ЦИФРОВАЯ КУЛЬТУРА

УНИВЕРСИТЕТ ИТМО

Введение в автоматическую обработку текстов

Высшая школа цифровой культуры

Университет ИТМО

dc@itmo.ru

Оглавление

Введение **4**

Для кого этот курс **5**

Что такое NLP, обзор задач **6**

NLP vs CompLing **7**

Примеры трудностей в обработке языка **7**

Краткая и неполная история компьютерной лингвистики **12**

Докомпьютерная эра **12**

Отступление: эксперимент с “Онегиным” и цепи Маркова 12

Компьютерная эпоха 14

50-е 16

60-е и 70-е 16

80-е и 90-е 17

2000-е 17

2010-е 18

Готовим тексты для работы с машиной **18**

Стемминг 19

Стеммер Портера 20

Лемматизация 22

Морфологический вывод 22

Морфологическая неоднозначность 23

Инструментарий для русского языка 24

Pymorphy2 24

Mystem 26

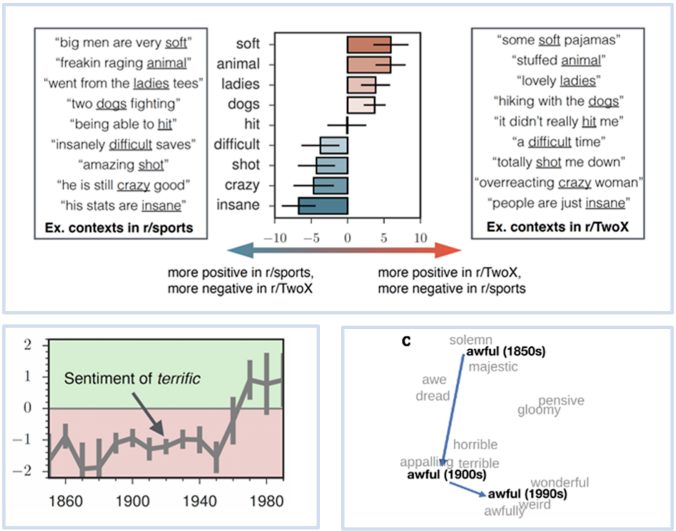
RNNMorph 28

Введение

Добро пожаловать на вводный курс по обработке естественного языка. NLP, natural language processing -- прикладная область знаний, целая наука, ставшая особенно популярной в последние десять лет.

Подсказки следующего слова в письмах и сообщениях в телефонах, определение языка и автоматический перевод на другой язык, готовые ответы в почте, автоматическая проверка орфографии и синтаксиса в текстовых редакторах -- и так далее. Эти помощники стали привычными в повседневной жизни многих. Принципами работы этих инструментов занимается обработка естественного языка.

Какой смысл слова crazy, как правило, имеют в виду фанаты хоккея и какой -- участники дискуссий о равноправии полов? Как менялся смысл качественных прилагательных с девятнадцатого века по настоящее время? Какие тропы и фигуры речи характерны для того или иного писателя? Вопросы прикладной лингвистики также часто относят к NLP.



Смысл слова Crazy:
https://nlp.stanford.edu/projects/socialsent/

О ком чаще пишут арт-критики Москвы и Петербурга, и в чьих оценках они сходятся? Что можно узнать о негативных побочных эффектах того или иного лекарства по текстам пациентов в социальных сетях? Тексты в электронном виде -- один из основных способов представления информации в интернете, поэтому именно методы обработки естественного языка часто используются как основной инструмент в прикладных междисциплинарных исследованиях на основе данных.

На момент 2020 года наука об автоматической обработке языка хорошо развита. Некоторые задачи решаются просто, а для решения других -- приходится обращаться к не самой простой математике. К счастью, общедоступные инструменты становятся всё удобнее. И, может быть, именно вы сделаете прорыв в вашей научной области, взяв NLP как метод анализа и автоматизации работы с большими (и не очень) массивами данных. Мы будем очень рады, если наш курс вам в этом поможет.

Для кого этот курс

Это вводный и обзорный курс, который покажет лишь небольшую, но, возможно, самую важную часть того, что можно сделать современными методами обработки естественного языка.

Мы ориентировались на тех, кому, как мы надеемся, ещё только предстоит работать с текстом в научных кросс-дисциплинарных или просто количественных исследованиях или заниматься анализом данных, например, в коммерческих организациях.

Курс не предполагает серьезной математической подготовки слушателя, но не предполагает и страха перед, возможно, новыми для слушателя моделями, подходами и идеями. Основы программирования в багаже также помогут, но можно успешно понять и сдать курс и без них. Главное -- смелость, открытость новому и интерес к машинным методам работы с языками.

Каждую неделю вам предстоит прослушать лекции, ответить на контрольные вопросы и решить задачу. Как правило, это запуск готовых или почти готовых программ.

Что такое NLP, обзор задач

NLP определяют по-разному, единого мнения нет.

[Jurafsky et al, ed. 2] The goal of this new field is to get computers to perform useful tasks involving human language, tasks like enabling human-machine communication, improving human-human communication, or simply doing useful processing of text or speech.

Перевод: Цель этой новой предметной области -- с помощью компьютеров решать полезные задачи, при решении которых приходится иметь дело с естественным языком. Сюда относятся человеко-машинная коммуникация, поддержка коммуникации между людьми или попросту осмысленная обработка текста и речи.

[Wikipedia 2019-08-25]: Natural language processing (NLP) is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data.

Перевод: Обработка естественного языка -- это поднаправление лингвистики, информатики, управления информацией и искусственного интеллекта, в рамках которого рассматривается взаимодействие между вычислительными машинами и человеческими естественными языками, в частности, методами компьютерной обработки и анализа больших объёмов данных на естественных языках.

Во всех определениях важно, что это прикладная область на стыке лингвистики, информатики и искусственного интеллекта. Можно также отметить, что иногда в определение включают обработку речи, то есть звуков. На наш взгляд, несмотря на

множество пересечений с NLP, это всё же отдельная наука, и касаться в этом курсе мы её не будем.

NLP vs CompLing

Важно отличать обработку естественного языка от компьютерной или, точнее, вычислительной лингвистики. Очевидно, они сильно пересекаются. Но если задача лингвистики -- изучение языка, его устройство, в том числе абстрактные теории и моделирование, то NLP фокусируется, как правило, на приложениях, обеспечивающих машинную обработку языка. Например, для обеспечения человеко-машинного взаимодействия. Это часто указывают как конечную цель обработки естественного языка. Озвученные только что определения спорны, но в рамках этого курса будем считать так.

В эпохи оптимизма казалось даже достойнейшим из лингвистов казалось, что язык можно описать наборами простых законов и правил. Действительно, нас же учат в школе, что есть правила, есть исключения. Давно готовы словари многих языков в электронном виде. Нужно только один раз очень постараться, задать все правила в одной большой системе, и машины запросто смогут решать все стандартные задачи обработки языка, что и мы. И некоторые узкоспециализированные системы на правилах действительно иногда отлично работают.

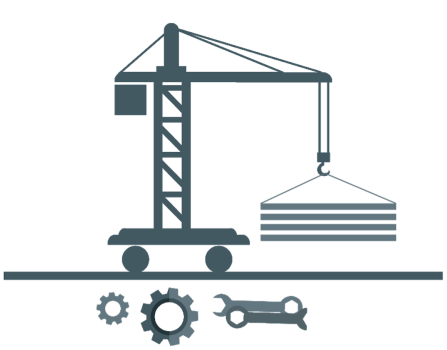
Но язык куда сложнее, чем может показаться на первый взгляд. И полное понимание смысла часто неформально относят к так называемым AI-hard problems, то есть задачам, решение которых настолько же сложно, насколько сложно и создание общего искусственного интеллекта.

Примеры трудностей в обработке языка

Давайте рассмотрим несколько известных примеров.

Что не так с предложением “**Эти типы стали есть в цехе**”? Может быть, здесь речь идёт о типах металла, которые оказались в наличии в цехе, а может быть -- о халатных посетителях цеха, поленившихся дойти до столовой. Не знаем, что за цех, но надеемся, они не повредили ни своему здоровью, ни производству. Обе трактовки допустимы, и нужную нам может подсказать только контекст.

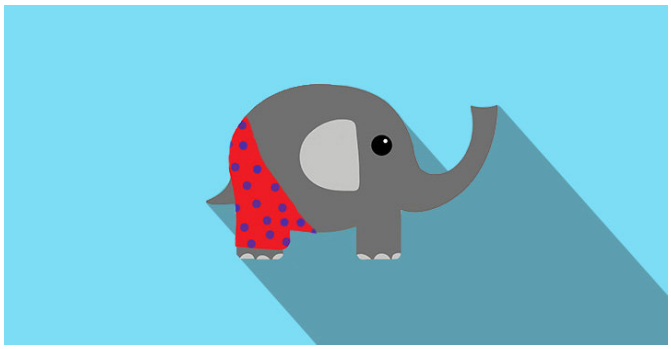
Эти типы стали есть в цехе



Не так-то просто придётся машине и с предложением “**За песчаной косой лопоухий косой пал под острой косой…**” “Косой” здесь -- это и коса как сельскохозяйственное орудие труда, и коса как узкий мыс в водоёме, и косой -- субстантивированное прилагательное, которым в данном случае, скорее всего, обозначают зайца. Если нам важно, чтобы машина поняла, какой из смыслов скрывается за каждым из словоупотреблений, придётся постараться.

Рассмотрим предложение “**Мама вымыла раму, и теперь она устала**”. Не всегда очевидно сходу, в чём здесь проблема, всё же так понятно. Но здравого смысла у машины вообще говоря нет. Поэтому понять, кто именно устал -- мама или рама, запросто не получится. Мы знаем что “рама” -- это предмет и устать буквально она не может, поэтому речь, конечно, о маме. Эта важная для лингвистики задача называется разрешением анафоры.

Похожий пример, часто используемый для демонстрации случаев неоднозначности синтаксического разбора -- знаменитая шутка из фильма 1930 года Animal Crackers: **One morning I shot an elephant in my pajamas. How he got into my pajamas I'll never know. (Однажды я в моей пижаме застрелил слона. Как слон влез в мою пижаму мне узнать не суждено.)** Если отбросить второе предложение, будет совершенно непонятно, кто же был в пижаме. Важно здесь, что каждой из возможных трактовок соответствует своё дерево разбора в рамках грамматики непосредственных составляющих.



Перед вами неполный список задач по обработке языка.

Spelling checking -- проверка правописания.

Hyphenation -- расстановка переносов.

Keyword extraction -- выделение ключевых слов.

Language modeling -- языковое моделирование.

Part-of-speech tagging -- частеречная разметка,

Stemming / Lemmatization -- приведение текстов к нормальной, словарной форме;

Named entity recognition -- выделение именованных сущностей.

Text (topic) classification -- классификация текстов

Syntax parsing -- синтаксический разбор.

Machine translation -- машинный перевод

Text clustering -- кластеризация текстов.

Topic modeling -- тематическое моделирование.

QA systems, dialogue systems -- вопросно-ответные и диалоговые системы

Text summarization -- автореферирование
Plagiarism detection -- обнаружение плагиата

Spelling checking -- проверка правописания. Здесь всё понятно, во многих продвинутых текстовых редакторах есть такая функция, в том числе для русского языка.

Hyphenation -- автоматическая расстановка переносов.

Keyword extraction -- выделение ключевых слов. Их извлекают из текстов с самыми разными целями. Например, для поиска сложных терминов, или понимания в первом приближении, каким проблемам посвящены исследуемые тексты.

Language modeling -- языковое моделирование. Задача предсказания следующего слова по нескольким предыдущим. Имеет очень много приложений в современном NLP.

Part-of-speech tagging -- частеречная разметка, как все мы делали в школе. По сути задача разрешения морфологической неоднозначности.

Stemming / Lemmatization -- приведение текстов к нормальной, словарной форме; как мы видим из примеров трудностей в обработке языка, это не всегда тривиальная задача, иногда не удаётся определить и часть речи без контекста.

Named entity recognition -- выделение именованных сущностей. Далеко не всегда просто выделить из текста топонимы, названия организаций, имена и так далее. Широко используется во многих прикладных системах.

Text (topic) classification -- одна из самых часто решаемых задач в обработке языка. При наличии некоторых меток у текстов (например, положительный тон у рецензии или нет), суметь расставлять метки у неаннотированных ещё текстов. Сюда относятся анализ тональности, классификация новостей по тематикам, выявление токсичных комментариев и многое другое.

Syntax parsing -- синтаксический разбор. Тоже знакомая всем школьная задача.

Machine translation -- машинный перевод, очень далеко шагнувший в последние годы по сравнению с системами, распространявшимися на сиди-дисках в нулевых.

Text clustering -- группировка похожих текстов и разделение непохожих без всякой человеческой разметки. В качестве примера можно вспомнить агрегаторы новостей, которые зачастую удивительно хорошо склеивают новостные статьи по событиям.

Topic modeling -- тематическое моделирование. Задача, похожая на кластеризацию. Выделение из набора текста тематик и сопоставление каждому документу их распределения. Часто используется в социальных науках для анализа неразмеченных текстов и для разведочных поиска и анализа данных. Тематическое моделирование позволяет быстро дать ответ на вопрос, о чём же все эти тексты, так как каждой тематике сопоставлено распределение наиболее вероятных из составляющих её слов.

QA systems, dialogue systems -- вопросно-ответные и диалоговые системы. Например, автоматические помощники, которые внедряют всё чаще мобильные операторы, банки и другие коммерческие компании для быстрых ответов на, как правило, простые вопросы пользователей. Сюда же можно отнести и популярных встроенных в мобильные устройства виртуальных ассистентов последних лет.

Text summarization -- автореферирование, сложная задача составления краткого содержания сравнительно большого текста.

Plagiarism detection -- обнаружение плагиата. Студентам не нужно объяснять, что такое системы обнаружения плагиата, так как через них, как правило, проходят все курсовые и дипломные работы. Эту задачу также относят к NLP.

Этот список -- далеко не полный. Есть и другие практически важные, и более эзотерические задачи. Кроме того, некоторые задачи имеют смысл только в отдельных языках. Например, в некоторых иероглифических системах поделить текст на слова -- уже проблема.

Конечно, всего в этом курсе мы не рассмотрим, пока займёмся основами. Но в области обработки естественного языка ещё много задач, в которых на момент 2020 года нет удовлетворительного решения; особенно если речь идёт не об английском. NLP сейчас -- очень конкурентная область, но работы много, добро пожаловать.

Краткая и неполная история компьютерной лингвистики

Докомпьютерная эра

Начнём с докомпьютерной эры. Если не считать различных опытов с шифрованием текстов, первый вычислительный подход к языку, о котором хочется обязательно упомянуть -- это применение великим русским математиком Андреем Андреевичем Марковым разработанной им в тот момент модели к тексту поэмы Александра Сергеевича Пушкина “Евгений Онегин”. Эту модель позже назовут марковской цепью.

Отступление: эксперимент с “Онегиным” и цепи Маркова

Сам Марков не рассматривал никаких естественнонаучных или технических приложений цепей. В работе «Замечательный случай испытаний, связанных в цепь», которая помещена в конце книги [Марков, 1924], Марков исследует чередование гласных и согласных букв в русском языке. Для этого он берет последовательность из 20 000 букв из «Евгения Онегина» А. С. Пушкина. Он пишет, что эту последовательность с некоторым приближением можно рассматривать как простую цепь. Марков исследовал также последовательность из 100 000 букв из «Детских годов Багрова-внука» С. Т. Аксакова.

Марков искал вероятность того, что наугад взятая буква из русского текста будет гласной. Эта вероятность зависит от того, гласной или согласной будет предшествующая буква. Для «Евгения Онегина» вероятность появления гласной буквы после гласной равна $\alpha = 0{,}128$, а гласной после согласной $\beta = 0{,}663$.

Простая цепь Маркова обладает эргодическим свойством, т. е. свойством того, что предельное распределение не зависит от начального состояния.

В этом случае

$$\lim_{s\to\infty} \alpha_s = \lim_{s\to\infty} \beta_s = \frac{\beta}{1-\alpha+\beta}.$$

Майстров Л.Е. Развитие понятия вероятности

v – гласная, c – согласная

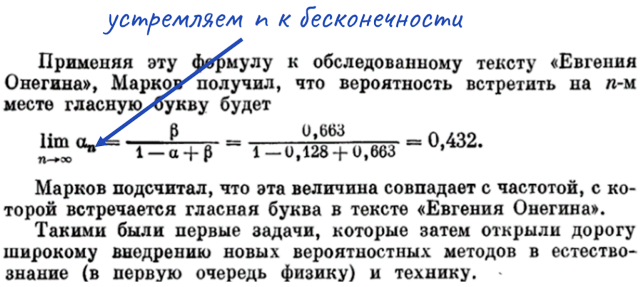
$\alpha = p(v \mid v) = 0.128$
$\beta = p(v \mid c) = 0.663$

Марковское свойство:

p(c, c | v) =
p(c | v, c) p(c | v) =
p(c | c) p(c | v)

Вероятность гласной на n-й позиции:

p(*, *..., v) =
p(c, ..., v) + ... + p(v, ..., v)



Простая цепь Маркова -- последовательность событий, в которой вероятность очередного события зависит только от конечного числа предыдущих. Например, одного. Андрей Андреевич Марков рассматривал в качестве таких событий появление гласной или согласной букв в тексте и предполагал, что гласной или согласной очередная буква будет только в зависимости от того к какому из двух классов относится текущая буква. Ведь чаще всего за согласной следует гласная -- и наоборот. При этом мы считаем, что весь предшествующий контекст неважен.

Рассмотрев первые 200 000 знаков “Евгения Онегина” великий математик без всяких компьютеров вручную подсчитал, что за гласной гласная идёт в 12.8% случаев, а за согласной гласная в 66.3%.

Как оценить вероятность, что, наткнувшись на гласную букву, мы увидим, что следующие две будут согласными? В рамках простой цепи Маркова нам просто нужно перемножить вероятности “согласная после гласной” и “согласная после согласной”. То же верно и для более длинных последовательностей. Используя простейшие свойства вероятностей можно для любой по номеру N буквы в тексте оценить вероятность, что она окажется гласной или согласной. Для этого нужно просуммировать вероятности всех цепочек, длины N, которые заканчиваются гласной буквой.

Важнейшее свойство марковской цепи -- для больших значений N вероятности, что буква на позиции N будет гласной или согласной, почти перестают меняться и не зависят от начального состояния. Это свойство называется эргодическим, вероятности при бесконечно большом N называют предельными.

Андрей Андреевич Марков вычислил с помощью этого свойства вероятность, что случайно взятая буква будет гласной, и она совпала с долей гласных букв в рассматриваемом тексте.

Компьютерная эпоха

Однако по-настоящему история компьютерной лингвистики началась гораздо позже. Она очень непроста. Сложно даже перечислить основные вехи так, чтобы случайно что-нибудь не забыть и кого-нибудь не обидеть, обделив веткой пальмы первенства. Дело в том, что с развитием электронно-вычислительной техники сразу появились задачи по работе с человеческим языком, и в нескольких странах одновременно начались исследования, посвящённые различным человеко-машинным взаимодействиям с помощью естественного языка. В первую очередь в Советском Союзе и Соединённых Штатах Америки.

На протяжении всей истории вычислительной лингвистики одной из самых важных и сложных задач был и остаётся машинный перевод. Сейчас большинство успешных систем машинного перевода работают на алгоритмах машинного обучения, требующих большого количества пар предложений на двух языках. Но в далёком 1954 году многие верили, что правилами можно обойтись, именно тогда в США состоялась демонстрация автоматического перевода 60 предложений с русского языка на английский. Система перевода работала всего на 6 грамматических правилах и была совместным проектом IBM и Джорджтаунского университета. Демонстрация силы, вау-эффект и видимость успеха помогли убедить что в развитие машинного перевода нужно вкладывать большие деньги.

Однако вскоре оказалось, что задача не решается правилами, и через 12 лет финансирование было значительно урезано.

Нужно также отметить, что Джорджтаунский эксперимент -- это самый громкий, но не первый подход, и что в те же годы в нескольких институтах в Советском Союзе также над машинным переводом работало несколько исследовательских групп.

Ещё один эксперимент из эпохи оптимизма, о котором нельзя не упомянуть, -- это Элиза, один из первых чат-ботов, виртуальных ассистентов, разработанный в недрах лаборатории искусственного интеллекта в Массачусетском технологическом Джозефом Вейценбаумом.

Элиза была разработана как пародия на так называемого клиент-центрированного психотерапевта, практикующего метод активного слушания. То есть по большей части она, перефразируя, обращала все реплики собеседника в вопросы ему же. Давайте посмотрим на типичный диалог с Элизой.

Human: Well, my boyfriend made me come here.

ELIZA: Your boyfriend made you come here?

Human: He says I'm depressed much of the time.

ELIZA: I am sorry to hear you are depressed.

Human: It's true. I'm unhappy.

ELIZA: Do you think coming here will help you not to be unhappy?

Человек: Что ж, сюда прийти меня заставил мой молодой человек.

ELIZA: Вас сюда заставил прийти ваш молодой человек?

Человек: Он говорит, что я постоянно чем-то подавлена.

ELIZA: Как жаль, что вы чем-то подавлены.

Человек: Всё так и есть. Я несчастна.

ELIZA: Полагаете, ваш визит сюда поможет вам перестать быть несчастной?

Как нетрудно догадаться, Элиза -- тоже система работающая на правилах. Её ответ выбирался в соответствии с шаблоном, под который подходила фраза пользователя, она ничему не училась и не держала контекст. И даже при этом у некоторых из её собеседников создавалось впечатление, что Элиза эмоционально вовлечена в беседу и проявляет к ним интерес. Поэтому антропоморфизацию вычислительной техники, когда её пользователь знает, что имеет дело с компьютером, -- называют **эффектом Элизы**. С тех пор утекло много воды, и сейчас самые современные и сложные чат-боты работают на нейронных сетях, но обо всём расскажем по порядку.

Увы, для краткости повествования нам придётся опустить многие важные события.

50-е

50-е годы двадцатого века, как мы уже поняли, ознаменовались первыми попытками к решению задач, которые сейчас относят к вычислительной лингвистике. В 1948 году вышла статья Клода Шеннона “Математическая теория связи”, в которой были изложены основы теории информации. В 1957 году увидела свет совершившая революцию в лингвистике работа “Синтаксические структуры” Ноама Хомского.

60-е и 70-е

60-е и 70-е годы можно назвать эрой взлёта и падения символического искусственного интеллекта, то есть веры в то, что можно создавать разумные машины с помощью методов логического вывода и экспертных систем. В 60-е началась и разработка советскими лингвистами теории “Смысл-Текст” Игоря Александровича Мельчука, которая оказала значительное влияние на отечественную лингвистику, и о состоятельности

которой много лет велись жаркие споры. В 60-х был разработан корпус английского языка Университета Брауна (Браун-корпус), один из важнейших лингвистических ресурсов многих последующих лет, от момента создания которого принято отсчитывать существование корпусной лингвистики. Один из величайших учёных двадцатого века Андрей Николаевич Колмогоров занимается применением матметодов к анализу русского стиха.

80-е и 90-е

В 80-е и 90-е годы в обработку естественного языка, открыв дверь ногой, приходит машинное обучение. Нормой становятся мероприятия по численной оценке качества алгоритмов анализа текста -- на основе различных наборов данных. В советском союзе в 1985 году академик Андрей Петрович Ершов инициирует создание Машинного фонда русского языка. Все будущие важнейшие компоненты нейронных сетей для обработки текстов изобретаются в это время. Применяются структурные модели машинного обучения, например, для распознавания речи. Разрабатываются количественные подходы к дистрибутивной семантике.

2000-е

2000-е -- начало эпохи интернета, с каждым годом текстовых данных, доступных в сети становится всё больше. Открывается интерфейс к Национальному Корпусу Русского языка. Машинное обучение колонизирует обработку естественного языка. Хранение данных и вычислительные мощности становятся всё дешевле. Тем, кто сталкивается с задачами обработки текстов, приходится иметь дело со всё бОльшими и подчас меняющимися во времени наборами данных. Поэтому растёт интерес к так называемым методам “без учителя“ -- алгоритмам, пытающимся без всякой разметки и без экспертного мнения выделить в данных структуру -- сгруппировать похожие тексты, выделить тематики из набора текстов, о тематической структуре мы можем ничего не знать априори -- и так далее.

2010-е

2010-е, как многие, думаю, слышали, это эпоха гегемонии одной из важных моделей машинного обучения -- нейронных сетей. Эпоха так называемого глубокого или, как иногда говорят, глубинного обучения. В связи с относительной дешевизной хранилищ данных и вычислительных ресурсов, появлением новых методов обучения и инструментов, упрощающих жизнь исследователям, использование нейронных сетей стало эффективным и бьёт все рекорды по качеству предсказаний в компьютерном зрении и обработке естественного языка. В работе с ними есть много тонкостей -- **и** в том числе скорость предсказаний, необходимость наличия мощных вычислительных ресурсов -- так что на практике нейронные сети далеко не всегда первый метод, который стоит попробовать в надежде на мгновенный успех. Чтобы сохранить длительность и сложность курса подходящей для того, чтобы считать его вводным, мы не будем обсуждать нейронные методы, но очень рекомендуем материалы стенфордского курса по глубокому обучению для обработки естественного языка.

Готовим тексты для работы с машиной

В зависимости от выбранной задачи и метода, тексты можно подготовить для работы с компьютером по-разному. Скажем, для анализа тематик новостей, нам бывает достаточно каждую новость представить как множество слов. Для разметки частей речи или синтаксического разбора порядок слов нам важен, поэтому мы рассматриваем тексты как цепочку слов. И так далее.

Давайте начнём с простых задач предобработки текста на естественном языке, которые приходится решать почти каждый раз.

Итак, перед нами тексты -- допустим, на русском языке. И, допустим, наша задача для начала подготовить тексты так, чтобы мы, как поисковик, могли по запросу выбрать все документы, содержащие то или иное слово.

Изначально для машины каждый из наших текстов просто строка -- конечная последовательность символов -- букв, пробелов, знаков препинания и так далее. Конечно, чтобы осуществлять эффективный поиск по словам, нужно разделить строку на подстроки -- отделить друг от друга слова и знаки препинания. **Эта задача разбиения строки на так называемые токены называется токенизацией**. Обычно не нужно программировать её самостоятельно, есть много готовых инструментов, которые всё сделают за вас. В их основе часто эвристические правила вида “токен -- это подпоследовательность букв, отделённая от остального текста пробелами и запятыми или точками”.

Важно! Если вы пользуетесь каким-то лингвистическим инструментом, который принимает на вход готовые токены, при обработке ваших текстов важно использовать именно тот токенизатор, на который рассчитывает инструмент. Например, некоторые токенизаторы опускают знаки препинания, а некоторые выделяют их в отдельный токен. Алгоритм, не приспособленный к работе со знаками препинания, или напротив, использующий их, может потерять в качестве своей работы, если тексты будут разбиты на токены неподходящим образом. То же касается не только токенизации, но и других этапов подготовки текста. Вернёмся к ним.

Не всегда, но зачастую смысл отдельных слов нам важнее, чем их падеж, число, род и так далее. Скажем, если мы хотим найти все документы, содержащие слово “крот”, нам явно будут нужны и документы со словами “кротик”, “крота”, “кроту”, “кроты”, “кротами” и так далее. Для этого исходные слова преобразуются так, чтобы различные формы слова, преобразовывались в какой-то единый для них токен. Есть два стандартных способа.

Стемминг

Первый способ -- так называемый **стемминг**. В разных источниках стеммингом называют разное, но обычно подразумевают, что стемминг -- это попытка выделить часть слова, общую для всех его морфологически выводимых форм, в английском она называется “stem”. В примере с “кротом” таким идеальным общим для всех форм фрагментом будет “крот”. Обратите внимание, что стем -- это не всегда слово.

Гипотетический стеммер из глагола “солить”, “солю”, “солим”, “солите” -- выделит “сол”. Технически, сол -- это слово, которым обозначают марсианские сутки (в NASA даже придумали слова yestersol, solmorrow и tosol), но всё же к соли они не имеют никакого отношения.

Стеммер Портера

Первый по-настоящему успешный и **до сих пор иногда использующийся стеммер придумал Мартин Портер**. Он так и называется в литературе -- стеммер Портера. Принцип работы -- последовательное применение к каждому слову нескольких правил преобразования строк. Алгоритм подробно описан на сайте автора. Мы не будем погружаться в детали, но дадим исходный алгоритм очень крупными мазками.

Шаг 1a.

SSES -> SS

IES -> I

SS -> SS

S ->

Эти правила преобразовывают постфиксы (то есть концы) слов. Здесь мы избавляемся, например, от множественного числа существительных и единственного числа глаголов. Если слово заканчивается на **sses**, то этот постфикс будет подменён на **ss**. Например, caresses -> caress.

В этом примере слово осталось словом. А вот второе правило явно превратит многие слова в неслова. Например ties → ti, ponies -> poni через i -- и так далее.

Шаг 1b.

(m>0) EED -> EE

(*v*) ED->

(*v*) ING ->

“Эм больше нуля” -- это оценка, сколько раз до суффикса строки EED (и-и-ди) нам встретилась пара рядом идущих букв вида “гласная, потом согласная”. Так слова **feed** и **breed** мы сокращать не должны, а вот **disagreed** уже обрежем.

“Вэ в звёздочках” требует, чтобы в стеме были гласные буквы. Так, bed обрезать не удастся, а из carved выкинем окончание. То же работает с герундиями, заканчивающимися на ing. Замысел понятен -- здесь мы тоже избавляемся от окончаний, без которых общий смысл слова не должен потеряться.

Если применились второе и третье правила на этом шаге, мы как бы восстанавливаем форму глагола, выкидывая при необходимости удвоенные согласные на конце и добавляя букву E, когда следует.

AT -> ATE
BL->BLE

IZ->IZE
(*d and not (*L or *S or *Z))->single letter
(m=1 and *o)->E

Шаг **1c** заменяет, когда уместно, уай на ай на концах слов.

(*v*) Y->I

Мы рассмотрели первый, самый сложный шаг стеммера Портера, работающий с временами и множественными числами. Общая идея ясна, а дальше, на этапах со второго по пятый, всё гораздо проще. Полный алгоритм описан по ссылке[1] на сайт автора.

У статьи Мартина Эф Портера **An algorithm for suffix stripping** на 2019 год более десяти с половиной тысяч цитирований в работах других исследователей по данным Google

[1] **http://snowball.tartarus.org/algorithms/porter/stemmer.html**

Scholar. Со стеммером Портера проще всего ознакомиться “в бою”, воспользовавшись библиотекой NLTK, с которой мы ещё не раз столкнёмся в этом курсе. Автор стеммера разработал специальный фреймворк для задания своих стеммеров преобразований. Он называется Snowball.

Лемматизация

Второй и, как правило, предпочтительный способ приведения разных форм слова к некоторой единой сущности, называется **лемматизацией**, то есть приведением словоформ к лемме, то есть словарной форме. Скажем, лемматизация слова “рычали” даст нам “рычать”, слова “укуса” -- “укус”, “острому” -- “острый” и так далее. Отметим, что в литературе часто лемматизацию называют одним из вариантов стемминга.

Морфологический вывод

Лемматизаторы, чаще прочих использующиеся на практике, как правило, используют определённым образом обработанные словари -- то есть списки слов. Однако, увы, нет и не может быть универсального словаря, так как языки постоянно растут. Поэтому требуются некоторые правила, как привести к лемме неизвестное лемматизатору слово лишь по его написанию и, возможно, окружающему тексту. Очевидно, человеку это под силу, давайте вспомним Кэрролла

А в глу́ше ры́мит исполин —
Злопастный Брандашмыг!

И ещё один классический пример из лекций академика Льва Владимировича Щ**е**рбы:

Гло́кая ку́здра ште́ко будлану́ла бо́кра и курдя́чит бокрёнка

Здесь нам помогают окончания. Мы можем определить часть речи, род, число, падеж и время. Так получится сделать не во всех языках, и в этом курсе мы не будем

углубляться в их классификацию по роли в каждом из них морфологии. Но в русском языке отдельные морфемы могут дать очень много информации о происходящем, даже если смысл корней слов нам неизвестен.

Для сравнения, в более современном литературном источнике -- в дебютном романе Владимира Георгиевича Сорокина “Норма” -- используется своеобразный художественный приём. Часть слов персонажей заменяется на слова, которым явно не присуща морфология русского языка.

— Ну, это слишком серьёзный **пловркнрае**, — усмехнулся зам главного редактора.
[...]
— Старичок, но **домлоанр говпр**, **дочапвепк нав**! — засмеялся Александр Павлович.

В первом случае мы, конечно, можем догадаться, что скорее всего, *пловркнрае* -- это существительное, вероятно, похожее по смыслу на слово “вопрос”. Но мы это вывели из контекста, то есть только благодаря окружающим русским словам и их положению в предложении. А вот во второй реплике нам остаётся только догадываться, какую остр**о**ту отпустил Александр Павлович, и как мы могли бы сделать разбор предложения. Контекстной информации слишком мало даже для того, чтобы понять, из каких частей речи составлена фраза.

Морфологическая неоднозначность

Как мы уже отметили, часто для приведения слов к словарной форме, алгоритмы учитывают и окружающий это слово текст. Зачем это нужно делать?

Попробуем определить словарную форму слова “мыла”. Сразу ясно, где подвох, не так ли? Без контекста мы не можем определить, какая это часть речи, -- либо глагол в прошедшем времени, и тогда лемма “мыть”. Либо имя существительное в родительном падеже. Нет чего? Нет мыла. То есть лемма “мыло”. Однако когда это слово появится в предложении “мама мыла раму”, все наши сомнения отпадут.

Это явление называется морфологической неоднозначностью. Поэтому, несмотря на то, что само написание слов в русском языке нам помогает, размечать части речи всё же нужно с оглядкой на контекст. И поэтому, когда мы хотим получить словарную форму лексемы, стоит пользоваться инструментами работающими с морфологической неоднозначностью. Но этим часто пренебрегают в случаях, когда неправильное определение леммы не вносит большого вклада в результат -- ради быстродействия.

Инструментарий для русского языка

Для лемматизации на практике используют разные инструменты. Рассмотрим несколько средств для морфологического анализа, широко используемых на момент 2019 года.

Pymorphy2

Второй pymorphy -- морфологический анализатор Михаила Коробова, распространяемый как библиотека на языке Python. Пакет умеет приводить слово к словарной форме, ставить в нужную форму и возвращать грамматическую информацию о слове (число, род, падеж, часть речи и т.д.). Работает на основе словарей, а именно на основе разметки OpenCorpora, словоформы хранятся в детерминированном ациклическом конечном автомате (DAFSA), что позволяет компактно хранить огромное количество словоформ и эффективно работать с буквой “ё”, которую иногда подменяют на “е”.

Общий подход, по словам автора, похож на тот, что описан на сайте aot.ru -- сайте другого проекта, очень важного для развития обработки естественного языка в России.

Как работать с пайморфи рассказано в очень подробной документации. Давайте рассмотрим пример.

```
>>> import pymorphy2

>>> morph = pymorphy2.MorphAnalyzer()

>>> morph.parse("мыла")

[Parse(word='мыла',       tag=OpencorporaTag('NOUN,inan,neut
sing,gent'),          normal_form='мыло',          score=0.333333,
methods_stack=((<DictionaryAnalyzer>, 'мыла', 54, 1),)),

     Parse(word='мыла',       tag=OpencorporaTag('VERB,impf,tran
femn,sing,past,indc'),    normal_form='мыть',     score=0.333333,
methods_stack=((<DictionaryAnalyzer>, 'мыла', 1813, 8),)),

     Parse(word='мыла',       tag=OpencorporaTag('NOUN,inan,neut
plur,nomn'),          normal_form='мыло',          score=0.166666,
methods_stack=((<DictionaryAnalyzer>, 'мыла', 54, 6),)),

     Parse(word='мыла',       tag=OpencorporaTag('NOUN,inan,neut
plur,accs'),          normal_form='мыло',          score=0.166666,
methods_stack=((<DictionaryAnalyzer>, 'мыла', 54, 9),))]
```

Вызвав метод parse от слова “мыла”, мы получили несколько возможных вариантов, то есть несколько объектов класса Parse, которые включают в себя в том числе словарную форму, выведенную грамматическую информацию и некий score. Варианты с наибольшим score -- имя существительное -- единственное число родительный падеж (генитИв) -- и глагол прошедшего времени. Этот параметр -- условная вероятность набора граммем для данного слова (то есть совокупности части речи, числа, падежа и так далее). Она оценивается по той части Открытого корпуса, где снята неоднозначность, как простое отношение частот: сколько раз слово “мыла” встречается как глагол (единственного числа и так далее) делим на общее количество употреблений слова “мыла”.

Пока только так, не используя окружающий контекст, pymorphy2 помогает выбрать вариант в условиях неоднозначного разбора.

Mystem

Майстем -- это морфологический анализатор русского языка с поддержкой снятия морфологической неоднозначности. Первую версию разработали Илья Сегалович и Виталий Титов в компании «Яндекс». Программа работает на основе “Грамматического словаря русского языка” Андрея Анатольевича Зализняка и способна формировать морфологические гипотезы о незнакомых словах. Распространяется через сайт “Яндекса” в виде исполняемого файла, код закрыт. По лицензии исполняемый файл нельзя использовать только в случае, если вы разрабатываете какой-то продукт, потенциально конкурентоспособный по отношению к какому-либо сервису “Яндекса”.

Так как нас интересует возможность дальнейшей обработки того, что будет на выходе после обработки майстемом, стоит упомянуть хотя бы одну из обёрток на языке Python, с которым мы будем иметь дело в ходе работы над заданиями.

Pymystem3 -- библиотека, разработанная Денисом Сухониным и Александром Панченко. Интерфейс очень простой и интуитивный, лемматизация выполняется в пять строчек

```
>>> from pymystem3 import Mystem

>>> mystem = Mystem()

>>> text = 'Как насчёт небольшого стемминга'

>>> lemmas = mystem.lemmatize(text)

>>> print(''.join(lemmas))
```

как насчет небольшой стемминг

(пример позаимствован с сайта nlpub.ru)

Попробуем применить mystem к нашему примеру с мамой, моющей раму. Сначала старый вариант, без работы с морфологической неоднозначностью.

```
>>> from pymystem3 import Mystem

>>> mystem = Mystem(disambiguation=False)

>>> text = "Мама мыла раму."

>>> lemmas = mystem.lemmatize(text)

>>> " ".join(lemmas)

'мама   мыло   рама . \n'
```

А вот вариант с обработкой неоднозначности.

```
>>> mystem = Mystem(disambiguation=True)

>>> text = "Мама мыла раму."

>>> lemmas = mystem.lemmatize(text)

>>> " ".join(lemmas)

'мама   мыть   рама . \n'
```

Как мы видим, на этот раз всё верно.

Mystem также выдаёт всё, что он смог понять о тексте, поступившем на вход

```
>>> mystem.analyze(text)

[{'analysis': [{'gr': 'S,жен,од=им,ед', 'lex': 'мама'}],
'text': 'Мама'}, {'text': ' '}, {'analysis': [{'gr':
```

```
'V,несов,пе=прош,ед,изъяв,жен',    'lex':    'мыть'}],    'text':
'мыла'},     {'text':     '     '},     {'analysis':     [{'gr':
'S,жен,неод=вин,ед', 'lex': 'рама'}], 'text': 'раму'}, {'text':
'.'}, {'text': '\n'}]
```

Эту информацию можно в дальнейшем также использовать для решения различных задач. Посмотрите документацию, у майстема есть различные возможности форматирования вывода, которые делают его удобным.

RNNMorph

RNNMorph Ильи Гусева -- на момент середины 2019 года один из самых точных морфологических анализаторов для русского языка, работающий на рекуррентных нейронных сетях и использующий pymorphy2 как компонент.

```
>>> from rnnmorph.predictor import RNNMorphPredictor

>>> predictor = RNNMorphPredictor(language="ru")

>>> forms = predictor.predict(["мама", "мыла", "раму"])

>>> forms

[<normal_form=мама;           word=мама;           pos=NOUN;
tag=Case=Nom|Gender=Fem|Number=Sing;               score=0.9998>,
<normal_form=мыть;              word=мыла;              pos=VERB;
tag=Gender=Fem|Mood=Ind|Number=Sing|Tense=Past|VerbForm=Fin|Voic
e=Act;  score=0.9651>,  <normal_form=рама;  word=раму;  pos=NOUN;
tag=Case=Acc|Gender=Fem|Number=Sing; score=0.5768>]
```

Так как для каждого поступающего предложения запускаются предсказания с помощью нейронной сети на TensorFlow, эта библиотека работает гораздо медленнее, чем рассмотренные выше.

Также среди готовых решений для русского языка стоит обратить внимание на TreeTagger и UDPipe, в разные годы в разных работах показывавшие лучшие результаты.

Выбирать средства для морфологического анализа и нормализации слов -- то есть приведения к нормальной форме -- нужно исходя из требований к задаче, поняв, что важнее -- качество, быстродействие, хорошая лицензия и так далее.