Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования «Московский политехнический университет»

(МОСКОВСКИЙ ПОЛИТЕХ)



Компьютерная лингвистика

Тема: Базовые операции NLP

Выполнил студент: Гетун Дмитрий Юрьевич

Группа: 224-322

Преподаватель: Филиппович Юрий Николаевич

Москва 2023

**Оглавление**

[**Введение** 3](#_Toc124598750)

[**Базовые операции NLP** 4](#_Toc124598751)

[**Лемматизация** 5](#_Toc124598752)

[**Использование лемматизации для распознавания смысла** 7](#_Toc124598753)

[**Разметка частей речи** 8](#_Toc124598754)

[**Поиск соответствующих глаголов с помощью тегов частей речи** 9](#_Toc124598755)

[**Синтаксические отношения** 11](#_Toc124598756)

[**Главный и дочерний элемент** 12](#_Toc124598757)

[**Распознавание именованных сущностей** 14](#_Toc124598758)

[**Заключение** 15](#_Toc124598759)

# **Введение**

Все чаще, позвонив в банк или поставщику услуг Интернета, мы можем услышать нечто вроде: «Здравствуйте, я ваш виртуальный помощник. Слушаю вас!» С каждым днем роботы становятся умнее и уже могут общаться с людьми на естественном языке. Но как работают такие технологии и как их использовать в своих проектах?

Обработка естественного языка (NLP) - одно из направлений искусственного интеллекта (ИИ), ставящее перед собой цель научить машины понимать человеческие языки и реагировать на высказывания. Без этой базовой технологии невозможен ни один виртуальный помощник. Данная лекция поможет вам обрести навыки, необходимые для создания собственного чат-бота и NLP-приложений, способных решать реальные задачи, такие как анализ предложений, улавливание смысла текста, составление текстов и т. д.

# **Базовые операции NLP**

Цепочка базовых операций NLP представляет собой конвейер обработки. Конвейер обработки обычно включает операции токенизации, лемматизации, разметки частей речи, разбора синтаксических зависимостей и распознавания именованных сущностей.

Каждый текст может быть разбит на следующие составляющие:

1. Главы и секции
2. Параграфы
3. Пункты
4. Предложения
5. Фразы
6. Слова
7. Морфемы (корень, суффикс, постфикс, приставка, окончание)

Здесь нас интересуют слова, которые и станут отправной точкой для дальнейшего анализа текста.

Токенизация — процесс разделения текста на составляющие (их называют «токенами»). Чаще всего текст разделяется на токены по словам или предложениям. В русском и английском языках в качестве индикаторов, по которым осуществляется разделение, используются:

* для слов — пробел;
* предложений — точка, восклицательный и вопросительный знаки.

Использование тех или иных индикаторов для разделения текста зависит от особенностей языка, на котором написан текст.

Токенизация по предложениям преимущественно используется в таком направлении, как машинный перевод, когда важен контекст целого предложения, а не отдельных слов. Другое направление — синтез человеческой речи, в рамках которого происходит воспроизведение (озвучивание) заданного текста.

Токенизация по словам имеет массовое применение. На её основе работает большинство автокорректоров орфографии, а также может осуществляться разметка текста, частным случаем которой является классификация (категоризация) текста.

Часто при токенизации по словам производят дополнительную разметку ещё и по части речи для каждого токена. После осуществления подобной операции у каждого слова появляется дополнительный признак в виде части речи, к которой оно относится.

В некоторых случаях используется более редкий вид токенизации — по группе слов. Он является частным случаем токенизации по словам. При подобном подходе важно не одно слово, а сочетание определенных пар (например, глагол + существительное).

В качестве примера будем рассматривать текст “Я подготовил лекцию по компьютерной лингвистике и надеюсь хорошо сдать экзамен, поэтому постарался хорошо подготовиться!”.

Следующим кодом библиотеки spacy мы можем разбить исходный текст на токены

1. text = 'Я подготовил лекцию по компьютерной лингвистике и надеюсь хорошо сдать экзамен, поэтому постарался хорошо подготовиться!'
2. doc = nlp(text) *# создаем экземпляр класса*
3. **print**([token.text **for** token **in** doc])

В результате получаем следующий набор слов

['Я', 'подготовил', 'лекцию', 'по', 'компьютерной', 'лингвистике', 'и', 'надеюсь', 'хорошо', 'сдать', 'экзамен', ',', 'поэтому', 'постарался', 'хорошо', 'подготовиться', '!']

## **Лемматизация**

Следующим пунктом в конвейере обработки текста идет лемматизация. Лемма – начальная, словарная форма слова. В русском языке для существительных и прилагательных это форма именительного падежа единственного числа, для глаголов и глагольных форм – форма инфинитива.

Лемматизация — это процесс приведения всех изменяемых форм слова к единому значению. Снижает вариативность одного и того же слова, что повышает качество анализа текста.

Алгоритм лемматизации основан на поиске наиболее подходящего варианта слова по словарю. При анализе текстовой информации обычно используются данные, полученные после процесса токенизации, подразумевающего разделение текста на отдельные слова или предложения. После сопоставления со словарем все словоформы одного слова заменяются на одно конкретное значение.

В языках со сложным словообразованием (например, русском) может потребоваться помимо стандартных словарей использовать дополнительные, учитывающие специфику речи. Отдельно к процессу лемматизации подключаются словари сленга, аббревиатуры и сокращений.

Пример для английского языка: слово «walk» будет являться единой формой для глаголов «walking», «walked», «walks» и непосредственно «walk», а слово «eat» для «ate», «eaten», «eating», «eat». Таким образом, все словоформы после применения алгоритма лемматизации примут одно единое значение.

Пример для русского языка: слово «отправлять» будет являться единой формой для глаголов «отправляю», «отправляешь», «отправляет», «отправляем», «отправляете», «отправляют», и, непосредственно, «отправлять».

Следующий пример кода иллюстрирует процесс лемматизации для нашего текста.

1. text = 'Я подготовил лекцию по компьютерной лингвистике и надеюсь хорошо сдать экзамен, поэтому постарался хорошо подготовиться!'
2. doc = nlp(text) *# создаем экземпляр класса*
3. **for** token **in** doc:
4. table.add\_row([token.text, token.lemma\_])

В результате получаем следующую таблицу, где столбец слева содержит токены, столбец справа – соответствующие леммы.

|  |  |
| --- | --- |
| **Слово** | **Лемма** |
| Я | я |
| подготовил | подготовить |
| лекцию | лекция |
| по | по |
| компьютерной | компьютерный |
| лингвистике | лингвистика |
| и | и |
| надеюсь | надеяться |
| хорошо | хорошо |
| сдать | сдать |
| экзамен | экзамен |
| , | , |
| поэтому | поэтому |
| постарался | постараться |
| хорошо | хорошо |
| подготовиться | подготовиться |
| ! | ! |

## **Использование лемматизации для распознавания смысла**

При решении задачи распознавания смысла особо важное значение имеет этап лемматизации. Чтобы понять почему, рассмотрим предложение “Я лечу в Питер”.

Пусть это предложение было подано на вход NLP-приложения, взаимодействующего с онлайн-системой, которая предоставляет API для бронирования билетов на различный транспорт. Приложение обрабатывает запрос пользователя, выделяет из него необходимую информацию и передает ее нижележащему API. NLP-приложение пытается извлечь из запроса пользователя следующую информацию: вид путешествия (поезд, автобус, самолет) и пункт назначения.

Сначала приложению необходимо выяснить, что хочет приобрести пользователь: билет на автобус, поезд или самолет. Для этого оно ищет слово, совпадающее с одним из ключевых слов заранее заданного списка. Простейший способ облегчить поиск – преобразовать все слова в предложении в соответствующие леммы, и тогда список ключевых слов может быть более коротким и понятным. Например, нет необходимости включать в него все слоформы глагола “летать” (такие как “лететь”, “лечу”,”летаю”), чтобы приложение могло понять желание пользователя заказать именно авиабилет. Достаточно свести все возможные варианты к базовой форме слова – летать.

Лемматизация также удобна при определении пункта назначения на основе запроса пользователя. У многих городов мира есть прозвища, но для системы бронирования билетов необходимы официальные названия. Конечно, объекту Tokenizer, который выполняет лемматизацию по умолчанию, неизвестна разница между разговорными и официальными названиями городов, стран и т.д. Для решения этой проблемы добавим в имеющийся экземпляр объекта правила для подобных исключений

Следующий сценарий иллюстрирует возможную реализацию лемматизации для нескольких городов назначения. В нем выводятся леммы составляющих предложение слов.

Описываем исключение для слова Питер, заменив его лемму по умолчанию на Санкт-Петербург. Далее добавляем это исключение в экземпляр объекта. Теперь объект всякий раз будет использовать данное исключение при запросе леммы для слова Питер. Для проверки выводим леммы всех слов в предложении

|  |  |  |
| --- | --- | --- |
| Слово | Лемма вариант 1 | Лемма вариант 2 |
| Я | Я | Я |
| Лечу | Лечу | Лечу |
| в | В | В |
| Питер | Питер | Санкт-Петербург |

## **Разметка частей речи**

Тег части речи – указывает, к какой части речи в этом конкретном предложении относится конкретное слово (оно может быть существительным, глаголом, наречием и т.д.). Слова в зависимости от контекста могут выступать в роли разных частей речи.

В библиотеке spaCy теги частей речи нередко содержат и подробную информацию о токене. Например, в информации о глаголе могут быть указаны следующие признаки: время – прошедшее, настоящее или будущее. Вид – совершенный или несоверешенный, лицо 1-е, 2-е или 3-е, число – единственное или множественное.

Извлечение тегов частей речи для глаголов может помочь с определением намерения пользователя, когда токенизации и лемматизации недостаточно. Скажем, сценарий лемматизации для приложения бронирования билетов из предыдущего раздела не поможет NLP-приложению выбрать слова в предложении для составления запроса к нижележащему API: на практике подобная задача бывает достаточно сложной. Например, если запрос пользователя состоит более чем из одного предложения.

Я прилетел в Москву. Сейчас я хочу полететь в Питер.

Результат лемматизации этого предложения выглядит следующим образом:

['я', 'прилететь', 'в', 'москва', '.', 'сейчас', 'я', 'хотеть', 'полететь', 'в', 'Санкт-Петербург']

Одной лемматизации здесь недостаточно, ведь приложение может счесть ключевыми словами леммы прилететь и Москва из первого предложения, указывающие на желание пользователя полететь в Москву, хотя тот на самом деле намерен отправиться в Питер. Проблему усложняет тот факт, что при лемматизации глаголы сводятся к формам инфинитива, из-за чего сложно определить их роль в предложении.

В такой ситуации на помощь приходят теги частей речи. В русском языке в число основных частей речи входят существительное, прилагательное, числительное, местоимение, глагол, наречие, причастие. Также есть предлог, союз.

В таблице ниже приведено сопоставление обозначений частей речи в библиотеке spaCy с их значениями.

|  |  |
| --- | --- |
| TAG | Описание |
| NN | Существительное,  Единственное число |
| NNS | Существительное,  Множественное число |
| PRP | Местоимение, личное |
| PRP$ | Местоимение, притяжательное |
| VB | Глагол, инфинитив |
| VBD | Глагол, прошедшее время |
| VBG | Глагол или причастие настоящего времени |
| JJ | Прилагательное |

Вероятно, самые интересные для NLP-приложений свойства глаголов – время и вид. Взятые вместе, они указывают на то, как определенное глаголом действие протекает во времени. Например, глагол совершенного вида прошедшего времени описывает, что человек на момент речи уже завершил действие.

## **Поиск соответствующих глаголов с помощью тегов частей речи**

Как было сказано в начале лекции, токенизация может осуществляться не только по отдельным словам, а по связкам слов.

Приложение по бронированию билетов может использовать теги частей речи, чтобы отфильтровать глаголы в тексте и оставить лишь те, которые нужны для определения намерений пользователя.

Для этого можно воспользоваться библиотекой Natasha для языка python, которая является более специализированной для русского языка т.к. spaCy нацелен в основном на английский язык, и хотя поддерживает и русский – плохо справляется с определением частей речи.

Прежде чем перейти к реализации кода этого процесса, давайте подумаем, какими фразами пользователи могут выражать намерение забронировать билет, скажем, в Санкт-Петербург. Начнем с несколько фраз, содержащих следующее словосочетание лемм: лететь, в, Питер. Вот несколько простейших вариантов:

1. Мне нужно лететь в Питер
2. Я хочу лететь в Питер
3. Я летел в Питер
4. Я лечу в Питер

Обратите внимание: несмотря на то, что после сведения до лемм все эти предложения будут включать сочетание лететь в Питер, лишь часть из них выражает реальное намерение пользователя забронировать билет в Питер. Последние два явно не подходят.

Проанализировав ситуацию, мы поймем что форма глагола лететь прошедшего и настоящего времени, использованные в последних двух предложениях, не подразумевают искомое намерение. Нам подходят только формы инфинитива и настоящего времени. Следующий сценарий иллюстрирует, как можно найти эти формы в нашем примере текста

Изображение выглядит как текст

Автоматически созданное описание

На данном скриншоте показан морфологический разбор слов в нашем тексте. Здесь видно, что слово прилетел является глаголом прошедшего времени и совершенного вида. Глагол полететь также является глаголом прошедшего времени и совершенного вида, но перед ним находится глагол хочу, который сам по себе является глаголом несовершенного вида настоящего времени.

Вид глагола может зависеть от контекста предложения. Например:

1. Она сейчас (что делает?) ночует у подруги. — Настоящее время, несовершенный вид.
2. Она завтра (что сделает?) ночует у подруги. — Будущее время, совершенный вид.

Местоимение “я” во втором предложении имеет часть речи “nsubj”, что переводится как подлежащее. Глагол полететь имеет часть речи “xcomp”, что на русском будет звучать как “составное именное сказуемое”. Это значит, что глагол полететь рассматривается вместе с глаголом хочу. Мы видим, что глагол хочу вообще не имеет какую-либо часть речи.

Изображение выглядит как текст, табло

Автоматически созданное описание

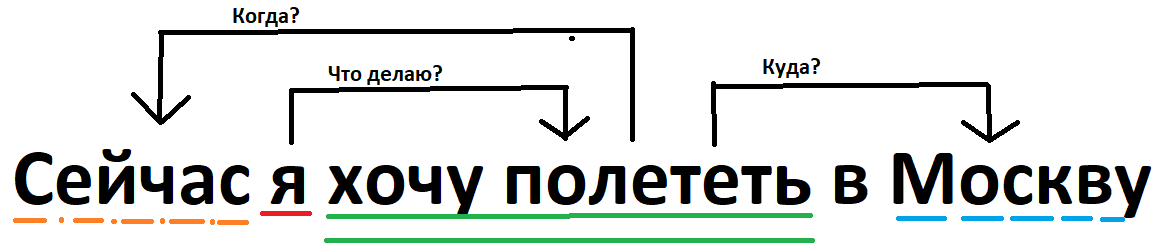
Из предыдущего раздела можно сделать вывод о том, что тегов частей речи не всегда достаточно для определения смысла высказывания – иногда необходимо учитывать и контекст.

## **Синтаксические отношения**

Следующим шагом для анализа нашего предложения является соединение имен собственных с составным именным сказуемым. Как видно из предыдущего примера, недостаточно просто найти существительное, потому что под такой параметр может попасть как местоимение “я”, так и название города “Москва”, которая в данном контексте подлежащим являться не может. Человеку очень легко понять, что человек хочет лететь в Москву, но нам надо научить компьютер делать это автоматически, без нашего вмешательства.

Для обработки подобных ситуаций мы используем средство разбора синтаксических зависимостей, которое выявляет синтаксические отношения между отдельными токенами в предложении и соединяет дугами синтаксически связанные пары слов.

Метки синтаксической зависимости, равно как и леммы и теги частей речи – это лингвистические признаки. Например, метка зависимости case будет означать предлог. Соответствующее отношение можно проиллюстрировать направленной дугой



## **Главный и дочерний элемент**

Метка синтаксический зависимости описывает вид синтаксического отношения между двумя словами в предложении. В подобной паре одно слово является синтаксически управляющим элементом (его иногда называют главным или родительским элементом), а второе зависит от него (его называют дочерним элементом).

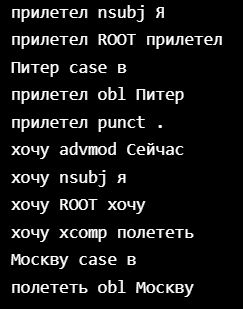
У каждого слова в предложении есть только один главный элемент. Соответственно, любое слово может быть дочерним по отношению лишь к одному главному элементу. Обратное утверждение справедливо не всегда: одно и то же слово может вообще не выступать в роли главного элемента или играть эту роль как в одной, так и в нескольких парах. Последнее означает, что у главного элемента есть несколько дочерних. Таким образом, метка зависимости всегда присваивается дочернему элементу пары.

Изображение выглядит как текст

Автоматически созданное описание

Скриншот не демонстрирует связь слов в предложении с помощью упомянутых в начале данного раздела дуг зависимостей. Чтобы посмотреть дуги зависимостей для нашего примера текста, заменим цикл из предыдущего сценария таким образом.

Свойство head объекта Token ссылается на синтаксический главный элемент элемент данного токена. Вывод этой информации позволяет увидеть, какими синтаксическими зависимостями связаны между собой слова в предложениях текста. Если изобразить зависимости графически, каждой строке следующего вывода будет соответствовать дуга. Исключение составляет отношение ROOT (поскольку слово с этой меткой – единственное в предложении, у которого нет главного элемента или же главным элементом этого слова является оно само).



Учитывая приведенный выше список синтаксических зависимостей, попробуем разобраться, какие метки указывают на токены, лучше всего описывающие намерение пользователя. Другими словами, найдем пару, которая способна более или менее точно передать основной посыл текста.

В этом смысле многообещающе выглядят токены, маркированные метками зависимостей xcomp и obl (в данном примере они играют ключевую роль в распознавании намерения). Метка xcomp обозначает целевое действие, в obl в этом примере отмечает сущность, которая в сочетании с глаголом резюмирует смысл всего высказывания.

text = 'Я прилетел в Питер. Сейчас я хочу улететь в Москву'

doc = nlp(text)

for token in doc.sents:

    print([w.text for w in token if w.dep\_ == 'xcomp' or w.dep\_ == 'obl'])

Здесь мы разрезаем связный текст, разбивая его на предложения с помощью свойства doc.sents, предназначенного для прохода в цикле по отдельным фразам документа. Разрезание текста на отдельные предложения удобно, например, для поиска определенных частей речи в каждом из предложений текста. Благодаря этому на основе конкретных меток зависимости, присвоенных токенам, для каждого предложения можно создать список потенциальных слов. Условия фильтрации в примере выбирались исходя из анализа синтаксически связанных пар, сгенерированных предыдущим сценарием. В частности, выбирались токены с метками зависимости xcomp и obl, посколько именно они формируют интересующие нас пары.

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

В обеих парах предложений выходные существительные были маркированы как obl, что в приложении бронирования билетов поможет в выборе существительного, лучше всего подходящего к глаголу. В данном случае это составное сказуемое хочу полететь и существительное Москва.

Стоит помнить, что привиденный пример – упрощенный вариант выделения информации с помощью меток зависимостей. Существует множество вариантов того, как пользователь может сформировать запрос.

## **Распознавание именованных сущностей**

Именованная сущность – объект, на который можно ссылаться по его собственному наименованию. Именованной сущностью может быть человек, организация, место или другая сущность. Именованные сущности играют важную роль в NLP, поскольку позволяют выяснить, о каком месте или организации говорит пользователь. Следующий сценарийй ищет именованные сущности в тексте, который мы использовали в предыдущих сценариях.

…

Если атрибуту ent\_type токена не присвоено значение 0, значит, этот токен – именованная сущность. В таком случае выводим атрибут ent\_type\_ токена, содержащий тип именованной сущности в поле. В результате выполнения нашего сценария должно выводиться следующее:

Изображение выглядит как текст

Автоматически созданное описание

И Питер, и Москва маркированы меткой LOC, которая расшифровывается как location (местоположение).

# **Заключение**

В данной лекции мы рассмотрели возможность использования различных библиотек для решения конкретной задачи. Конечно, это далеко не все возможности данных библиотек и уж тем более рассмотренный вариант взаимодействия с пользователем не единственный.

Среди рассмотренных этапов обработки текста мы рассмотрели токенизацию, лемматизацию, разметку, синтаксический разбор и распознавание сущностей.