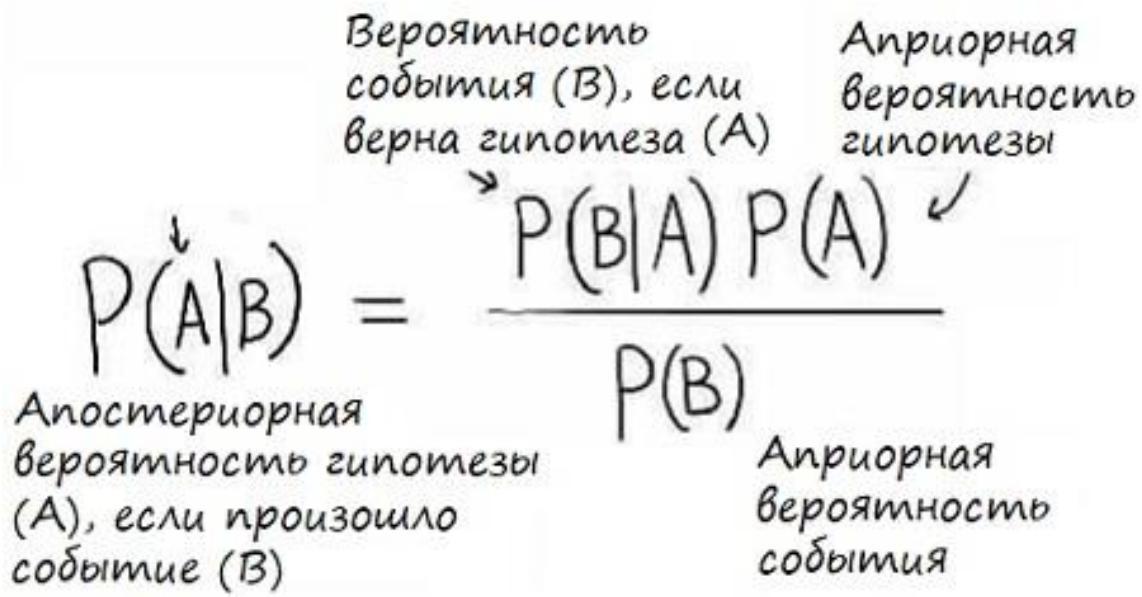


После очистки текстов от «стоп-слов» и знаков пунктуации количество уникальных слов снизилось до 142253, а «облако слов» стало более осмысленным:

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.

Инструменты

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.