МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БАШКИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ РОМАНО-ГЕРМАНСКОЙ ФИЛОЛОГИИ
КАФЕДРА ЛИНГВОДИДАКТИКИ И ПЕРЕВОДОВЕДЕНИЯ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ПО ПРОГРАММЕ БАКАЛАВРИАТА

САЛИКАЕВОЙ ДАРЬИ ЭДУАРДОВНЫ

МЕТОДЫ РАСПОЗНАВАНИЯ ОСНОВЫ СЛОВА В МАШИННОМ ПЕРЕВОДЕ

Выполнил:
Студент(ка) 4 курса очной формы обучения
Направление подготовки (специальность)
45.03.02 «Лингвистика»
Направленность (профиль) перевод и переводоведение

Руководитель
к.ф.н., доц.
(ученая степень, ученое звание, должность)

Р.Г Мифтахова
(И.О. Фамилия)

Саликаева Д. Э.

УФА – 2020

Оглавление

Введение

Появление и развитие технологий дало возможность машинного перевода, т.е. преобразование текста с одного естественного языка на другой эквивалентный по содержанию текст. Таким образом, возникает и сам вопрос понимания естественного языка. Обработка того или иного естественного языка основывается на особенностях данного языка. Многое зависит от типологической классификации языков т.е. от принадлежности языка к тому или иному типу: флективный, агглютинативный, изолирующий (аморфный) или инкорпорирующий (полисинтетический). Кроме этого учитываются грамматические, фонетические и лексические особенности каждого языка. Таким образом, модели по обработке естественного языка подстраивают под каждый обрабатываемый язык. Создаются и часто обновляются программы для понимания естественного языка. У каждой программы и модели есть свои задачи и методы их выполнения. Чтобы система могла определять естественный язык ей необходимо перевести его в «свой», с этой целью существует огромное количество задач и этапов представляющих собой сложную структуру.

В данной работе рассматривается один из важнейших этапов обработки текста – определение основы слова в машинном переводе, точнее два метода их определения: стемминг и лемматизация.

Актуальность выбранной темы дипломной работы обусловлена постоянным развитием искусственного интеллекта и необходимостью улучшать понимание и обработку естественных языков. Одним из основных свойств естественных языков является их эволютивность, т.е. способность к бесконечному развитию и модификации. Исходя из этого: все что связано с естественными языками является бесконечно-развивающимся процессом – прослеживается необходимость адаптировать искусственные технологии на постоянно изменяющийся процесс.

Предмет исследования: два метода определения основы слова в машинном переводе: стемминг и лемматизация.

Объект исследования: лексические базы для стемминга: Porter Stemmer, Lancaster Stemmer и Snowball Stemmer; лексические базы для лемматизации: WordNetLemmatizer и TextBlob.

Теоретической основой исследования служили научные труды ученых-лингвистов и программистов, посвященные обработке естественного языка и машинному переводу.

Цель представленной работы: оценить качество двух представленных методов – стемминга и лемматизации, сравнить и доказать, что они компенсируют друг друга.

Для выполнения цели важно выполнить следующие **задачи**:

1) выяснить на чем основываются стемминг и лемматизация;

2) изложить различия и недостатки стемминга и лемматизации;

3) объяснить причины возникновения неправильного определения или не распознавания основы слова представленными методами и предоставить методы улучшения их качества;

4) сравнить лексические базы стемминга (Porter Stemmer и Lancaster Stemmer) между собой;

5) сравнить лексические базы лемматизации (WordNetLemmatizer и TextBlob) между собой;

6) сравнить стемминг и лемматизацию (Porter Stemmer и WordNetLemmatizer) между собой.

В работе также рассматривается обработка естественного языка, машинный перевод, как одна из важных задач обработки естественного языка, и три его основных подхода, не считая их гибридных форм: 1) машинный перевод на базе лингвистических данных, 2) статистический машинный перевод и 3) нейронный машинный перевод.

В дипломной работе широко используются такие методы исследования, как теоретический анализ и синтез (рассматривается изучаемый объект по частям и после объединяется), изучение научных работ, сравнение, классификация, моделирование (шаблоны на практическую часть).

Практическая значимость исследования заключается в сравнении лемматизации и стемминга с определенными пакетами (WordNetLemmatizer, TextBlob, PorterStemmer, LancasterStemmer и SnowballStemmer) и их эффективности.

База исследования: язык программирования Python.

Гипотезы:

1) лемматизация и стемминг имеют свои недостатки, но способны компенсировать друг друга

2) результат определения базовой формы слова зависит от используемой лексической базы.

Структура данной работы обусловлена предметом, целью и задачами исследования. Работа состоит из введения, двух глав и заключения, а также списка использованных источников и литературы.

Глава 1.Обработка естественного языка

1.1 Обработка естественного языка: понятие и основные задачи

Обработка естественного языка (NLP — Natural language processing) — область, находящаяся на пересечении компьютерных наук, искусственного интеллекта и лингвистики.

Для более глубокого понимания, фраза была разделена на смысловые сегменты.

Обработка естественного языка

обработка естественный язык

В толковом словаре Ушакова слово «обработка» определяется следующим образом: «1**.** *только ед.* Действие по гл. "обработать"-"обрабатывать". *«Стоит только улучшить использование машин и тракторов, стоит только улучшить обработку земли - и мы добьемся того, что увеличим количество наших продуктов вдвое, втрое.» Сталин*. Обработка металла. Обработка статьи. **2.** Результат такого действия. Сказки Пушкина - обработки народных сказочных сюжетов. [Ушаков 2014: 379]

Итак, вывод: обработка - это либо действие, либо результат действия.

Определение естественного языка в «Википедии» - есте́ственный язы́к - в философии языка и лингвистике язык, используемый для общения людей (в отличие от формальных языков и других типов знаковых систем, также называемых языками в семиотике) и не созданный целенаправленно (в отличие от искусственных языков) [25]. Выделяя важные моменты получается, что естественный язык – “язык, используемый для общения людей” и “не созданный целенаправленно” то есть, это язык на котором общаются люди между собой, и он в какой-то мере спонтанный.

В результате, отмечаем, что обработка естественного языка – это действие с результатом, совершаемое над языком людей, общающихся «спонтанной» речью. Каждый день люди обмениваются бесчисленными

словами с другими людьми с самыми разными целями. Но общение – это гораздо больше, чем просто слова: есть контекст, язык тела, интонация и многое другое, что помогает нам понять смысл слов в процессе общения. Естественные языки невероятно сложны для обработки. Именно это делает обработку естественного языка, способность машины понимать человеческую речь, тем, что имеет огромный потенциал так сильно повлиять на наше современное существование. На сегодняшний момент имеется широкий спектр приложений, за которые отвечает обработка естественного языка.

Процесс обработки естественного языка включает в себя следующие этапы:

1) Восприятие запроса на естественном языке. Запрос может быть написанным, голосовым или текстовым.

2) Анализ запроса. Анализ происходит в лингвистическом процессоре (рис.1), который представляет собой сложную структуру, состоящую из последовательностей соединённых блоков и использующую многостороннюю лингвистическую информацию и информацию об объективной реальности. Лингвистический процессор выявляет и исправляет ошибки во входных данных, что включает токенизацию, морфологический анализ каждого токена, синтаксический и семантический анализ всей входящей фразы. Таким образом лингвистический процессор преобразовывает язык входных данных в некоторый внутренний язык.

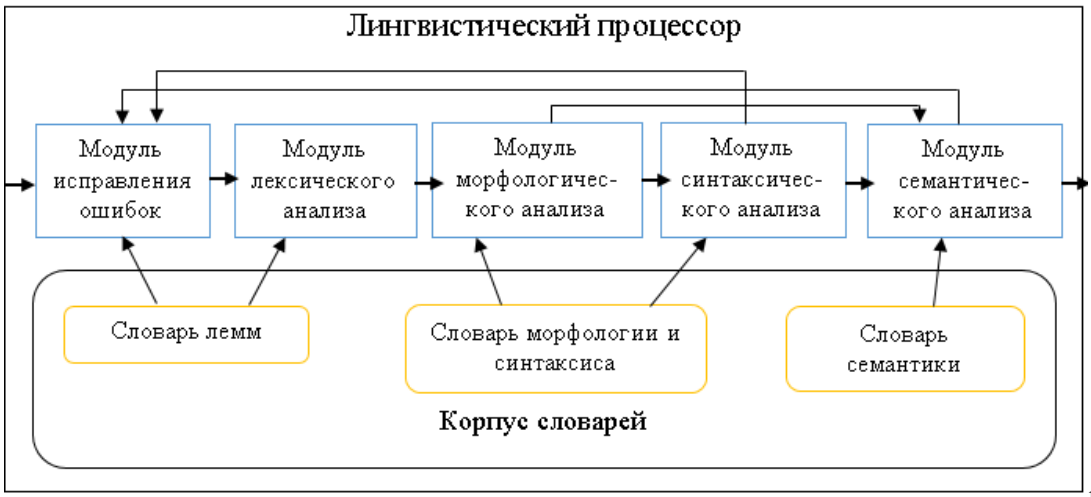


рис.1

3) «Понимание» смысла запроса – формальное описание задач, которые предстоит выполнить системе.

4) Генерация ответа – для систем и приложений, предполагающих некого рода общения с пользователем (например, чат-боты).

5) Реализация ответа.

Теоретически можно выделить 2 типа обработки: обработка текстов и обработка речи, которые в свою очередь делятся на анализ речи, синтез речи, анализ текстов, синтез текстов (рис.2).

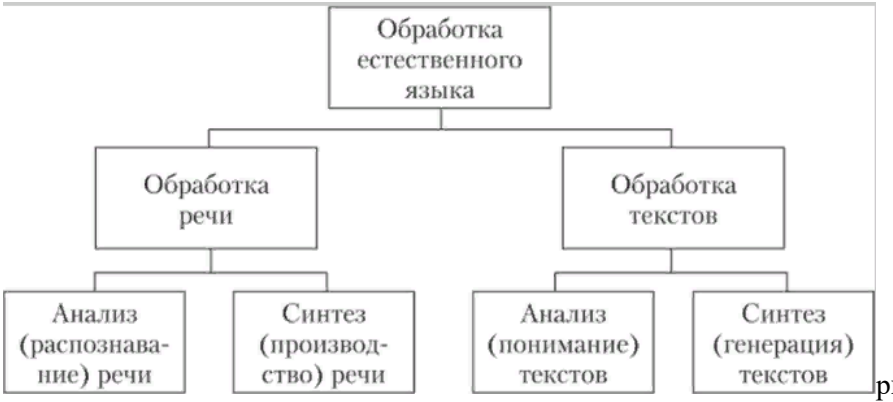


рис.2

Приведенная классификация полезна только с теоретической стороны, но с практической может вызвать некоторые трудности, так как многие прикладные задачи невозможно четко отнести к одному из типов. Например, разработка вопросно-ответных систем, способных принимать вопросы и отвечать на них на естественном языке — это сложная практическая задача, относящаяся как минимум к двум классам (понимание речи и синтез текстов).

Обработка естественного языка предполагает решение множества задач. Здесь будут представлены только основные, которые используются в повседневной жизни:

1) распознавание речи

существует целый ряд программ распознавания речи, которые позволяют декодировать человеческий голос. Это мобильная связь, бытовая автоматизация, громкая связь, виртуальная помощь, видеоигры и другие. В целом, эта технология используется для замены других методов ввода,

например, ввода с клавиатуры. Сегодня распознавание речи является элементом искусственного интеллекта, например, голосовых помощников Cortana, Google Assistant, Siri, Алиса и др.

2) анализ эмоциональной окраски высказываний

выявляет субъективную информацию в тексте. Например, это может быть обзор фильма или эмоциональное состояние, вызванное этим фильмом. Данная функция очень полезна коммерческим компаниям, чтобы проверять удовлетворены ли клиенты конкретными товарами или услугами. Люди охотно делятся своим мнением в социальных сетях. Поиск негативных текстов и выявление основных жалоб существенно помогает изменить концепции, улучшить качество продукции и рекламы, а также снизить уровень неудовлетворенности. В свою очередь, явные положительные отзывы повышают рейтинги и спрос.

3) классификация текстов

- это классификация свободных текстов в предопределённые категории. Классификаторы текста могут использовать для организации, структурирования и категоризации практически что угодно. Например, проходит сортировка по определенным категориям. Приходит новый документ, и необходимо определить, к какой категории он принадлежит. Используя ОЕЯ, классификаторы текста могут автоматически анализировать текст, а затем назначать набор предопределенных тегов или категорий на основе его содержимого.

4) сортировка писем

При помощи ОЕЯ система сортирует сообщения в почтовом ящике, и определяет какие являются спамом и должны быть отсортированы. Естественно, что представленная задача широко используется в почтовых ресурсах, таких как, например, почта Mail.ru, Gmail.com и другие.

5) ответы на вопросы

касается построения систем, автоматически отвечающих на вопросы, заданные людьми на естественном языке. Покупки в интернете,

взаимодействие с чат-ботом на веб-сайте являются алгоритмами, которые используют обработку естественного языка, чтобы понимать запрос и отвечать на вопросы адекватно, автоматически и в режиме реального времени.

6) извлечение и аннотирование информации

извлечение и синтез информации из различных текстовых источников, таких как новостные сообщения, руководства пользователя и многое другое. Это процесс создания резюме и получения краткой характеристики содержания массивных текстовых документов. Важным преимуществом выполнение данной задачи является сокращение времени чтения. Вот некоторые из программных интерфейсов: Aylien Text Analysis, Summarization MeaningCloud, ML Analyzer, Summarize Text, Text Summary.

7) проверка орфографии

это программный инструмент, который выявляет и исправляет любые орфографические ошибки в тексте. Автоматические исправления, проверка грамматики и орфографии, а также автоматическое завершение-все эти функции обеспечиваются обработкой естественного языка. Одним из примеров является приложение Grammarly – онлайн-проверка грамматики, которая сканирует ваш текст и находит все типы ошибок, начиная от опечаток и заканчивая проблемами со структурой предложений и так далее.

8) машинный перевод

это огромное приложение для ОЕЯ, которое позволяет преодолевать языковые барьеры в общении с людьми, а также понимать технические руководства и каталоги, написанные на иностранном языке. Например, широко известные сервисы Google Translate и Яндекс переводчик.

В задачах ОЕЯ часто используются методики машинного обучения. Независимо от представленной задачи в обработке естественного языка есть общая последовательность этапов их решения, к ним относятся: 1) определение задачи (знание сущности задачи определяет алгоритм действий); 2) выбор модели (выбор соответствующего алгоритма); 3) создание и обучение модели (процесс выполнения определенного алгоритма с набором данных,

описывающих выбранную модель); 4) проверка модели (оценка качества модели в сравнении результата применения модели с правильным результатом); 5) практическое использование модели (применение проверенной модели непосредственно к поставленной задаче). [Риз 2016: 45].

Реализация этапов может несколько отличатся в зависимости от поставленных задач, но последовательность этапов решения задач повторяется с небольшими вариациями.

1.2 Машинный перевод как одна из задач обработки естественного языка

Машинный перевод – процесс перевода текстов, как письменных, так и устных, с одного языка на другой. Идея машинного перевода проста- разработать компьютерные алгоритмы, позволяющие осуществлять автоматический перевод без какого-либо вмешательства человека. Но для искусственного интеллекта полное понимание и воспроизведение смысла языка является чрезвычайно сложной задачей, так как язык имеет свои особенности: язык — это символьная система передачи информации и смысла, сказанного или написанного. Трудности возникают из-за фонетических, грамматических и лексических языковых особенностей – начиная с различий алфавитов и заканчивая национальными несходствами языков. Большую трудность представляют переводы последовательностей (например, предложений) в последовательности, что намного труднее операций с цифрами.

Признан тот факт, что машинный перевод не может обеспечить высокую точность перевода. При обработке на уровне слова, система сталкивается с проблемой синонимов, на синтаксическом уровне не всегда определяется соотношение между лексическими единицами в предложении. [Мифтахова 2014: 24].

Машинный перевод относительно старая задача. Самое первое тестирование было проведено 7 января 1954 и назывался он Georgetown IBM. В данном эксперименте предусматривался полностью автоматизированный перевод более 60 предложений с *романизированного* русского на английский. Этот эксперимент положил начало развитию машинного перевода.

До конца 1980-х годов основной акцент исследования в машинном переводе был на создание лингвистических правил, синтаксического анализа, правил передачи лексических единиц и др. [Мифтахова 2014: 24].

В течении нескольких лет появилось три главных подхода:

1) Машинный перевод на базе лингвистических правил (RBMT – Rule-based machine translation);

2) Статистический машинный перевод (SMT – Statistical Machine Translation);

3) Нейронный машинный перевод (NMT – Neural Machine Translation)

Машинный перевод играет значительную роль в современном мире. Машинный перевод – необходимое условие для работы в таких государственно-политических образованиях, как, например, Евросоюз, для поддержания переговоров и сотрудничества на различных уровнях.

Сегодня существенное развитие МП получает так же в странах Азии, так как существует множество отраслей его применения, от торговых и социальных до военных и политических задач. Сам МП день превратился в одну большую индустрию. Ежедневно Google переводит до 100 миллиарда слов. [Мифтахова 2018:711]

1.2.1 Машинный перевод на базе лингвистических правил

Само название дает понять, что технология RBMT использует большие коллекции лингвистических правил. Коллекции применяются в трех разных фазах: анализ, передача и генерализация. RBMT использует построенные вручную лексические базы перевода, некоторые из которых могут редактироваться и уточняться пользователями для улучшения перевода.

Первые системы RBMT были разработаны в начале 1970-х годов. Наиболее важными этапами этой эволюции были появление следующих систем RBMT:

- Systran (URL: http://www.systran.de/) (одна из самых первых систем машинного перевода, переводит с/на 20 языков; перешел на статистическую систему машинного перевода)
- Японские системы MT (URL: http://aamt.info/english/mtsys.htm, http://www.wtec.org/loyola/ar93_94/mt.htm)

Сегодня другие системы общего RBMT включают:

- Apertium (доступен в 35 языках, создан для языков родственным испанскому)
- GramTrans (для скандинавских языков) [33].

Система RBMT связывает структуры данного входного предложения со структурой требуемого выходного предложения, обязательно сохраняя их уникальное значение. Рассмотрим на примере:

“A girl entered the room”

исходный язык – английский.

требуемый целевой язык – русский.

Для того, чтобы перевести это английское предложение, как минимум, потребуется:

- словарь, который сопоставит каждое английское слово с соответствующим русским.
- правила, представляющие регулярную английскую структуру предложений.
- правила, представляющие регулярную русскую структуру предложения.
- и, наконец, необходимы правила, согласно которым можно связать эти две структуры вместе [34].

Соответственно, можно констатировать следующие этапы перевода:

1-й: получение базовой информации части речи для каждого исходного слова:

a – indefinite article;

girl – noun;

entered – verb;

the – definite article;

room – noun

2-й: получение синтаксической информации о глаголе «entered»:

Here: entered – Past Simple, Active Voice

3-й: анализ исходного предложения:

(a girl) – a female person

В частности, достаточно и частичного анализа, чтобы понять синтаксическую структуру исходного предложения и отобразить его в структуре целевого предложения.

4-й: перевод английских слов на русский

a (category = indef.article) => нет аналога

girl (category = noun) => девочка (категория = существительное)

entered (category = verb) => вошла (категория = глагол)

the (category = def. article) => нет аналога

room (category = noun) => комната (категория = существительное)

5-е: Сопоставление словарных статей в соответствующие словоформы (окончательная генерация):

A girl entered the room => Девочка вошла в комнату.

Подходы, основанные на базе лингвистических правил сосредотачиваются на сопоставлении шаблонов или синтаксических разборов. RBMT является эффективной системой машинного перевода. Этот подход высокоэффективен в конкретных ситуациях, но при генерализации и при неоднозначности терминов эффективность снижается. Более того RBMT является трудоемкой, массивной, энергоемкой и дорогостоящей системой. Трудоемкость объясняется тем, что система использует построенные вручную базы перевода, и т.к. каждый раз база редактируется, т.е. происходит частое добавление новых словоформ или неологизмов, она становится объемным, что, с одной стороны, повышает качество перевода, но, с другой, уменьшает скорость работы. Также для постоянных обновлений и редактирований требуется соответствующий персонал (лингвисты, программисты и.т.д.). Еще одним основным недостатком данной системы является то, что для каждой пары языков используются отдельные системы, один словарь используется только на одно направление, т.е. для перевода с русского на английский и с английского на русский требуются две совершенно разные системы.

Разработчики RBMT попытались устранить некоторые из ограничений RBMT, дополнив свою основную технологию некоторыми методами статистического машинного перевода, и продвигают свои продукты как модель гибридной системы машинного перевода (HMT[1]), например, PROMT DeepHybrid (www.promt.com).

[1] Hybrid Machine Translation –интеграция разных подходов машинного перевода из возможных вариантов машинного перевода.

1.2.2 Статистический машинный перевод

Этот подход использует статистические модели. Впервые он был представлен в 1955 году [Weaver 1955: 15], но интерес к нему возник только после 1988 года, когда исследовательский центр IBM Watson начал его использовать [Brown1988: 71].

Появление параллельных корпусов способствовала развитию статистического машинного перевода. В отличии от RBMT статистический машинный перевод использует двуязычные и одноязычные корпуса текстов, что позволяет переводить с любого языка на любой при наличии корпусов данных языков. Таким образом, качество перевода статистическим методом зависит от объемы того или иного языка. Естественно и вполне логично, что для языков с большим количеством носителей и переведенных текстов объем корпуса большой и в основном в свободном доступе. Также существуют коллекции предложений и их переводов, подобранных автоматически в интернете с помощью специальных алгоритмов, например, сайт paracrawl.eu.

Статистический машинный перевод использует n-grams, последовательность n-элементов. N-граммная модель рассчитывает вероятность последнего слова n-граммы, если известны все предыдущие, таким образом, каждое слово зависит друг от друга. В переводе n-граммная модель, на основе корпусов данных, высчитывает вероятность перевода слов и словосочетаний, т.е. сколько раз то или иное слово с одного языка было переведено так или иначе на другой. Например, допустим, что вероятность перевода английского слова ***cat*** на русский

1) как ***кот*** равна 0,08,

2) как ***кошка*** 0,07,

3) как ***котенок*** 0,003. Подсчитав вероятность система выдает наиболее частотно-используемый вариант перевода.

Одним из применяемых подходов в SMT является теорема Байеса:

$$P(e|f) = \frac{P(e)P(f|e)}{P(f)}$$

Теорема соотносит вероятность события с вероятностью определенного исхода. P(e) — априорная вероятность события e;

Мы можем соотнести $P(e|f)$ – вероятность события *e*, если дан исход *f*, и P $(f|e)$ – вероятность исхода *f*, если дано событие *e*. P(*f*) — полная вероятность наступления события *f*.

Для максимизирования условной вероятности слева нужно максимизировать величину справа. Следующее уравнение называют фундаментальным уравнением машинного перевода [Кан, 201: 11]:

$$P(e|f) = P(e) * P(f|e)$$

Где $P(e)$ является моделью языка, а $P(f|e)$ – моделью перевода. Представленная формула является общей для всех предложений. Для более глубокого понимания рассмотрим конкретный пример с предложением *"I love cats"* высчитываем с помощью google n-grams по данным 2011 года.

start of the sentence

вероятность того, что после "I" в английском языке может следовать "love"

end of the sentence

$$P(\ I\ love\ cats) = P(I|sos) \times P(love|I) \times P(cats|\ love) \times P(eos|cats)$$
$$= 0{,}200358419 \times 0{,}0002773069 \times 0{,}0000001286 \times 0{,}0000000692$$

Как вы уже заметили в этой формуле все вероятности со множеством нулей, и, учитывая тот факт, что если их перемножить – их будет еще больше, используем логарифмы, и получается: $= log_2 1{,}1489 + log_2 1{,}0001871 + log_2 1{,}00000008317 + log_2 1{,}00000000069314 = 0{,}20052$

Таким образом, SMT предусматривает три этапа:

1) языковая модель – правильно подобранное слово в данном контексте;

2) модель перевода – каков наилучший перевод данного слова;

3) метод нахождения правильного порядка слов.

Наиболее часто используемая модель – перевод на основе фраз. Например, английская фразы “ is buying” и “an ice cream” переводятся на русский как “покупает” и “мороженное” соответственно.

Самым известным приложением, использовавшим SMT, вероятно, является Google Translate (с 2006 по 2016) Google translate — система, разрабатываемая компанией Google. Она считывает множество текстов, находит параллели между двумя языками, обрабатывает данные и затем выдает возможные варианты. Данная система не осуществляет прямой перевод для языковых пар, в которые не входит английский язык. То есть, при переводе, например, с русского языка на французский, сначала будет осуществлен перевод с русского на английский язык, а после этого — с английского на французский. Более того, для некоторых языков таких операций больше. Например, тексты на белорусском языке вначале переводятся на русский, после этого на английский и только потом на целевой язык. Это значительно снижает точность перевода и делает систему абсолютно непригодной для перевода ряда текстов.

Другим сервисом, также использующим SMT является Яндекс перевод. Яндекс перевод представляет собой сервис автоматического перевода слов, фраз, целых текстов, а также веб-страниц. Яндекс. Перевод состоит из двух

частей — модели перевода и модели языка. Модель перевода занимается построением графа, содержащего все возможные варианты перевода предложения. Модель языка выбирает лучший вариант перевода с точки зрения оптимальной сочетаемости слов в естественном языке. Хотя список доступных для перевода языков у Яндекс перевода значительно меньше, чем у Google Translate, перевод осуществляется напрямую, без использования промежуточных звеньев. [Андреева 2013: 66]

Преимущества SMT:

1) по сравнению с RBMT требуется меньше редактирований вручную;

2) наличие корпусов позволяет одному SMT переводить с любого языка на любой;

3) уровень перевода выше по сравнению с RBMT: при правильной языковой модели перевод будет более плавным.

Недостатки SMT:

1) требуется двуязычный корпус;

2) трудно исправить некоторые конкретные ошибки;

3) трудности возникают с языковыми парами с большими различиями в порядке слов.

4) качество перевода снижается у языков с небольшим объемом корпусных данных

1.2.3 Нейронный машинный перевод

Нейронная система машинного перевода – подход к машинному переводу, предложенный относительно недавно (2016). Был разработан и предложен компанией Google. В отличие от двух ранее представленных подходов машинного перевода, нейронный нацелен на построение единой нейронной сети, вместо конвейеризации нескольких задач. Нейронная сеть настраивается совместно для максимальной эффективности процесса перевода.

Как и статистический переводчик, нейронная сеть анализирует массив параллельных текстов и учится находить в них закономерности. Сам перевод, однако, устроен по-другому [26]. Нейросеть работает с предложениями целиком, нежели с фрагментами, что обеспечивает более связный и слаженный перевод. Она получает на вход предложение на одном языке, а на выходе выдаёт предложение на другом языке.

Нейронная сеть анализирует «окружение» каждого слова, даже если слова находятся в разных частях предложения (Skip-grams), что позволяет правильно определить контекст, который, в свою очередь, обеспечивает связь слов в предложении при переводе [Мифтахова 2019: 497]

Основная форма NMT состоит из двух компонентов: 1) шифратора (кодера), который вычисляет способ задания функции *s* для каждого исходного предложения и 2) дешифратора (декодера), генерирующего одно целевое слово за один раз и, таким образом, подвергающее декомпозиции условную вероятность. В разных ситуациях используются разные виды нейросетей, к примеру, сверточные нейросети (CNN[2]) отлично подойдут для картинок, так как работают с независимыми блоками пикселей. Но если брать тексты – они представляют собой последовательность слов, в котором каждое слово зависит от «окружающих» его слов – для их обработки лучше подходят

[2] CNN (convolutional neural network) – свёрточная нейронная сеть – предложена Яном Лекуном в 1988 году и нацелена на эффективное распознавание образов.

структуры рекуррентной нейронной сети (RNN[3]), ведь они помнят предыдущий результат, то есть предыдущие слова в предложении. [Luong 2015: 1413]. На рисунке №3 показано что из себя представляет RNN в машинном переводе. Красным отмечены входные данные, синим – вывод и зеленым – RNN. Исходный текст подается в рекуррентную нейронную сеть, которая затем создает переведенный текст в качестве результата вывода;

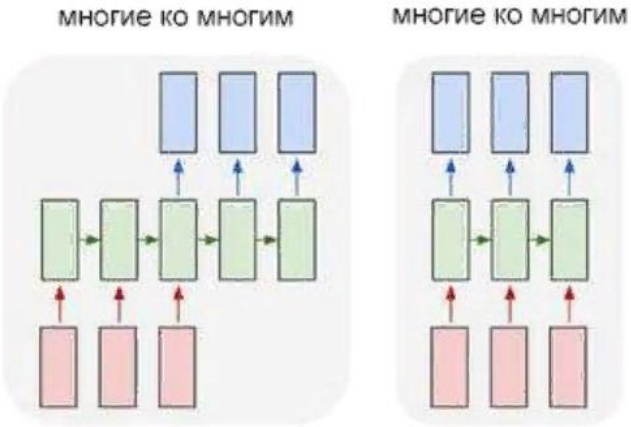


рис.3

Есть еще более сложные специфические виды RNN – LSTM[4] и GRU[5] – решающие конкретные задачи более эффективно. Слова находятся в многомерном векторном пространстве (600-мерные, 300-мерные и.т.д) и каждое слово кодируется в вектор из действительных чисел. «Используя методы для изучения векторов слов… и помещая эти слова в многомерные векторные пространства, можно спровоцировать их работу в качестве семантически связных слов. [Мифтахова 2019:498] Постепенно система сама определяет семантику определенных векторов. К примеру, возьмем ряд слов “mother”, “father”, “sister” и “brother”. Эти слова семантически связаны друг с другом – слова, обозначающие членов семьи. Благодаря семантическим связям нейронные системы эффективно запоминают и переводят, используя синонимы и вообще слова, которые они прежде не находили в «человеческом»

[3] RNN (recurrent neural network) –нейронная сеть с обратными связями т.е. между элементами связи образуют направленную последовательность.
[4]LSTM (long-stem translation memory) – рекуррентные нейронные сети с долгой краткосрочной памятью. Впервые был предложен в 1997 году немецкими исследователями Хохрайтером и Шмидхубером.
[5]GRU (gated recurrent units) – управляемый рекуррентный блок. Представлен в 2014 году.

переводе слов из данного предложения. Помимо семантического сходства слов можно выделить и внешнее (по форме). Слова “cat” и ‘cats’представляют собой сходство по форме и расстояние между векторами для данных слов в многомерном пространстве будет небольшое.

Система нейронного машинного перевода, как и статистическая, использует миллионы пар предложений, перевод которых раннее был сделан человеком. Нейронный машинный перевод на данный момент учится и постепенно сам начинает определять семантику контекста. Программы, используемые для тренировки нейронной сети, позволяют задать параметры нейронной сети, параметры обучения и.т.д. В основном, все программы написаны на языке Python по NLTK[6].

Несмотря на большие преимущества, NMT сталкивается с проблемами при переводе мало распространённых имён, топонимов и других редких слов, также при переводе коротких предложений или словосочетаний. Поэтому многие компании, предоставляющие переводческие услуги предпочитают использовать гибридные системы машинного перевода. Возьмем, к примеру, Яндекс, который официально сообщил об использовании гибридной системы. В Яндекс переводчике перевод выполняется статистической и нейронной моделью, и затем алгоритм на основе метода машинного обучения CatBoost сравнивает результаты и предлагает лучший вариант перевода. Например, допустим, что нейронная система перевела предложение “A girl entered the room” как “Одна девочка вошла в комнату”, а статистический – “Девочка вошла в комнату” далее CatBoost анализирует весь русский корпус и вычисляет вероятность вариантов перевода, затем выдает тот вариант перевода, у которого вероятность использования больше. «…Нейронный переводчик обрабатывает целые предложения, а статистический – делит его на n-граммы, при этом нейронный переводчик не может рассчитать модель

[6] NLTK (Natural language Toolkit) – пакет библиотек и программ для символьной и статистической обработки естественного языка. Разработан в Пенсильванском университете в 2001 году. Обновлен до версии 3.5 (12 апреля 2020).

языка, тогда как статистический может.» [Мифтахова 2019:500]. Таким образом, два подхода компенсируют недостатки друг друга.

Google и Microsoft перешли на нейросети в 2016 году, но скорее всего они также используют гибридные системы, хотя официального сообщения на сегодняшний момент нет. Переводы в Facebook также осуществляются с помощью нейронной сети.

Выводы по главе 1

Обработка естественного языка предполагает создание систем, обрабатывающих язык с целью выполнения определенных задач. На сегодняшний день ОЕЯ используется во многих сферах и решает множество задач, что является важной частью развития искусственных интеллектов. К основным задачам, непосредственно относящихся к лингвистике или затрагивающих ее, относятся: машинные перевод, распознавание речи (то, что выполняют голосовые помощники Cortana, Siri, Алиса и др.), анализ эмоциональной окраски высказываний, сортировка писем, извлечение и аннотирование информации, проверка орфографии и многое другое.

Машинный перевод представляет собой автоматизированный перевод письменных или устных текстов с одного языка на другой. Данный процесс непосредственно связан с естественными языками, что выявляет важность знаний в области лингвистики. В истории машинного перевода выделяются три главных подхода: машинный перевод на основе лингвистических правил, статистический и нейронный, но также можно выделить гибридные системы, комбинирующие представленные подходы.

Каждый подход является комплексной системой, базирующейся на определенных правилах и последовательностях этапов.

Первый подход – машинный перевод на базе лингвистических правил – стал базой для развития дальнейшего автоматизированного перевода. Несмотря на качественный и адекватный перевод, наблюдались недостатки, в других аспектах. Т.к. система базировалась на списке лингвистических правил перевода, согласования падежей и остального, требовались постоянные поправки и с увеличение объема словарей увеличивалась и скорость работы. Самым большим недостатком является то, что один словарь используется только на одно направление, т.е. для каждого языка требуется совершенно новая система.

Второй подход – статистический машинный перевод – базируется на n-граммной модели, т.е. высчитывает вероятность перевода по n-граммной

модели (биграммной – два слова, триграммной – три слова), основываясь на ранее сделанных людьми переводах, которые хранятся в параллельных корпусах. Наличие корпусов позволяет делать перевод с/на любой язык не требуя построения совершенно новой системы, что требовалось в RBMT. Несмотря на явное преимущество над первым подходом, статистический также имеет недостатки: требуется двуязычный корпус, качество перевода зависит от качества и объема корпусов, при малом объеме корпуса адекватность перевода снижается.

Третий подход – нейронный машинный перевод – совершенно новая система, базирующаяся на нейронной сети. В данном подходе наблюдается skip-grams, что отличает его от раннее представленных подходов. Нейросети анализируют слова даже если они находятся в разных частях предложения, что позволяет правильно определить контекст. Нейронный машинный перевод, как и статистический обучается, и т.к. он был представлен относительно недавно (2016) сейчас нейронный машинный перевод на стадии обучения, и, поэтому многие компании, предлагающие продукты машинного перевода, предпочитают использовать гибридные формы машинного перевода, а именно смешивать статистический и нейронный. К примеру, в Яндекс переводчике перевод выполняется статистической и нейронной моделью, и затем алгоритм на основе метода машинного обучения CatBoost сравнивает результаты и предлагает лучший вариант перевода.

Глава 2. Этапы обработки текстов в машинном переводе

2.1 Классическая поэтапная обработка текстов

Текст – одна из самых распространенных форм последовательностей, данных – символов и слов. Текст часто обрабатывается на уровне слов. Под обработкой текстов понимается анализ, преобразование, поиск, порождение текстовой информации. «При работе с отдельными словами как с элементами текста часто приходится определять синонимы, аббревиатуры, акронимы и правильность написания» [Риз 2016:23-24]. Модели для обработки последовательностей могут на основе текста формировать понимание естественного языка в простейшей форме, достаточных для таких применений, как классификация документов, анализ эмоциональной окраски, идентификация автора и даже получение ответов на вопросы в ограниченном контексте. Но следует помнить о том, что ни одна из этих моделей в действительности не понимает текст в человеческом смысле; они лишь отражают статистическую структуру письменного языка с целью решения многих простых задач обработки текста [Шолле 2018:211].

Обработка текстов, независимо от того, на каком он естественном языке, проходит одинаковые стадии. Специфичные черты, зависящие от естественного языка могут наблюдаться только в обработке сокращения слов и знаков препинания. Во всех современных системах обработки текстов, начиная от простейшего поиска вхождения слов и заканчивая машинным переводом, предусмотрено несколько этапов. Классическая поэтапная обработка проиллюстрирована на рисунке № 3 [28].

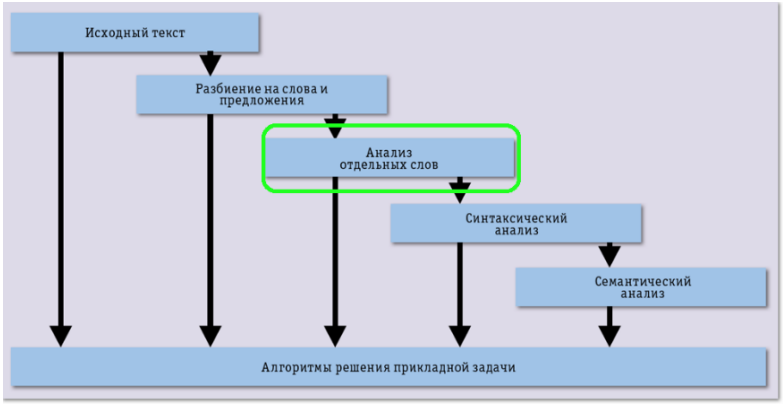


рис.4

На вход системы поступает исходный текст, представляющий собой последовательность слов и символов. На первом этапе – лексического анализа – происходит распознавание единицы текста. На входе система получает текст и разбивает ее на лексические единицы. Одним из фундаментальных процессов лексического анализа является токенизация – разбиение текста на предложения и/или отдельные слова. Лексические единицы, получаемые при токенизации, принято называть токенами, они могут совпадать со словоформой, с морфемой или даже со словосочетаниями.

Следующий этап представляет собой морфологический анализ – анализ и отдельных слов, распознавание элементов морфологической структуры слова (корень, основа, аффиксы, окончания) и определение морфологических параметров слова и основной словоформы. Морфологический синтез – генерация нужной словоформы слова (или всей ее парадигмы) по основе слова и морфологическим характеристикам (морфологический синтез).

После анализа и обработки слов начинается разбор отдельных предложений – синтаксический анализ – позволяющий определить взаимосвязи между отдельными словами и частями предложения. Система распознает предложения благодаря символам форматирования текста (знаки пунктуации, пробелы). Сложности возникают если в тексте отсутствуют знаки, выполняющие разделительную функцию (точки, восклицательные знаки, вопросительные знаки) или же если они используются не в конце, а в середине предложения. Таким образом, предложения в программировании могут не быть грамматически организованными и могут не обладать смысловой законченностью.

Семантический анализ текста базируется на результатах синтаксического анализа, получая на входе уже не набор слов, разбитых на предложения, а набор деревьев, отражающих синтаксическую структуру каждого предложения. Для выполнения алгоритмов семантического уровня требуются словари-тезаурусы, семантические словари, онтологии [Brinton, 2000: 149].

Все этапы взаимосвязаны. Каждый этап также имеет свои специфичные стадии обработки. Например, для семантического анализа текста не обязательно рассматривать все его слова, учитывая наличие стоп-слов[7], – смысл предложения определяется по ключевым словам и наличию связей между ними.

Результаты анализа текста представляют собой как концептуальную сеть, способной формально описать смысл, который содержится в исходном тексте, так и семантическую сеть, представляющую собой синтаксическую структуру (дерево) каждого предложения.

[7] Стоп-слова – слова, не несущие контекстуального смысла, например, “в общем”, “должен”, “I”, “because” и.т.д. Они удаляются с целью ускорения работы системы. В Python каталоге есть список стоп – слов для 16 языков.

2.2 Регулярные выражения.

2.2.1 Основные шаблоны и функции

Регулярные выражения (Regex) – последовательность символов, определяющая текстовый шаблон, соответствующий данному тексту. Регулярные выражения используются практически во всех языках программирования для описания языков с использованием алгебраических конструкций. В Python они реализованы в стандартном модуле ***re***. Широко используется в NLP, в приложениях, требующих проверки ввода текста и почти во всех проектах в области анализа данных, которые включают в себя интеллектуальную обработку текста. Модуль re позволяет построить практически любой текстовый шаблон, который понадобится при работе, связанным с поиском текста. Шаблон представляет собой специальный язык, который используется для представления общих цифр или символов, текста, извлечение текстов, соответствующих шаблону.

Базовые специальные символы, которые используются в шаблонах:

/	начальный/конечный символ-разделитель
\	Символ экранирования или начала метасимвола
|	Означаете ИЛИ
^	Начало строки
$	Конец строки
.	Любой символ, кроме перевода строки
?	НОЛЬ или ОДИН раз
*	НОЛЬ или МНОГО раз
+	ОДИН или МНОГО раз

Импорт модуля re и компилирование[8] шаблона регулярного выражения.

[8] Компилировать - проводить трансляцию машинной программы с предметно-ориентированного языка на машинно-ориентированный язык.

```
>>> import re
>>> regex= re.compile ('\s+')
```

Некоторые другие обычно используемые шаблоны:

Основной синтаксис:

\.	Просто точка `.`, обратный слеш `\` убирает магию всех специальных символов.
\d	Одна цифра
\D	Один символ кроме цифры
\w	Один буквенный символ, включая цифры
\W	Один символ кроме буквы и цифры
\s	Один пробельный (включая таб и перенос строки)
\S	Один не пробельный символ
\b	Границы слова
\n	Новая строка
\t	Табуляция

Модификаторы:

ab|cd	Соответствует ab или de.
[ab-d]	Один символ: a, b, c, d
[^ab-d]	Любой символ, кроме: a, b, c, d
()	Извлечение элементов в скобках
(a(bc))	Извлечение элементов в скобках второго уровня

Повторы:

[ab]{2}	2 непрерывных появления a или b
[ab]{2,5}	от 2 до 5 непрерывных появления a или b
[ab]{2,}	2 и больше непрерывных появления a или b

Основные функции модуля ***re*** для работы с регулярными выражениями:

1) re.search (pattern, string). Найти в строке первую строчку, подходящую под шаблон:

```
>>> import re
>>> regex= re.compile ('\s+')
>>> match=re.search (r'\d\d\D\d\d', r' Телефон 987-12-13')
>>> print (match [0] if match else 'not found')
87-12
>>> match=re.search (r'\d\d\D\d\d', r' Телефон 9871213')
>>> print (match [0] if match else 'not found')
not found
>>>
```

2) re.fullmatch (pattern, string). Проверка – подходит ли строка под шаблон:

```
>>> import re
>>> match = re.fullmatch (r'\d\d\D\d\d', r'55-55')
>>> print ('YES' if match else 'NO')
YES
>>> match = re.fullmatch (r'\d\d\D\d\d', r'T. 55-55')
>>> print ('YES' if match else 'NO')
NO
```

3) re.split (patter, string, maxsplit=0). Аналог str.split(), только разделение по подстрокам, подходящим под шаблон:

```
>>> print (re.split(r'\W+', 'Что такое регулярные выражения?'))
['Что', 'такое', 'регулярные', 'выражения', '']
```

4) re.findall (pattern, string). Найти в строке все непересекающиеся шаблоны, например, только цифры:

```
>>> print (re.findall (r'\d\d\.\d\d\.\d{4}', r'Он написал 02.02
.2020, а не 01.01.2020'))
['02.02.2020', '01.01.2020']
>>> text= """ 456 комнат  213 солдат 2 кг"""
>>> renum= re.compile ('\d+')
>>> renum.findall (text)
['456', '213', '2']

>>> text1= '17 мая 2020'
>>> print (re.findall ('\d+', text1))
['17', '2020']
```

Все, кроме цифр:

```
>>> print (re.findall ('\D+', text1))
[' мая ']
>>>
```

5) re.finditer. Итератор[9] всем непересекающимся шаблонам в строке. (выдаются match- объекты)

```
>>> for m in re.finditer (r'\d\d\.\d\d\.\d{4}', r'Он написал 02
.02.2020, а не 01.01.2020'): print ('Дата', m[0], 'начинается с
позиции', m.start())

Дата 02.02.2020 начинается с позиции 11
Дата 01.01.2020 начинается с позиции 28
```

\d{4} – последовательность из 4 цифр

6) re.sub (pattern, repl, string, count = 0). Замена в строке всех непересекающихся шаблонов на repl:

```
>>> print (text)
 456     комнат
213      солдат
2        кг

>>> re=re.compile('\s+')
>>> print (re.sub(' ', text))
 456 комнат 213 солдат 2 кг
```

Каждой из перечисленных выше функций можно дать дополнительный параметр ***flags*** - константы[10]

re.ASCII ускоряет работу, если все соответствия лежат внутри ASCII:

```
>>> print (re.findall (r'\w+', 'Welcome to Россия!'))
['Welcome', 'to', 'Россия']
>>> print (re.findall (r'\w+', 'Welcome to Россия!', flags=re.ASCII))
['Welcome', 'to']
```

re.IGNORECASE. Функция – не различать заглавные и маленькие буквы. Работает медленнее, но иногда удобно:

```
>>> print (re.findall (r' [аеиоуэюя]+', 'АААААА аааа ппппп
яЯяя ооооОО'))
[' аааа', ' ооо']
>>> print (re.findall (r'[аеиоуэюя]+', 'АААААА аааа ппппп Я
яяя ооооОО', flags=re.IGNORECASE))
['АААААА', 'аааа', 'яЯяя', 'ооооОО']
>>>
```

[9] Итератор – объект, позволяющий перебирать элементы контейнера, переходя от одного элемента к другому.

[10] Константа - способ адресации данных, изменение которых не предполагается или даже запрещается рассматриваемой программой. Использование именованных констант повышает надежность и безошибочность программ.

re.MULTILINE. Символ $ выполняет поиск в конце любой строки текста (не только конце текста) и символ ^ выполняет поиск в начале любой строки текста (не только в начале текста).

```
>>> text = r"""
Кекс
с вишней1
вишней2
"""
>>> print(re.findall(r'виш\w+', text, flags=r
e.MULTILINE))
['вишней1', 'вишней2']
>>> print(re.findall(r'^виш\w+', text, flags=
re.MULTILINE))
['вишней2']
>>>
>>> text = r"""
Кекс с черешней и вишней """
>>> print (re.findall (r'\w+\шней', text, flags = re.
MULTILINE))
['черешней', 'вишней']
>>>
```

re.DOTALL. По умолчанию символ ***\n*** конца строки не подходит под точку. С этим флагом точка – любой символ.

```
>>> text = r"""
Кекс
с вишней1
вишней2
"""
>>> print(re.findall(r'Кекс.с', text))
[]
>>> print(re.findall(r'Кекс.с', text, flags=r
e.DOTALL))
['Кекс\nс']
```

Стемминг создается на базе регулярных выражений.

Regex Stemmer:

```
>>> from nltk.stem import RegexpStemmer
>>> st= RegexpStemmer ('ing$|s$|ed$|able$', min=4)
>>> st.stem ('caring')
'car'
>>> st.stem ('entered')
'enter'
>>> st.stem ('cats')
'cat'
>>> st.stem ('comfortable')
'comfort'
>>>
```

2.3.2 Группы регулярных выражений

Группы регулярных выражений позволяют извлекать нужные объекты соответствия как отдельные элементы. Скобки (...) в шаблоне регулярного выражения формируют группы, например, группа чисел ([0-9]).

Пример извлечения цифр, слов и аббревиатур как отдельные элементы:

```
>>> text2= """ 1945 ООН Организация объединенных нация
1948 ВОЗ Всемирная организация здравоохранения
1946 МВФ Международный валютный фонд """
>>> re.findall ('[0-9]+', text2)
['1945', '1948', '1946']
>>> re.findall ('[А-ЯЁ]{3}', text2)
['ООН', 'ВОЗ', 'МВФ']
>>> re.findall('[а-яА-ЯёЁ]{4,}', text2)
['Организация', 'объединенных', 'нация', 'Всемирная', 'организация',
'здравоохранения', 'Международный', 'валютный', 'фонд']
```

Или:

```
>>> course_pattern = '([0-9]+)\s*([А-ЯЁ]{3})\s*([а-яА-ЯёЁ]{4,})'
>>> re.findall(course_pattern, text2)
[('1945', 'ООН', 'Организация'), ('1948', 'ВОЗ', 'Всемирная'), ('1946
', 'МВФ', 'Международный')]
>>>
```

Match-объекты

Если функции ***re.search, re.fullmatch*** не находят соответствие шаблону в строке, то они возвращают ***None,*** иначе возвращается ***match-объект***, что содержит в много полезной информации о соответствии шаблону. Например, ***match.start()*** и ***match.end ()*** - индексы в исходной строке, начиная с которого идёт найденная подстрока и который следует сразу за найденной подстрокой соответственно.

```
>>> import re
>>> pattern = r'\s*([А-Яа-яЁё]+)(\d+)\s*'
>>> string = r'  Опять45  '
>>> match = re.search(pattern, string)
>>> print(f'Найдена подстрока >{match[0]}< с позиции {match.start
(0)} до {match.end(0)}')
Найдена подстрока >  Опять45  < с позиции 0 до 11
>>> print(f'Группа букв >{match[1]}< с позиции {match.start(1)} д
о {match.end(1)}')
Группа букв >Опять< с позиции 2 до 7
>>> print(f'Группа цифр >{match[2]}< с позиции {match.start(2)} д
о {match.end(2)}')
Группа цифр >45< с позиции 7 до 9
```

Скобки (?:...), например, (?:REGEXP) в шаблоне регулярных выражений являются эквивалентным шаблону REGEXP. Но ?: позволяет применять

квантификаторы, показывая сколько раз должна повториться группа. Более того (?:...) позволяют локализовать часть шаблона, внутри которого происходит перечисление. К примеру, шаблон (?:\+7|8)(?:-\d{2,3}){4} применяется к тексту +7-926-123-12-12, 8-926-123-12-12. В (?:\+7|8) знак “ | ” означает “ИЛИ”.

Группирующие скобки могут находиться внутри других группирующих скобок. Их нумерация производится в соответствии с номером появления открывающей скобки с шаблоне:

```
>>> import re
>>> pattern = r'((\d)(\d))((\d)(\d))'
>>> string = r'123456789'
>>> match = re.search(pattern, string)
>>> print(f'Найдена подстрока >{match[0]}< с позиции {match.start
(0)} до {match.end(0)}')
Найдена подстрока >1234< с позиции 0 до 4
>>> for i in range(1, 7):
        print(f'Группа №{i} >{match[i]}< с позиции {match.start(i
)} до {match.end(i)}')

Группа №1 >12< с позиции 0 до 2
Группа №2 >1< с позиции 0 до 1
Группа №3 >2< с позиции 1 до 2
Группа №4 >34< с позиции 2 до 4
Группа №5 >3< с позиции 2 до 3
Группа №6 >4< с позиции 3 до 4
```

Использование групп добавляет замене очень удобную возможность. Например, перевод формата ММ/ДД/ГГГГ в формат ДД.ММ.ГГГГ,

```
>>> import re
>>> text = "We arrive on 01/01/2020. You are welcome after 04/01/2020."
>>> text
'We arrive on 01/01/2020. You are welcome after 04/01/2020.'
>>> print(re.sub(r'(\d\d)/(\d\d)/(\d{4})', r'\2.\1.\3', text))
We arrive on 01.01.2020. You are welcome after 01.04.2020.
>>>
```

Если групп больше 9, то используется конструкция вида \g<12>.

Сам стемминг создается на основе регулярных выражений.

2.3. Нормализация текста. Стемминг и лемматизация

Нормализация текста – приведение всех слов текста к словарной форме: к именительному падежу, единственному числу (если таковое есть) или инфинитиву для глаголов.

Нормализация нужна, например, для быстрого поиска слова в словарях, синтаксического и семантического разбора текста.

Особенность нормализации заключается в том, что в нем задействована вся последовательность лингвистической обработки текста:

1) разбиение текста на предложения;

2) определение части речи всех слов текста (так называемый PoS-tagging - Part-of-Speech tagging);

3) нахождение морфологических характеристик всех слов;

4) снятие омонимии.

Основой методов нормализации служит морфологический и анализ части речи каждого слова.

Морфологический разбор состоит из:

1) морфологического анализа — получение леммы или основы (псевдоосновы) заданного токена, а при необходимости, морфологических параметров;

2) морфологического синтеза — это генерация нужной словоформы слова или всей его парадигмы по нормальной форме (или основе) и морфологическим характеристикам. [Большакова 2017: 33]

Предполагается, что каждый токен обладает начальной (базовой) формой. В лингвистике традиционно считается, что одно и то же слово может выступать в различных грамматических формах [Шайкевич 2005: 157].

Существует три основных подхода к проведению морфологического анализа:

1) «точная морфология» – предполагает построение одного большого словаря с характеристикой каждого слова. Например, для русского языка применяется словарь Зализняка. [Зализняк 1987: 800]. Словарь Зализняка

содержит основные словоформы слов русского языка, для каждой из которых указан определенный код, насчитывается более 8 млн слов. Анализ на основе словаря:

рис.5

Основным недостатком данного подхода является то, что не все слова, поступающие на вход, могут входить в словарь всех словоформ, в частности это имена собственные, неологизмы, которые еще не внесены в словарь и.т.д. В таком случае, используется следующий подход.

2) «неточная морфология» – характеризуется некоторой системой правил, согласно которому по заданному слову определяются его морфологические характеристики. Система правил опирается на наличие или отсутствие каких-либо частей и выдает одно или несколько предположений о морфологических параметрах. Задачу построения системы правил решается с помощью самообучающейся системы (рис. 6). В процессе анализа программа сопоставляет аффиксы во входном тексте, анализ начинается с последнего символа слова.

рис.6

3) вероятностный подход, основан на сочетаемости слов с конкретными морфологическими характеристиками; он широко применяется при обработке

языков со строго фиксированным порядком слов в предложении, что означает, что он практически неприменим при обработке текстов на русском языке [28].

Как система разбирает слово? Допустим, целью является сбор информации о представленном слове, возьмем, к примеру, *disconnected,* – определение эмоциональной окраски, значение слова и.т.д.

На основе лингвистических знаний, определяем основу и аффиксы данного слова.

Приставка *dis-* в английском языке означает отрицание; с помощью окончания *-ed* образовываются правильные глаголы в Past Simple, также *-ed* образовывает причастие прошедшего времени во временах группы Perfect, стоит также отметить, что многие глаголы с окончанием *-ed* приобрели функцию прилагательных, образовываются, так называемые, отглагольные прилагательные. Таким образов, выясняется, что слово *disconnected* может выполнять функцию глагола, причастия и отглагольного прилагательного, в зависимости от положения в предложении. Приняв во внимание большое многообразие приставок и постфиксов английского языка, понадобится очень трудоемкий анализ для понимания всевозможных комбинаций и их значений. (В приложении 1 представлена таблица с некоторыми примерами префиксов, суффиксов и корней в английском языке, с их значениями и примером).

Морфемный разбор слова важен для понимания и обработки естественного языка. Обычно тексты содержат слова в разных грамматических формах, к тому же могут содержать и однокоренные слова. В большинстве случаев система заменяет все прописные буквы строчными. Однако есть исключения, особенно если прописные буквы встречаются в середине предложения: GeneralMotors, US – us [Мифтахова 2018:711]. К широко

применяемым на морфологическом уровне алгоритмам относятся стемминг и лемматизация.

Задачей стемминга и лемматизации является приведение всех встречающихся словоформ к одной, основной словарной форме, к примеру:

cat, cats, cat's, cats' => cat

Стемминг – упрощенная форма морфологического анализа, процесс выведения базовой формы слов путем измельчения их концов [29]. Например, в слове "caring" стеммер "отрезает" -ing и приводит к форме "car", что приводит к неверному значение (вместо заботы – автомобиль). Стемминг – грубый процесс приведения к базовой форме слова.

Стеммеры, разрабатываемые с 1950-х классифицируется на алгоритмическую и словарную [Hull 1996: 71]. Алгоритмические стеммеры основываются на файловых данных, а словарные на словарях основ слов. В процессе алгоритмического морфологического анализа программа сопоставляет суффиксы и окончания слов во входном тексте и на выходе получается список основ слов входного предложения, анализ слова начинается с последнего символа и по правилам того или иного языка срезается суффиксы и окончания. Но программа может привести к основе по которой можно отождествить два абсолютно семантически разных слова, что имеет название избыточный стемминг. К примеру основа ***ent*** слов ***enter*** и ***entity***. Но это не единственный недостаток алгоритмического стемминга: иногда система не может отождествить два семантически одинаковых слова по одной основе, например, по основе ***wolv*** невозможно отождествить формы единственного числа ***wolf*** и множественного ***wolves.*** Словарный стеммер в процессе анализа сопоставляет основу слова во входном тексте и в соответствующем выбранном словаре и анализ слова осуществляется с первого символа. Словарный стеммер обеспечивает высокую точность, но несмотря на это алгоритмические стеммеры используются чаще. Одной из причин является свойство изменчивости языков, т.е. изменения структуры на морфологическом

уровне происходят медленнее, чем на лексическом т.к. количество аффиксов в конкретном языке ограничено, а в связи со стремительным развитием технологий наблюдается тенденция появления новых слов, в частности существительных для обозначения тех или иных понятий. Вполне логично также отметить большой размер словаря (с постепенными обновлениями), что напрямую влияет на скорость работы системы.

Лемматизация – способ извлечения базовой формы слова, обычно, путем удаления флективных окончаний с помощью словарного и морфологического анализа. После лемматизации базовая форма любого слова называется леммой [29]. Предполагается, что лемматизация определив контекст правильно определит основу “care” в слове “caring”.

Лемматизация также отождествляет основу слова, но в отличии от стемминга проводит это с учетом частей речи слов. Предполагается, что лемматизация отождествит глагольные формы ***singing***, ***sings*** с основой ***sing***, а именные ***singer***, ***singers*** с леммой ***singer.***

Словари лемм широко используются в корпусной лингвистике. Распределение по частям речи является существенным параметром при проведении автоматической классификации и категоризации текстов [Santini 2006: 38].

Лемматизация учитывает контекст, стемминг просто отрезает аффиксы. Стемминг проще внедрить, и он работает быстрее. Поэтому стемминг является предварительным анализом лемматизации.

Далее проведем практическую работу – стемминг и лемматизацию отдельных слов, предложения и текста. Реализация стемминга и лемматизации невозможна без пакетов (библиотеки), так называются лексические базы данных. Их можно скачать вводом определенной команды.

Стемминг:

Для стемминга английских текстов широко употребляется алгоритм Портера (Porter Stemmer опубликован в 1980 году Мартином Портером) и Ланкастера (Lancaster Stemmer).

Стемминг портера реализован с помощью класса Regex, так как регулярные выражения позволяют автоматизировать сложную обработку текста [Фаленов 2009:254]. По регулярным выражениям проверяется принадлежность аффиксов слова к той или иной группе.

Стемминг отдельных слов при помощи Porter Stemmer и Lancaster Stemmer. Сначала загружаем библиотеку Porter Stemmer, и Lancaster Stemmer, затем определяем основу слов функцией *stem ():*

Стемминг слов в Porter Stemmer:

```
>>> import nltk
>>> from nltk.stem import PorterStemmer
>>> ps=PorterStemmer()
>>> print (ps.stem ("cats"))
cat
>>> print (ps.stem("teeth"))
teeth
>>> print (ps.stem("feet"))
feet
>>> print (ps.stem("wolves"))
wolv
>>> print (ps.stem("entered"))
enter
>>> print (ps.stem("caring"))
care
>>> print (ps.stem("carefully"))
care
>>> print (ps.stem ('replayed'))
replay
>>> print (ps.stem ('disconnected'))
disconnect
>>> print (ps.stem ('disconnect'))
disconnect
>>> print (ps.stem ('sings'))
sing
>>> print (ps.stem ('singers'))
singer
>>>
```

Стемминг слов в Lancaster Stemmer:

```
>>> import nltk
>>> from nltk.stem import LancasterStemmer
>>> ls= LancasterStemmer ()
>>> print (ls.stem ('cats'))
cat
>>> print (ls.stem ('feet'))
feet
>>> print (ls.stem ('wolves'))
wolv
>>> print (ls.stem ('singers'))
sing
>>> print (ls.stem ('sings'))
sing
>>> print (ls.stem ('disconnected'))
disconnect
>>> print (ls.stem ('restart'))
restart
>>> print (ls.stem ('entered'))
ent
>>> print (ls.stem ('caring'))
car
>>> print (ls.stem ('carefully'))
car
>>> print (ls.stem ('entered'))
ent
>>> print (ls.stem ('replayed'))
replay
>>>
```

Porter Stemmer прекрасно справился со словами, где грамматическая форма была представлена при помощи аффиксов без изменений в корне слова ***(cats- cat; carefully – care; entered - enter***). Что еще более интересно и не характерно для стемминга – в слове caring выделил основу ***care***, хотя предполагалась, что выделит ***car***, что можно заметить в Lancaster Stemmer. Возможно в Porter Stemmer подключена функция лемматизация. Стоит отметить, слово ***entered*** Lancaster Stemmer приводит к основе ***ent.*** Существует вероятность, что программа привела к основе по которой можно отождествить два абсолютно семантически разных слова. Проверим слова с возможным приведение к такой же основе.

```
>>> print (ls.stem ('entered'))
ent
>>> print (ls.stem ('enterprise'))
enterpr
>>> print (ls.stem ('entertain'))
entertain
>>> print (ls.stem ('entity'))
ent
```

Как и предполагалось, из списка представленных слов с возможным выявление базовой формы ***ent*** Lancaster Stemmer приводит слово ***entity*** к такой же нормальной форме. Слова ***enter*** и ***entity*** семантически не связаны друг с другом (enter – входить, entity- сущность, организация), таким образом, наблюдается избыточный стемминг. Это может привести к тому, что при переводе слово ***enter*** может перевестись со значение слова ***entity.***

Lancaster Stemmer привел слова ***singers*** и ***sings*** к одной основе ***sing.*** Но Porter Stemmer отождествил их глагольную форму с ***sing,*** а именную с ***singer***, что также дает ему преимущество.

Стемминг слов с апострофом

```
>>> ps.stem ("singer's
"singer'"
>>> ls.stem ("singer's
"singer's"
```

Porter Stemmer

Lancaster Stemmer

Оба стеммера не смогли определить приставку в словах ***disconnected, replayed, restart,*** а также слово ***wolves*** приводят к форме ***wolv***, определив ***-es***, как аффикс, по своему алгоритму все верно, но с лингвистической точки зрения основа ***wolv*** не отождествляет единственное число ***wolf*** со множественным ***wolves*** – подтверждается недостаточный стемминг. И как следствие далее используется лемматизация. Метод грубого срезания аффиксов эффективен в некоторых случаях, если, например, аффикс просто “вставлен” в слово, без каких-либо изменений в корне слова, но, и здесь, бывают ошибки ***(ent- Lancaster Stemmer).*** Стеммер относительно агрессивно срезает то, что он принимает за окончание, и при этом, не всегда приводит к правильной лексеме. Это объясняется тем, что стеммеры не содержат словарей лексем. Таким образом стеммер не эффективен с образованием словоформ с особыми правилами видоизменения, например, множественного числа некоторых слов:

Porter Stemmer

```
>>> print (ps.stem('knives'))
knive
>>> print (ps.stem('mice'))
mice
>>> print (ps.stem('data'))
data
>>> print (ps.stem('datum'))
datum
>>> print (ps.stem ('phenomena'))
phenomena
>>> print (ps.stem ('phenomenon'))
phenomenon
```

Lancaster Stemmer

```
>>> print (ls.stem ('knives'))
kniv
>>> print (ls.stem ('mice'))
mic
>>> print (ls.stem ('data'))
dat
>>> print (ls.stem ('datum'))
dat
>>> print (ls.stem ('phenomena'))
phenomen
>>> print (ls.stem ('phenomenon'))
phenomenon
```

Стемминг предложения:

Porter Stemmer:

```
>>> sent=['He', 'considered', 'himself', 'the', 'luckiest', '
person', 'for', 'about', 'seventeen', 'years', 'he', 'has', '
received', 'kindness', 'and', 'encouragement', 'from', 'her']
>>> [ps.stem (t) for t in sent]
['He', 'consid', 'himself', 'the', 'luckiest', 'person', 'for
', 'about', 'seventeen', 'year', 'he', 'ha', 'receiv', 'kind'
, 'and', 'encourag', 'from', 'her']
>>>
```

Lancaster Stemmer:

```
>>> sent = ['He', 'considered', 'himself', 'the', 'luckiest',
'person', 'for', 'about', 'seventeen', 'years', 'he', 'has', '
received', 'kindness', 'and', 'encouragement', 'from', 'her']
>>> [ls.stem (t) for t in sent]
['he', 'consid', 'himself', 'the', 'luckiest', 'person', 'for'
, 'about', 'seventeen', 'year', 'he', 'has', 'receiv', 'kind',
 'and', 'enco', 'from', 'her']
```

Porter Stemmer в слове ***“has”*** отрезал ***“–s”*** посчитав ее окончанием, чего не замечается в Lancaster, но правильно определил основы для “**years”, “kindness”, “received”**, **“considered”** как и Lancaster.

Оба стеммера не смогли определить суффикс превосходной формы прилагательных ***“-est”*** в слове ***“luckiest”,*** а также в слове ***“encoragement”*** Porter выдает ***“encourag”,*** а Lancaster ***“enco”*** – оба не смогли отождествить со словом ***courage*** или ***encourage.***

Выше предложение было разделено на слова вручную, для облегчения работы и сокращения времени можно использовать команды *word_tokenize ()*

(при делении текста на слова) и *sent_tokenize ()* (при делении текста на предложения):

```
>>> from nltk.tokenize import word_tokenize
>>> sent= 'Stemming and Lemmatization are Text Normalizati
on (or sometimes called Word Normalization) techniques in
the field of Natural Language Processing that are used to
prepare text, words, and documents for further processing.
'
>>> print (word_tokenize(sent))
['Stemming', 'and', 'Lemmatization', 'are', 'Text', 'Norma
lization', '(', 'or', 'sometimes', 'called', 'Word', 'Norm
alization', ')', 'techniques', 'in', 'the', 'field', 'of',
 'Natural', 'Language', 'Processing', 'that', 'are', 'used
', 'to', 'prepare', 'text', ',', 'words', ',', 'and', 'doc
uments', 'for', 'further', 'processing', '.']
>>>
```

Далее стемминг произведения. В качестве примера было взято произведение Джейн Остин «Эмма», которое содержится в корпусе данных Python. Так как произведение представляет собой объемную последовательность слов и символов, для примера проанализируем только первые 50 слов и символов.

Porter Stemmer:

```
>>> from nltk.stem import PorterStemmer
>>> ps=PorterStemmer()
>>> from nltk.corpus import gutenberg
>>> text=gutenberg.words('austen-emma.txt')
>>> text1=text[0:50]
>>> text1
['[', 'Emma', 'by', 'Jane', 'Austen', '1816', ']', 'VOLUME'
, 'I', 'CHAPTER', 'I', 'Emma', 'Woodhouse', ',', 'handsome'
, ',', 'clever', ',', 'and', 'rich', ',', 'with', 'a', 'com
fortable', 'home', 'and', 'happy', 'disposition', ',', 'see
med', 'to', 'unite', 'some', 'of', 'the', 'best', 'blessing
s', 'of', 'existence', ';', 'and', 'had', 'lived', 'nearly'
, 'twenty', '-', 'one', 'years', 'in', 'the']
>>> [ps.stem (t) for t in text1]
['[', 'emma', 'by', 'jane', 'austen', '1816', ']', 'volum',
'I', 'chapter', 'I', 'emma', 'woodhous', ',', 'handsom', ',
', 'clever', ',', 'and', 'rich', ',', 'with', 'a', 'comfort
', 'home', 'and', 'happi', 'disposit', ',', 'seem', 'to', '
unit', 'some', 'of', 'the', 'best', 'bless', 'of', 'exist',
';', 'and', 'had', 'live', 'nearli', 'twenti', '-', 'one',
'year', 'in', 'the']
>>>
```

Lancaster Stemmer:

```
>>> [ls.stem (t) for t in text1]
['[', 'emm', 'by', 'jan', 'aust', '1816', ']', 'volum', 'i', '
chapt', 'i', 'emm', 'woodh', ',', 'handsom', ',', 'clev', ',',
 'and', 'rich', ',', 'with', 'a', 'comfort', 'hom', 'and', 'ha
ppy', 'disposit', ',', 'seem', 'to', 'unit', 'som', 'of', 'the
', 'best', 'bless', 'of', 'ex', ';', 'and', 'had', 'liv', 'nea
r', 'twenty', '-', 'on', 'year', 'in', 'the']
>>>
```

В общей сложности оба стеммера в основном определяют основу слова у одних и тех же слов с разным “отрезанием” концов. Lancaster Stemmer имеет тенденцию приводить к наиболее короткой словоформе. Оба стеммера отлично справляются с отрезанием таких аффиксов как ***-ed, -er, -s, -able, -ion***, и ***-ence.***

Недостатки определения основы корня можно попытаться обусловить принципом работы самих стеммеров. Для более глубокого понимание далее представлен принцип работы Porter Stemmer.

Алгоритм Портера состоит из 5 этапов. На каждом этапе отсекается словообразующий суффикс, и оставшаяся часть проверяется на соответствие правилам. Если полученное слово удовлетворяет правилам, происходит переход на следующий шаг. Если нет — алгоритм выбирает другой суффикс для отсечения. [Porter 1980: 130] Таким образом стеммер Портера это набор простых правил. Рассмотрим их как перечень этапов:

1a:

sses	→	ss	caresses	→	caresss
ies	→	i	ponies	→	poni
ss	→	ss	caress	→	caress
s	→	ø	cats	→	cat

Когда находит строчку ***sses, ies, ss,*** отсекает все после указанного сочетания ***ss, i, ss*** соответственно, а ***s*** по указанным правилам всегда отсекается.

1b: *отсекаются все указанные окончания глаголов:*

(*verb*) ing	→	ø	walk***ing***	→	walk
			sing	→	***sing***

(*verb*) ed ⟶ ø plaster***ed*** ⟶ plaster

Но правило имеет уточнения: только у слов с гласными до ***-ing*** отсекается окончание ***–ing***. Т.к к примеру в слове sing нет других гласных кроме сочетании *ing.*

2: *для слов с изменениями в словоформе*

ational	⟶	ate	relational	⟶	relate
izer	⟶	ize	digitizer	⟶	digitize
ator	⟶	ate	operator	⟶	operate

3:

al	⟶	ø	revival	⟶	relative
able	⟶	ø	adjustable	⟶	adjust
ate	⟶	ø	activate	⟶	activ

Цель стемминга состоит в том, чтобы свести вместе различные формы слов, для этого используется метод грубого срезания окончаний.

Таким образом, можно прийти к выводу, что стеммеры имеют только список возможных вариантов окончаний и по определенным правилам их срезают. То, что стемминг не сопоставляет слова с “парадигмальной” формой, можно объяснить тем, что у стеммеров нет словарей лексем.

Стемминг русского языка. Для начала, рассмотрим какие стеммеры существуют для русского языка:

1) Уже известный нам **алгоритм Портера**. Основа должна содержать не менее одной гласной. Также состоит из 5 шагов:

1) отсечение максимально-формообразующего суффикса и окончания;

2) если слово оканчивается на букву «и» она отсекается;

3) отсечение словообразующего суффикса;

4) отсечение суффиксов превосходных форм;

5) если слово оканчивается на «нн», то одна из «н» отсекается, «ь» на конце слова также отсекает.

Наибольшим недостатком является избыточный стемминг т.к. стеммер выбирает для удаления наиболее длинную морфему.

2) **Stemka** –разработан Андреем Коваленко в 2002 году, основан на вероятностной модели: слова из обучающего текста разбираются анализатором на пары «последние две буквы основы» + «суффикс» и если такая пара уже существует в модели – её вес увеличивается. Далее все ранжируется по убыванию веса и отсекаются маловероятные модели. Результат — набор потенциальных окончаний с условиями на предшествующие символы — инвертируется для удобства сканирования словоформ «справа налево» и представляется в виде таблицы переходов конечного автомата [31].

Доступен в исходных текстах и может использоваться в свободной форме с условием ссылки на источник.

3) **MyStem** разработан Ильёй Сегаловичем в 1998. Сейчас является собственностью компании Яндекс. Также состоит из нескольких этапов: на первом этапе при помощи дерева суффиксов во входном слове определяются возможные границы между основой и суффиксом, после чего для каждой потенциальной основы начиная с самой длинной бинарным поиском по дереву основ проверяется её наличие в словаре либо нахождение наиболее близких к ней основ мерой близости является длина общего «хвоста». Если слово словарное — алгоритм заканчивает работу, в противном случае— переходит к следующему разбиению [32].

4) **Snowball Stemmer** также создан Мартином Портером. Принцип работы очень схож с алгоритмом Портера, – применяя последовательно ряд правил, отсекает окончания и суффиксы – т.к. Snowball Stemmer является видоизмененной версией Porter Stemmer.

Пример стемминга некоторых слов и предложения на русском языке, с библиотекой Snowball Stemmer

Стемминг некоторых слов:

```
>>> from nltk.stem.snowball import SnowballStemmer
>>> snow=SnowballStemmer("russian")
>>> snow.stem ("добрый")
'добр'
>>> snow.stem ("везти")
'везт'
>>> snow.stem ("рвать")
'рват'
>>> snow.stem ("учительница")
'учительниц'
>>> snow.stem ("читающий")
'чита'
>>> snow.stem ("засохший")
'засохш'
>>> snow.stem ("росший")
'росш'
>>>
```

Стемминг предложения:

```
>>> from nltk.stem.snowball import SnowballStemmer
>>> snow=SnowballStemmer("russian")
>>> from nltk.tokenize import word_tokenize
>>> sent= "Перед тем как определить основу слова, раз
делим предложение на токены"
>>> print (word_tokenize(sent))
['Перед', 'тем', 'как', 'определить', 'основу', 'слов
а', ',', 'разделим', 'предложение', 'на', 'токены']
>>> sent1=word tokenize(sent)
>>> [snow.stem (t) for t in sent1]
['перед', 'тем', 'как', 'определ', 'основ', 'слов', '
,', 'раздел', 'предложен', 'на', 'ток']
>>>
```

Как и предполагалось стеммер отлично справился с окончаниями ***“-и”, “-ый”, “-ь”, “-а”*** … но не определил суффиксы ***“-ш-”, “-и-”, “-тель-” …***

Лемматизация:

Перечень пакетов:

Один из самых ранних пакетов **WordNet Lemmatizer** создан Princeton и является общедоступным и наиболее употребляемым пакетом для лемматизации английских текстов.

Text Blob – это мощный, быстрый пакет для обработки текстовых данных. Он предоставляет простой API[11] для выполнения основных общих

[11] API - интерфейс прикладного программирования. Описание способов которыми одна компьютерная программа может взаимодействовать с другой программой.

задач: PoS-тегирование, извлечение именной группы, анализ эмоциональной окраски высказываний, перевод и многое другое. Быстрая лемматизация слов и предложений обеспечивается объектами Word и TextBlob. Интегрирует с WordNet.

spaCy Lemmatizer. spaCy не является API, т.е. не предоставляет программное обеспечение в качестве web-приложения, а является библиотекой с открытым исходным кодом, как и NLTK. Относительно новый пакет, создан и развит в 2015 году. Некоторые функции spaCy предусматривают статистические модели, что позволяет распознавать лингвистические характеристики слов, т.е. является ли слово глаголом или же существительным. Модели различаются в размере, скорости выполнения операций, памяти, точности, и данных, которые они включают. Модель зависит от входного текста. Вот некоторые из моделей: 1) Бинарный вес для PoS- тегов; 2) словарная статья[12]; 3) файлы данных, к примеру, правила лемматизации; 4) векторы слов; 5) опция конфигурации, например, настройки языка или конвейерная обработка, чтобы правильно загружать spaCy. spaCy Lemmatizer поставляется с предварительно созданными моделями, которые могут анализировать текст и выполнять различные функции, связанные с обработкой естественного языка.

CLiPS – универсальный пакет со многими полезными возможностями обработки естественного языка. Базируется на шаблоне **pattern,** который содержит объемный набор документаций и академических публикаций. Возможно, самым большим недостатком является то, что пакетом можно воспользоваться только при наличии лицензии 2.5+ Python.

Standford CoreNLP изначально реализован на Java, но несколько программных модулей адаптированы в Python. Инструменты предоставляют точные и оптимизированные методы теггирования, определения основы слова

[12] Словарная статья содержит лексический вход, стилистическую помету, грамматическую информацию, толкование, примеры употребления, зона идиоматики (устойчивые сочетания, фразеологизмы).

и анализа текста на различных языках. Также поддерживает аннотации и способен расширяться.

Gensim – специализированная библиотека. Высокоэффективна для семантического и тематического моделирования, а также в обработке текста. Предоставляет средства лемматизации на основе пакета pattern (интеллектуальный анализ web- данных, общие задачи).

Treetagger является PoS тегером для многих языков. Он также предоставляет возможности лемматизации.

Лемматизация на примере двух пакетов – WordNet Lemmatizer и TextBlob:

Лемматизация слов при помощи TextBlob:

```
>>> from textblob import TextBlob, Word
>>> word1="cats"
>>> w=Word(word1)
>>> w.lemmatize()
'cat'
>>> word2="sings"
>>> w1=Word(word2)
>>> w1.lemmatize()
'sings'
>>> word3="singer"
>>> w2=Word(word3)
>>> w2.lemmatize()
'singer'
```

Лемматизация отдельных слов при помощи WordNetLemmatizer:

```
>>> import nltk
>>> from nltk.stem import WordNetLemmatizer
>>> lem=WordNetLemmatizer()
>>> print (lem.lemmatize ("cats"))
cat
>>> print (lem.lemmatize ("teeth"))
teeth
>>> print (lem.lemmatize ("feet"))
foot
>>> print (lem.lemmatize ("wolves"))
wolf
>>> print (lem.lemmatize ("entered"))
entered
>>> print (lem.lemmatize ("caring"))
caring
>>> print (lem.lemmatize ("carefully"))
carefully
>>> print (lem.lemmatize ("replayed"))
replayed
>>> print (lem.lemmatize ("disconnected"))
disconnected
>>> print (lem.lemmatize ("disconnect"))
disconnect
>>> print (lem.lemmatize ("restart"))
restart
>>> print (lem.lemmatize ("knives"), ('mice'), ('data'), ('datum'))
knife mice data datum

 >>> print (lem.lemmatize("sings"))
 sings
 >>> print (lem.lemmatize ('singers'))
 singer
 >>>
 >>> print (lem.lemmatize('singing'))
 singing
 >>>
```

Команды в TextBlob для выполнения лемматизации слов являются трудоемкими и занимают дольше времени, чем в WordNetLemmatizer.

С одной стороны, WordNetLemmaztizer смог правильно определить базовую форму для слов: ***wolves (wolf), feet (foot), knives (knife).*** Но, с другой, имеются слова, у которых система не смогла или же неправильно определила лемму, с чем, в свое время, справился стемминг. (например, окончания у глаголов). Более того ***“singers”*** отождествил с именной формой ***“singer”,*** но глаголы ***“sings” “singing”*** оставил без изменений.

Одним из отличительных черт лемматизации является то, что в лемматизации используется PoS-теги. PoS-теги определяют к какой части речи относиться то или иное слово, и на их основе применяется ряд правил отдельно для существительных, глаголов и.т.д., таким образом и качество

определения улучшается. Если не ввести команду тегов, все слова по умолчанию будут просматриваться как существительные. Добавим PoS-теги вручную:

```
>>> print (lem.lemmatize ('singers', 'n'))
singer
>>> print (lem.lemmatize ('singing', 'v'))
sing
>>> print (lem.lemmatize ('sings', 'v'))
sing
```

или автоматическое определение PoS-тега:

```
>>> print (nltk.pos_tag(['feet']))
[('feet', 'NNS')]
```

Лемматизация предложения:

```
>>> from nltk.stem import WordNetLemmatizer
>>> lem=WordNetLemmatizer()
>>> sent= "He considers himself as the happiest person, for
about twenty years he has recieved kindness, encouragement,
caring and love from her wife"
>>> from nltk.tokenize import word_tokenize
>>> print (word_tokenize(sent))
['He', 'considers', 'himself', 'as', 'the', 'happiest', 'pe
rson', ',', 'for', 'about', 'twenty', 'years', 'he', 'has',
'recieved', 'kindness', ',', 'encouragement', ',', 'caring'
, 'and', 'love', 'from', 'her', 'wife']
>>> sent1=word_tokenize(sent)
>>> [lem.lemmatize (t) for t in sent1]
['He', 'considers', 'himself', 'a', 'the', 'happiest', 'per
son', ',', 'for', 'about', 'twenty', 'year', 'he', 'ha', 'r
ecieved', 'kindness', ',', 'encouragement', ',', 'caring',
'and', 'love', 'from', 'her', 'wife']
>>>
```

При лемматизации данного предложения результат не является столь удовлетворительным: в целом изменений не наблюдается, кроме срезания ***–s***, что лемматизатор принял за окончание ***(as, has).***

Можно сделать автоматическое определение PoS-тегов:

```
>>> print (nltk.pos_tag(nltk.word_tokenize(sent)))
[('He', 'PRP'), ('considered', 'VBD'), ('himself', '
PRP'), ('as', 'IN'), ('the', 'DT'), ('happiest', 'JJ
S'), ('person', 'NN'), (',', ','), ('for', 'IN'), ('
about', 'IN'), ('twenty', 'NN'), ('years', 'NNS'), (
'he', 'PRP'), ('has', 'VBZ'), ('received', 'VBN'), (
'kindness', 'NN'), (',', ','), ('encouragement', 'NN
'), (',', ','), ('caring', 'VBG'), ('and', 'CC'), ('
love', 'NN'), ('from', 'IN'), ('her', 'PRP$'), ('wif
e', 'NN')]
```

Лемматизация текста, на примере произведения Джейна Остин «Эмма». Также взяты первые 50 слов и символов:

```
>>> from nltk.corpus import gutenberg
>>> text=gutenberg.words ('austen-emma.txt')
>>> text1=text[0:50]
>>> text1
['[', 'Emma', 'by', 'Jane', 'Austen', '1816', ']', 'VOLU
ME', 'I', 'CHAPTER', 'I', 'Emma', 'Woodhouse', ',', 'han
dsome', ',', 'clever', ',', 'and', 'rich', ',', 'with',
'a', 'comfortable', 'home', 'and', 'happy', 'disposition
', ',', 'seemed', 'to', 'unite', 'some', 'of', 'the', 'b
est', 'blessings', 'of', 'existence', ';', 'and', 'had',
'lived', 'nearly', 'twenty', '-', 'one', 'years', 'in',
'the']
>>> from nltk.stem import WordNetLemmatizer
>>> lem=WordNetLemmatizer()
>>> [lem.lemmatize (t) for t in text1]
['[', 'Emma', 'by', 'Jane', 'Austen', '1816', ']', 'VOLU
ME', 'I', 'CHAPTER', 'I', 'Emma', 'Woodhouse', ',', 'han
dsome', ',', 'clever', ',', 'and', 'rich', ',', 'with',
'a', 'comfortable', 'home', 'and', 'happy', 'disposition
', ',', 'seemed', 'to', 'unite', 'some', 'of', 'the', 'b
est', 'blessing', 'of', 'existence', ';', 'and', 'had',
'lived', 'nearly', 'twenty', '-', 'one', 'year', 'in', '
the']
>>>
```

Результат лемматизации текста при помощи пакета TextBlob:

```
>>> print (lem)
['[', 'Emma', 'by', 'Jane', 'Austen', '1816', ']', 'VOLUME', '
I', 'CHAPTER', 'I', 'Emma', 'Woodhouse', ',', 'handsome', ',',
 'clever', ',', 'and', 'rich', ',', 'with', 'a', 'comfortable'
, 'home', 'and', 'happy', 'disposition', ',', 'seemed', 'to',
'unite', 'some', 'of', 'the', 'best', 'blessing', 'of', 'exist
ence', ';', 'and', 'had', 'lived', 'nearly', 'twenty', '-', 'o
ne', 'year', 'in', 'the']
```

WordNetLemmatizer и TextBlob не смогли определить основу слова в тексте. Системы выдают одинаковый результат. Для повышение качества можно добавить PoS-теги:

```
>>> blob=TextBlob(austen4)
>>> blob.tags
[('[', 'NNS'), ('Emma', 'NNP'), ('by', 'IN'), ('Jane', 'NNP'),
 ('Austen', 'NNP'), ('1816', 'CD'), (']', 'NNP'), ('VOLUME', '
NNP'), ('I', 'PRP'), ('CHAPTER', 'VBP'), ('I', 'PRP')]
>>> blob=TextBlob(text)
>>> blob.tags
[('Natural', 'JJ'), ('language', 'NN'), ('processing', 'NN'),
('NLP', 'NNP'), ('deals', 'VBZ'), ('with', 'IN'), ('the', 'DT'
), ('application', 'NN'), ('of', 'IN'), ('computational', 'JJ'
), ('models', 'NNS'), ('to', 'TO'), ('text', 'VB'), ('or', 'CC
'), ('speech', 'VB'), ('data', 'NNS')]
>>>
```

Необходимо подчеркнуть, что на результат определения основы слова может, также, повлиять язык программирования, например, кроме нами рассмотренной Python, есть Java, Perl и другие.

Выше был рассмотрен принцип работы стемминга и лемматизации, а также сравнение библиотек стемминга и лемматизатора.

Далее сравнение стемминг (PorterStemmer) и лемматизацию (WordNetLemmatizator) с PoS-тегами для лемматизатора командой *compare*.

```
>>> compare_stemmer_and_lemmatizer (stem, lem, word= "are", pos= wordne
t.VERB)
Stemmer: are
Lemmatizer: be

>>> compare_stemmer_and_lemmatizer (stem, lem, word= "has", pos= wordne
t.VERB)
Stemmer: ha
Lemmatizer: have

>>> compare_stemmer_and_lemmatizer (stem, lem, word= "wolves", pos= wor
dnet.NOUN)
Stemmer: wolv
Lemmatizer: wolf

>>> compare_stemmer_and_lemmatizer (stem, lem, word= "forgave", pos= wo
rdnet.VERB)
Stemmer: forgav
Lemmatizer: forgive
>>> compare_stemmer_and_lemmatizer (stem, lem, word= "better", pos= wor
dnet.ADJ)
Stemmer: better
Lemmatizer: good
>>> compare_stemmer_and_lemmatizer (stem, lem, word= "best", pos= wordn
et.ADJ)
Stemmer: best
Lemmatizer: best
```

Как и предполагалось PoS- теги улучшили качество распознавания базовой формы слова при лемматизации. Лемматизатор определил первоначальную форму неправильных глаголов, существительных с особым способом формирования множественного числа и прилагательных сравнительной степени, но не превосходной.

Задачей стемминга и лемматизации является сведение слова к базовой форме, но они отличаются по принципу работы.

Стемминг – грубый эвристический процесс, который отсекает концы слов, часто включает в себя удаление словообразовательных аффиксов, не имеет словарей – предварительный этап лемматизации.

Лемматизация, используя словарь и морфологический анализ слов, стремится удалить только инфлекционные окончания и вернуть словарную форму слова (лемму).

Что касается реализации: лемматизация – более сложный процесс и обычно требует лексики. При стемминге удовлетворительных результатов можно добиться с помощью довольно простых подходов, основанных на правилах.

Выводы по главе 2

На основе проведённой работы можно сделать следующие выводы:

- текст – последовательность символов и слов. Модели для их обработки могут формировать понимания в простейшей форме, но данные модели лишь отражают статистическую структуру текстов.
- обработка текста состоит из нескольких важных этапов. Сначала текст обрабатывается на лексическом уровне, затем производится морфологический анализ, далее синтаксический, и, наконец, последний основывается на результате третьего этапа и представляет собой синтаксическую структуру каждого предложения. Все эти этапы взаимосвязаны между собой.
- существуют три подхода морфологического анализа: «точная морфология», «неточная морфология» и вероятностный подход, последний не применятся при обработке русских текстов из-за того, что он флективный
- алгоритмический стемминг, даже при низкой эффективности, по сравнению со словарным, является более распространённым.
- лемматизация и стемминг – это частные случаи нормализации. Стемминг является предварительным этапом лемматизации. Особенность лемматизации – приведение слова к первоначальной форме (good- better) – дополняет уникальность стемминга – срезание аффиксов. Таким образом стемминг и лемматизация компенсируют друг друга.
- стемминг и лемматизатор не всегда могут определить базовую форму слова и не во всех случаях определяют правильно. Для улучшения качества лемматизации можно добавить дополнительные команды, например, теги.

ЗАКЛЮЧЕНИЕ

В данной работе были рассмотрены основные методы определения основы слова в машинном переводе: стемминг и лемматизация. Машинный перевод является одним из основных задач обработки естественного языка. Обработка естественного языка – комплексный процесс, состоящий из последовательности сложных операций (этапов), анализирующих и синтезирующих текст для выполнения поставленной задачи.

Этапы обработки текстов позволяют выполнить задачи ОЕЯ. В теоретической части данной работы рассмотрено появление, развитие и постепенное улучшение качества машинного перевода во всех аспектах (эффективности, скорости, требуемых ресурсах, адекватности перевода), что отражается в развитии трех основных подходов: машинный перевод на базе лингвистических правил, статистический и нейронный машинный перевод.

Каждый подход доминировал в свою “эпоху” пока не появлялся более эффективный метод, к примеру машинный перевод на базе лингвистических правил был преобладающей системой вплоть до 1990-х, пока не появились параллельные корпуса, что способствовало развитию статистического машинного перевода.

Как было выявлено ранее, у каждого подхода есть как свои сильные стороны, так и недостатки в выполнении конкретных задач – именно поэтому некоторые компании предпочитают использовать гибридные системы, например, Яндекс Переводчик, использует нейронный и статистический подходы. Создание “идеального” подхода является трудной и почти недосягаемой задачей, в первую очередь из-за постепенных изменений языковых форм и языка в целом. Таким образом, машинный перевод представляет собой бесконечно изменяющий и развивающийся процесс, а с развитием машинного перевода и других технологий – NLP стала одной из самых важных технологий искусственного интеллекта.

Определение основы слова в машинном переводе касается автоматического анализа словоформ текста, перевода слов из естественного языка в “свой”, для дальнейшего синтеза текста.

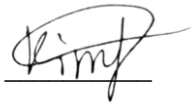
В практической части показан принцип работы определения основы слова системой такими алгоритмами, как стемминг и лемматизация. Алгоритмы рассмотрены в языке программирования Python с разными библиотеками. В ходе проведенной работы было выявлено, что:

1) стемминг и лемматизация основаны на регулярных выражениях. Регулярные выражения широко используется языками программирования для проведения анализа и синтеза текстов, перевода языков в язык, позволяющий системам понимать естественные языки;

2) многое зависит от лексической базы, начиная от принципа ввода команд и заканчивая результатом определения основы слова, что подтверждает гипотезу №2;

3) стемминг и лемматизация имеют свои недостатки – избыточность и недостаточность, неправильное определение основы слова или даже опущение определения основы слова. Но данные алгоритмы компенсируют недостатки друг друга: стемминг эффективен в “легких” случаях определения, где можно срезать аффиксы по шаблону (cats= ca<u>t</u>-s) – лемматизация принимает результат данного процесса т.к. слово находится в словаре. Но в случаях с видоизменённым корнем в словоформе, например, wolves –wolf – стеммер отрежет ***–es*** тем самым определив основу ***wolv***, но это не является словарной формой слова и, поэтому, лемматизатор сравнив слово в словаре лексем приводит его к основе wolf (именительному падежу, единственному числу). Гипотеза №1 подтверждается;

4) для улучшения качества определения основы слова используются дополнительные параметры и/или команды, например, PoS-теги.



Саликаева Д.Э.

Список использованных источников и литературы

1. Андреева, А. Д. Обзор систем машинного перевода / А. Д. Андреева, И. Л. Меньшиков, А. А. Мокрушин. // Молодой ученый. 2013. –№ 12 (59). – С. 64-66.
2. Большакова Е.И. Автоматическая обработка текстов на естественном языке и анализ данных: учебное пособие / Большакова Е.И., Воронцов К.В., Ефремова Н.Э, и др. // М.: Изд-во НИУ ВШЭ, 2017. – 269 с.
3. Большакова. Е.И. Автоматическая обработка текстов на естественном языке и компьютерная лингвистика: учебное пособие / Клышинский Э.С., Ландэ Д.В., Носков А.А., Пескова О.В., Ягунова Е.В. 2011 // М.: Изд-во МИЭМ, 2011. — 272 с.
4. Зализняк А.А. Грамматический словарь русского языка. Словоизменение. / Зализняк А.А. // М.: 3-е изд. Русский язык. 1987. – 800 с.
5. Кан Д. А. Применение теории компьютерной семантики русского языка и статистических методов к построению системы машинного перевода: диссертация кандидата физико-математических наук: 05.13.11 – Санкт-Петербург, 2011. – 129 с.
6. Маннинг К.Д. Введение в информационный поиск / Маннинг К.Д, Рагхаван П., Шютце Х.// М.: 2011. – 528 с.
7. Мифтахова Р.Г. Машинный перевод. Нейроперевод / Мифтахова Р.Г., Морозкина Е.А. // Вестник Башкирского университета. 2019. – Т. 24. №2 – 497-502 с.
8. Мифтахова Р.Г. Проблемы обработки естественного языка в машинном переводе // Автономная некоммерческая образовательная организация "Махачкалинский центр повышения квалификации". Махачкала, 2014. – 22-32 с.
9. Мифтахова Р.Г. Технологии машинного перевода. Нейроперевод. / Мифтахова Р.Г., Черепанова Е.М. // Доклады Башкирского университета 2018. – Т. 3. №6 – 711-715 с.
10. Николенко С. Глубокое обучение: погружение в мир нейронный сетей. / Николенко С., Кадурин А., Архангельская Е. // СПб.: Питер, 2018. – 480 с.
11. Риз Р. Обработка естественного языка на Java/ пер. с англ А.В Снастина. –М.: ДМК Пресс, 2016. – 264 с.

12. Ушаков Д.Н. Толковый словарь современного русского языка – М.: Изд-во “Аделант”, 2014. –800с.

13. Фаленов М.Е., Библия – СПб.: БХВ Петербург, 2009. – 560 с.

14. Шайкевич. А.Я. Введение в лингвистику: учебное пособие. – М.: “Academia”, 2005. – 394с.

15. Шолле Ф. Глубокое обучение на Python. – СПб.: Питер, 2018 – 400с.

16. Bird S. Natural Language Processing with Python/ Bird S., Klein E., Loper E. // Sebastopol: O’Reilly Media, 2009. – p.482.

17. Brinton, L.J. The structure of modern English – Amsterdam; Philadelphia: John Benjamins, 2000. – p. 335.

18. Brown, P. F. A statistical approach to machine translation. / Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Lafferty, J. D., … & Roossin, P. // Computational linguistics. 1990. – Vol. 16, №2. – p.79–85.

19. C.J. van Rijsbergen. New models in probabilistic information retrieval. / S.E. Robertson and M.F. Porter. // British Library Research and Development Report. – London: British Library. 1980. – №5587. p. 261-270.

20. Hull, D.A. Stemming algorithms: a case study for detailed evaluation // Journal of the American Society for Information Science. 1996. – Vol. 47, №1. – p. 70-84.

21. Luong M.T. Effective Approaches to Attention-based Neural Machine Translation // Luong M.T., Pham H., Manning C.D // Conference on Empirical Methods in Natural Language Processing. (Lisbon (Portugal) 17-21 September, 2015). – p. 1412 – 1421.

22. Porter, Martin F. An Algorithm for Suffix Stripping // Program: electronic library and information systems. 1980. — Т. 14, № 3.p. 130-137.

23. Santini, M. Common criteria for genre classification: annptation and granularity // 3-d international workshop on text-based information retrieval (TIR-06). – Riva del Garda, Italy: University of Trento, 2006. p. 35-40.

24. Weaver, W. Translation. Machine translation of languages. 1955. – Т.14. p. 15–23.

Электронные ресурсы

25. Википедия. Естественный язык. URL: https://ru.wikipedia.org/wiki/

26. Как победить морников: Яндекс запустил гибридную систему перевода. URL: https://yandex.ru/blog/company/kak-pobedit-mornikov-yandeks-zapustil-gibridnuyu-sistemu-perevoda

27. Лицензионное соглашение MyStem. URL: https://yandex.ru/legal/mystem/

28. Лингвистика и обработка текстов URL: https://www.osp.ru/os/2013/04/13035562

29. Обработка естественного языка // Краткое руководство. 16 июня 2018. URL: https://coderlessons.com/tutorials/akademicheskii/obrabotka-estestvennogo-iazyka/obrabotka-estestvennogo-iazyka-kratkoe-rukovodstvo

30. Подходы лемматизации с примерами. URL: https://webdevblog.ru/podhody-lemmatizacii-s-primerami-v-python/

31. Стемминг текстов на естественном языке // L-Tips. URL: http://r.psylab.info/blog/2015/05/26/text-stemming/

32. Segalovich I. A fast morphological algorithm with unknown word guessing induced by a dictionary for a web search engine [Электронный ресурс] // Yandex-Team. URL: http://cache-mskdataline03.cdn.yandex.net/download.yandex.ru/company/iseg-las-vegas.pdf

33. https://ru.qwe.wiki/wiki/Rulebased_machine_translation#Types_of_RBMT

34. http://www.inf.ed.ac.uk/teaching/courses/mt/lectures/history.pdf

Саликаева Д.Э.

Приложение №1

PREFIX/SUFFIX/ROOT	MEANING	EXAMPLE	PREFIX/SUFFIX/ROOT	MEANING	EXAMPLE
1. acou-	hear	acoustics	41. fiss-	split	fission
2. aer-, aero-	air	aerodynamics	42. -flect	bend	reflection
3. alt-, alti-, alto-	height	altitude	43. flu-	flow	flux
4. ampli-	large	amplitude	44. for-, fort-	strong, strength	fortification
5. angl, angul-	angle	triangulation	45. -fract	break	fracture
6. ann-, annu-	year	annual	46. frict-	rub	friction
7. ap-, apo-	away from	apogee	47. fus-	melting	fusion
8. astr-, astro-, aster-	star	astronomical	48. -fy	to make	magnify
9. -ation, -ition, -tion	process, result	condensation	49. ge-, geo, -gee	earth	geothermal
10. atmo-	air	atmosphere	50. -gen	to produce, origin	hydrogen
11. -atude, -itude, -tude	condition, state of	amplitude	51. -gon	angle	hexagonal
12. aud-, audio-	hear	audiologist	52. grad-	step	gradient
13. bar-, baro-	heavy	barometric	53. -gram	written record	cardiogram
14. calor-	heat	calorimetry	54. -graph	recording	spectrograph
15. can-, cand-	glow white	incandescence	55. grav-	heavy, weighty	gravitational
16. cap-, capac-	hold	capacitor	56. gyr-	rotate	gyroscope
17. carb-, carbo-	carbon	carbohydrate	57. helio-	sun	heliocentric
18. -ced, -ceed, -cess	going	recessional	58. hemi-	half	hemisphere
19. -celer	swift, to hasten	accelerometer	59. hepta-	seven	heptathlon
20. centr-, centri-	center	centripetal	60. hetero-	unlike, different	heterotroph
21. chem-, chemo-	transmutation	chemical	61. hexa-	six	hexagon
22. chrom-, chromo-	color	chromatograph	62. homo-	same, like	homophone
23. chron-, chrono-	time	chronograph	63. hydr-, hydro-	water	hydrothermal
24. circ-, circl-	circle	circulation	64. -ic, -tic	person	lunatic
25. con-, com-	with, together	compound	65. -ical	pertaining to	mechanical
26. con-, coni-	cone	conical	66. -ician, -icist	specialist	physicist
27. cosm-, cosmo-	universe	cosmological	67. -ics, -tics-	skill	mechanics
28. cry-, cryo-	cold	cryogenic	68. ign-	fire	ignition
29. cycly-, cyclo-	circle	cyclone	69. -ile	pertaining to, thing	projectile
30. deca-	ten	dodecagon	70. infra-	below	infrasonic
31. di-	two	dipole	71. inter-	between	interstitial
32. -duc, -duce, -duct	to lead	aqueduct	72. intra-	within	intramural
33. dyn-, dynam-	power	hydrodynamic	73. iso-	equal	isobaric
34. elect-, electro-	electric, amber	electrician	74. -ist	person	chemist
35. -ence	state of	luminescence	75. -istry	skill	artistry
36. end-, endo-, ento-	inside, within	endothermic	76. -ject	to throw	projectile
37. equa-, equi-, equ-	equal	equilibrium	77. kin-, kine-	motion, movement	kinematics
38. erg-, ergo-	work	ergonomics	78. lept-	small	lepton
39. ex-, exo-	outside, out of	exogenic	79. lev-	to raise	levitation
40. ferro-	iron	ferromagnetic	80. libr-, libri-	weight	equilibrium

Just a sample of the amount of prefixes, suffixes, and roots in the English language