Рабочая программа дисциплины (модуля)*

Интеллектуальный анализ текстов

Направление подготовки (специальность)

01.03.02 Прикладная математика и информатика

Направленность (профиль)

Прикладная математика и искусственный интеллект

Присваиваемая квалификация (степень)

бакалавр

Форма обучения

очная

Год набора 2024

*Рабочая программа дисциплины (модуля) адаптирована для инклюзивного обучения инвалидов и лиц с ограниченными возможностями здоровья

Челябинск 2023 г.

Содержание

1. ЦЕЛИ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Целью дисциплины является ознакомление студентов с методами интеллектуального анализа текстов, освоение принципов морфологического, синтаксического и семантического анализа, извлечения информации из текстов, классификации текстов, основами машинного перевода. Основные задачи дисциплины: - научить использовать известные программные инструменты обработки текстов - ставить и решать конкретные задачи анализа текстов - применять методы машинного обучения для анализа текстов - решать задачи извлечения из текстов именованных сущностей - выявлять фейковые новости - определять психологические особенности автора по его тексту.

2. МЕСТО ДИСЦИПЛИНЫ В СТРУКТУРЕ ОПОП

Цикл (раздел) ОПОП:	Б1.В.1.14

2.1 Требования к предварительной подготовке обучающегося:

Основы компьютерного зрения

2.2 Дисциплины и практики, для которых освоение данной дисциплины (модуля) необходимо как предшествующее:

3. КОМПЕТЕНЦИИ ОБУЧАЮЩЕГОСЯ, ФОРМИРУЕМЫЕ В РЕЗУЛЬТАТЕ ОСВОЕНИЯ ДИСЦИПЛИНЫ (МОДУЛЯ)

ПК-12: Способен создавать и внедрять одну или несколько сквозных цифровых субтехнологий искусственного интеллекта

Знать:

[ПК-9.1. З-1.] принципы построения систем компьютерного зрения, методы и технологии искусственного интеллекта для анализа изображений и видео, методы и подходы к планированию и реализации проектов по созданию систем искусственного интеллекта на основе сквозной цифровой субтехнологии "Компьютерное зрение";
[ПК-9.3. З-1.] фундаментальные правила построения систем поддержки принятия решений, основанных на интеллектуальных принципах, методы и подходы к планированию и реализации проектов по созданию систем искусственного интеллекта на основе сквозной цифровой субтехнологии "Рекомендательные системы и системы поддержки принятия решений";
[ПК-9.2. З-1.] принципы построения систем обработки естественного языка, методы и технологии искусственного интеллекта для анализа естественного языка, методы и подходы к планированию и реализации проектов по созданию систем искусственного интеллекта на основе сквозной цифровой субтехнологии "Обработка естественного языка";
[ПК-9.3. З-1.] фундаментальные правила построения рекомендательных систем, основанных на интеллектуальных принципах, методы и подходы к планированию и реализации проектов по созданию систем искусственного интеллекта на основе сквозной цифровой субтехнологии "Рекомендательные системы и системы поддержки принятия решений";
[ПК-9.4. З-1.] принципы построения систем распознавания и синтеза речи, методы и подходы к планированию и реализации проектов по созданию и поддержке систем искусственного интеллекта на основе сквозной цифровой субтехнологии "Распознавание и синтез речи"

Уметь:

[ПК-9.1. У-1.] применять методы и подходы к планированию и реализации проектов по созданию и поддержке системы искусственного интеллекта на основе сквозной цифровой субтехнологии "Компьютерное зрение";
[ПК-9.2. У-1.] применять методы и подходы к планированию и реализации проектов по созданию системы искусственного интеллекта на основе сквозной цифровой субтехнологии "Обработка естественного языка";
[ПК-9.3. У-1.] применять методы и подходы к планированию и реализации проектов по созданию систем искусственного интеллекта на основе сквозной цифровой субтехнологии "Рекомендательные системы и системы поддержки принятия решений";
[ПК-9.4. У-1.]применять методы и подходы к планированию и реализации проектов по созданию и поддержке системы искусственного интеллекта на основе сквозной цифровой субтехнологии "Распознавание и синтез речи"

Владеть:

построением системы поддержки принятия решения, основанной на интеллектуальных принципах; участием в разработке экспертных систем

В результате освоения дисциплины обучающийся должен

3.1	**Знать:**

3.1.1	принципы построения систем обработки естественного языка, методы и технологии искусственного интеллекта для анализа естественного языка, методы и подходы к планированию и реализации проектов по созданию систем искусственного интеллекта на основе сквозной цифровой субтехнологии "Обработка естественного языка".
3.2	**Уметь:**
3.2.1	применять методы и подходы к планированию и реализации проектов по созданию системы искусственного интеллекта на основе сквозной цифровой субтехнологии "Обработка естественного языка".
3.3	**Владеть:**
3.3.1	

4. ОБЪЕМ ДИСЦИПЛИНЫ (МОДУЛЯ)

Общая трудоемкость	**2 ЗЕТ**
Часов по учебному плану : 72 в том числе : аудиторные занятия : 36 самостоятельная работа : 31,75 : контактная работа: 40,25 ИКР: 4,25	Виды контроля в семестрах: зачеты 8

5. СТРУКТУРА И СОДЕРЖАНИЕ ДИСЦИПЛИНЫ (МОДУЛЯ)

Код занятия	Наименование разделов и тем /вид занятия/	Семестр / Курс	Часов	Литература
	Раздел 1. Введение в интеллектуальный анализ текстов			
1.1	Основные определения. Постановки задач обработки текстов и подходы к их решению. Особенности естественных языков. Типология естественных языков. Уровни анализа. Лингвистическое исследование. /Лек/	8	2	Л1.1 Л1.2Л2.1 Л2.2
1.2	Регулярные выражения /Лаб/	8	2	Л1.1 Л1.2Л2.1 Л2.2
1.3	Токенизация /Лаб/	8	2	Л1.1 Л1.2Л2.1 Л2.2
1.4	Подготовка к лекциям и лабораторным работам /Ср/	8	4	Л1.1 Л1.2Л2.1 Л2.2
	Раздел 2. Методы лингвистического анализа текстов			
2.1	Методы морфологического анализа. Проблема морфологической многозначности. Методы снятия морфологической многозначности. /Лек/	8	2	Л1.1 Л1.2Л2.1 Л2.2
2.2	Синтаксис естественных языков. Способы описания синтаксической структуры предложения. Методы синтаксического анализа. /Лек/	8	2	Л1.1 Л1.2Л2.1 Л2.2
2.3	Способы формализации семантики. Методы семантического анализа. Реляционно-ситуационный анализ текста. Дистрибутивная семантика. /Лек/	8	2	Л1.1 Л1.2Л2.1 Л2.2
2.4	Исправление опечаток /Лаб/	8	4	Л1.1 Л1.2Л2.1 Л2.2
2.5	Морфологический анализ /Лаб/	8	4	Л1.1 Л1.2Л2.1 Л2.2
2.6	Синтаксический анализ /Лаб/	8	4	Л1.1 Л1.2Л2.1 Л2.2
2.7	Подготовка к лекциям и лабораторным работам /Ср/	8	13,75	Л1.1 Л1.2Л2.1 Л2.2

	Раздел 3. Прикладные задачи интеллектуального анализа текстов			
3.1	Извлечение информации из текстов. Классификация и кластеризация текстов /Лек/	8	2	Л1.1 Л1.2Л2.1 Л2.2
3.2	Методы машинного обучения для решения задач обработки естественного языка. /Лек/	8	2	Л1.1 Л1.2Л2.1 Л2.2
3.3	Извлечение именованных сущностей /Лаб/	8	4	Л1.1 Л1.2Л2.1 Л2.2
3.4	Анализ тональности /Лаб/	8	4	Л1.1 Л1.2Л2.1 Л2.2
3.5	Подготовка к лекциям и лабораторным работам /Ср/	8	8	Л1.1 Л1.2Л2.1 Л2.2
	Раздел 4. Иная контактная работа			
4.1	Индивидуальны консультации, текущий контроль. /КурсР/	8	4,25	Л1.1 Л1.2Л2.1 Л2.2
	Раздел 5. Подготовка к зачету			
5.1	Подготовка к зачету /Ср/	8	6	Л1.1 Л1.2Л2.1 Л2.2

6. ФОНД ОЦЕНОЧНЫХ СРЕДСТВ

6.1. Перечень видов оценочных средств

Лабораторная работа № 1 "Регулярные выражения"
Лабораторная работа № 2 "Токенизация"
Лабораторная работа № 3 "Исправление опечаток"
Лабораторная работа № 4 "Морфологический анализ"
Лабораторная работа № 5 "Синтаксический анализ"
Лабораторная работа № 6 "Извлечение именованных сущностей"

6.2. Типовые контрольные задания и иные материалы для текущей аттестации

Типовые задания для лабораторных работ:
см. приложение.

6.3. Типовые контрольные вопросы и задания для промежуточной аттестации

1. Что такое регулярные выражения? Для чего они применяются?
2. Компиляция регулярных выражений
3. Функции Python для работы с регулярными выражениями.
4. Какие существуют операции над строками в языке Python?
5. Какие существуют методы обработки строк?
6. Преимущества и недостатки использования для обработки строк методов и регулярных выражений.
7. Что такое сегментация слов.
8. Проблемы токенизации слов в тексте.
9. Сегментация предложений в тексте. Бинарный классификатор выделения предложений.
10. Нормализация и лемматизация слов.
11. Стемминг текста. Особенности использования стеммера Портера
12. Токенизация слов с помощью библиотеки nltk python. Функции модуля nltk.tokenize.
13. Использование регулярних выражений для выделения токенов.
14. Каким методом объекта класса MorphAnalyzer() можно получить морфологический разбор заданного слова? Что является параметром данного метода?
15. Что собой представляет морфологический разбор, возвращаемый данным методом?
16. Какими свойствами и методами обладает полученный морфологический разбор?
17. Как получить тег морфологического разбора? Что собой представляет полученный тег?
18. Как получить нормальную форму слова?
19. Что подразумевается под понятием “граммема” в pymorphy2? Как получить значение граммемы для конкретного морфологического разбора?
20. Как получить кириллические названия граммем?
21. Для чего используется свойство грамматического разбора score?
22. Как определяется параметр score в морфологическом разборе pymorphy2?

23. Основные способы формализации синтаксической структуры предложения.
24. Деревья синтаксической зависимости. Проективные деревья синтаксической зависимости. Размеченные деревья синтаксической зависимости.
25. Формальное определение грамматики. Контекстно-свободные грамматики (КСГ).
26. Синтаксический анализ. Задача синтаксического анализа.
27. Дерево разбора для КСГ
28. Использование контекстно-свободных грамматик для реализации размеченных систем составляющих.
29. Неоднозначные контекстно-свободные грамматики и синтаксическая омонимия.
30. Показать многозначность контекстно-свободной грамматики, распознающей следующие предложения:
- Japanese eat sushi with the hands.
- Japanese eat sushi with pickled ginger.
31. Показать два способа формального выражения синтаксической структуры следующих предложений:
- Дипломная работа студента включает четыре раздела.
- The small boy put the tortoise on the colored rug.

6.4. Критерии оценивания

Лабораторная работа № 1

В задании 3 задачи. За решение каждой задачи можно получить до 2 баллов: задание выполнено полностью и правильно -2 балла; имеются незначительные ошибки -1 балл;
приложение не работоспособно – 0 баллов; Ответ на контрольный вопрос должен продемонстрировать понимание механизмов NLP, за ответ можно получить оценку: ответ полный и правильный -2 балла; ответ не полный – 1 балл; ответ не удовлетворительный – 0 баллов.
Максимальная оценка - 8 баллов

Лабораторные работы № 2 - 6

В задании 4 задачи. За решение каждой задачи можно получить до 2 баллов: задание выполнено полностью и правильно -2 балла; имеются незначительные ошибки -1 балл; приложение не работоспособно – 0 баллов; Ответ на контрольный вопрос должен продемонстрировать понимание механизмов NLP, за ответ можно получить оценку: ответ полный и правильный -2 балла; ответ не полный – 1 балл; ответ не удовлетворительный – 0 баллов.
Итого, максимальная оценка – 10 баллов

Зачет проводится в очной форме по билетам. Процедура прохождения зачета не является обязательной если по результатам текущего контроля БРС у студента положительная оценка и он с ней согласен. В каждом билете 2 теоретических вопроса. Суммируются оценки за каждый вопрос. Критерии оценки: полные и правильные ответы на вопрос билета и дополнительные вопросы - 3 балла; не полные или не совсем правильные ответы на вопрос билета и дополнительные вопросы - 2 балла; неправильный ответ на вопрос билета, правильные ответы на дополнительные вопросы - 1 балл; неудовлетворительные ответы на вопрос билета и дополнительные вопросы - 0 баллов. Итого, максимальная оценка - 6 баллов
Студент устно отвечает на вопросы билета. Студент должен находиться в аудитории на протяжении всей процедуры зачета. Число студентов, одновременно находящихся в аудитории, где сдается зачет, не более 8 человек. На подготовку к ответу студенту отводится не более 30 мин. Когда обучающийся будет готов к ответу, ему задаются контрольные вопросы по содержанию билета. Студент должен устно ответить на вопросы билета и дополнительные вопросы по теме билета в течение 5 мин. На этом основании преподаватель выставляет баллы за зачетную работу. Добор баллов осуществляется путем выполнения дополнительных заданий из контрольных материалов.

Полученные студентами баллы суммируются, итоговая оценка выставляется исходя из полученной суммы баллов:
От 0 до 40 баллов – «незачтено»
От 41 до 58 баллов – «зачтено».

7. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ (МОДУЛЯ)

7.1. Рекомендуемая литература

7.1.1. Основная литература

	Авторы, составители	Заглавие	Издательство, год	Ресурс
Л1.1	Гольдберг Й.	Неи?росетевые методы в обработке естественного языка (https://e.lanbook.com/book/131704)	Москва : ДМК Пресс, 2019	ЭБС
Л1.2	Ганегедара Т.	Обработка естественного языка с TensorFlow (https://e.lanbook.com/book/140584)	Москва : ДМК Пресс, 2020	ЭБС

7.1.2. Дополнительная литература

	Авторы, составители	Заглавие	Издательство, год	Ресурс
Л2.1	Бонцанини М.	Анализ социальных медиа на Python. Извлекайте и анализируйте данные из всех уголков социальной паутины на Python (https://e.lanbook.com/book/108129)	Москва : ДМК Пресс, 2018	ЭБС
Л2.2	Маккинни У.	Python и анализ данных. Первичная обработка данных с применением pandas, NumPy и Jupiter (https://e.lanbook.com/book/348086)	Москва : ДМК Пресс, 2023	ЭБС

7.3 Перечень информационных технологий

7.3.1 Программное обеспечение

Python

LMS Moodle

LibreOffice

7.3.2 Профессиональные базы данных и информационно-справочные системы

1. Электронный каталог научной библиотеки ЧелГУ [Электронный ресурс] : база данных / Челяб. гос. ун-т. – Челябинск, 1992

2. eLIBRARY.RU : научная электронная библиотека : сайт. – Москва, 2000 – . – URL: https://elibrary.ru – Режим доступа: для зарегистрир. пользователей. – Текст : электронный.

3. Mathematical Reviews (MR) : реферативная база данных / American Mathematical Society. – URL: http://www.ams.org/mathscinet/ – Яз. рус., англ. – Режим доступа: для зарегистрир. пользователей ЧелГУ. – Текст : электронный.

8. МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ (МОДУЛЯ)

Для реализации дисциплины используются учебные аудитории для проведения занятий лекционного типа, лабораторных занятий, групповых и индивидуальных консультаций, текущего контроля и промежуточной аттестации, а также помещения для самостоятельной работы.

Учебные аудитории укомплектованы специализированной мебелью и техническими средствами обучения (компьютерная техника с подключением к сети "Интернет", предустановленным программным обеспечением MS Office, Python для лабораторных занятий).

Для проведения занятий лекционного типа предлагаются наборы демонстрационного оборудования и учебно- наглядных пособий, такие как презентации.

Помещения для самостоятельной работы обучающихся оснащены компьютерной техникой с подключением к сети "Интернет" и обеспечением доступа в электронную информационно-образовательную среду университета.

9. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ (МОДУЛЯ)

Изучение дисциплины требует систематического и последовательного накопления знаний, следовательно, пропуски отдельных тем не позволяют глубоко освоить предмет. Именно поэтому посещение лекций обязательно.
Подготовка студента к предстоящей к лекции включает:
 узнайте тему предстоящей лекции (по тематическому плану, по информации лектора);
 ознакомьтесь с учебным материалом по учебнику и учебным пособиям;
 на отдельные лекции приносить соответствующий материал на бумажных носителях, представленный лектором в электронном курсе (таблицы, графики, схемы). Это позволит сэкономить время на конспектирование лекции. Данный материал будет охарактеризован, прокомментирован, дополнен непосредственно на лекции;
 постарайтесь уяснить место изучаемой темы в своей профессиональной подготовке;
 перед очередной лекцией необходимо просмотреть по конспекту материал предыдущей лекции;
 запишите возможные вопросы, которые вы зададите лектору на лекции.
При затруднениях в восприятии материала следует обратиться к основным литературным источникам. Если разобраться в материале опять не удалось, то обратитесь к лектору (по графику его консультаций) или к преподавателю на практических занятиях
Конспектирование лекции – одна из форм активной самостоятельной работы студентов, требующая навыков и умения кратко, схематично, последовательно и логично фиксировать основные положения, выводы, обобщения, формулировки. Каждая учебная дисциплина как наука использует свою терминологию, категориальный, графический материал, которыми студент должен научиться пользоваться и применять по ходу записи лекции.

Последующая работа над текстом лекции воскрешает в памяти ее содержание, позволяет развивать мышление.
Основная задача при слушании лекции – учиться мыслить, понимать идеи, излагаемые лектором. Большую помощь при этом может оказать конспект. Передача мыслей лектора своими словами помогает сосредоточить внимание, не дает перейти на механическое конспектирование. Механическая запись лекции приносит мало пользы. Ведение конспекта создает благоприятные условия для запоминания услышанного, т.к. в этом процессе принимают участие слух, зрение и рука. Конспектирование способствует запоминанию только в том случае, если студент понимает излагаемый материал.
Известный отечественный педагог В.А. Сухомлинский, рекомендовал учиться думать над конспектом уже на лекции и работать над записями ежедневно хотя бы в течение 2 часов. Он советовал также делить конспект на две графы: в первой кратко записывать изложенные лекции, а во второй – то, над чем надо подумать; сюда же следовало заносить узловые, главные вопросы, над которыми надо подумать постоянно, связывая с этим повседневное чтение. Он подчеркивал, что узловые вопросы предмета будут программой, на основе которой припоминается весь материал.
Важно помнить, что ни одна дисциплина не может быть изучена в необходимом объеме только по конспектам. Для хорошего усвоения курса нужна систематическая работа с учебной и научной литературой, а конспект может лишь облегчить понимание и усвоение материала.
Практические занятия (семинары) и лабораторные работы.
Лабораторные занятия завершают изучение наиболее важных тем учебной дисциплины. Они служат для закрепления изученного материала, получения практических навыков разработки программ на Java. а также для контроля преподавателем степени подготовленности студентов по изучаемой дисциплине.
При подготовке к практическим занятиям и лабораторным работам следует:
 ознакомиться с темой и планом занятия, чтобы выяснить круг вопросов, которые будут обсуждаться на занятии;
 внимательно прочитать материал лекций, относящихся к данному семинарскому занятию, ознакомиться с учебным материалом по учебнику и учебным пособиям;
 выписать основные термины;
 ответить на контрольные вопросы, готовиться дать развернутый ответ на каждый из вопросов;
 уясните, какие учебные элементы остались для вас неясными и постарайтесь получить на них ответ заранее (до лабораторного занятия) во время текущих консультаций преподавателя.
Подготовка к лабораторной работе (ЛР) включает в себя текущую работу над учебными материалами с использованием конспектов и рекомендуемой основной и дополнительной литературы; групповые и индивидуальные консультации; самостоятельное решение ситуационных задач, изучение литературы.
Работу с литературой рекомендуется делать в следующей последовательности: беглый просмотр (для выбора глав, статей, которые необходимы по изучаемой теме); беглый просмотр содержания и выбор конкретных страниц, отрезков текста с пометкой их расположения по перечню литературы, номеру страницы и номеру абзаца; конспектирование прочитанного.
Если самостоятельно не удалось разобраться в материале, необходимо сформулировать вопросы и обратиться за помощью к преподавателю на консультации или ближайшей лекции. Рекомендуется регулярно отводить время для повторения пройденного материала, проверяя свои знания, умения и навыки по контрольным вопросам.
Качество учебной работы студентов преподаватель оценивает в зашиты лабораторных работ, выставляя баллы..
Обучающийся имеет право ознакомиться с ними.
Обучающиеся, не отчитавшиеся по каждой не проработанной ими на занятиях теме к началу зачетной сессии, упускают возможность получить положенные баллы за соответствующее контрольное мероприятие.
Самостоятельная работа студентов
Важную роль в изучении курса играет самостоятельная работа студента. Она предполагает подготовку к лекциям, лабораторным занятиям путем изучения литературы по теме предстоящего занятия. Оценка сформированности компетенций по дисциплине осуществляется с помощью выполнения самостоятельных заданий к ЛР и тестов. Все задания снабжены методическими указаниями и примерами их выполнения. Поэтому рекомендуется внимательно изучить эти рекомендации.
Всего предусмотрено 8 лабораторных работ и 2 тестов. Полученные в текущем контроле оценки учитываются в общей оценке курса согласно БРС. При необходимости добор баллов может осуществляться путем выполнения дополнительных заданий или рефератов по указанию преподавателя.

10. СПЕЦИАЛЬНЫЕ УСЛОВИЯ ОСВОЕНИЯ ДИСЦИПЛИНЫ ОБУЧАЮЩИМИСЯ С ИНВАЛИДНОСТЬЮ И ОГРАНИЧЕННЫМИ ВОЗМОЖНОСТЯМИ ЗДОРОВЬЯ

Освоение дисциплины инвалидами и лицами с ограниченными возможностями здоровья осуществляется с использованием специальных технических средств и голо информационных технологий, предоставляемых Ресурсным учебно-методическим центром по обучению инвалидов и лиц с ограниченными возможностями здоровья ЧелГУ по запросу обучающегося.
1. Мобильные специальные технические средства для лиц с нарушениями зрения: портативный компьютер с вводом/выводом шрифтом Брайля с синтезатором речи «ElBraile-W14J G2»; ноутбуки с программной экранного

доступа NVDA; электронные увеличители для удаленного просмотра; видеоувеличители портативные; тифлоплеер; цифровые диктофоны.
2. Мобильные специальные технические средства для лиц с нарушениями слуха: система свободного звукового поля со встроенной совместимостью с FM-устройствами; радиоклассы «Сонет-РСМ» с передатчиком, заушным индуктором и индукционной петлей; система информационная для слабослышащих переносная «Исток» А2 со встроенным плеером – звуковым информатором; документ-камера; программируемые слуховые аппараты индивидуального пользования.
3. Ассистивные информационные технологии: программное обеспечение экранного доступа с синтезом речи NVDA; программы экранного увеличения; программы речевого синтеза для компьютеров и ноутбуков; программы речевого синтеза для мобильных устройств; экранная клавиатура; экранная лупа.
При необходимости для обучающихся с нарушениями зрения на рабочих местах для проведения практических или лабораторных занятий устанавливается специальное программное обеспечение (программа речевой навигации NVDA, речевые синтезаторы, экранные лупы).
В учебные аудитории обеспечивается беспрепятственный доступ для обучающихся инвалидов и обучающихся с ограниченными возможностями здоровья. В каждой аудитории, где обучаются инвалиды и лица с ограниченными возможностями здоровья, предусматривается соответствующее количество мест для обучающихся с учетом нарушений их здоровья.
Для освоения дисциплины инвалидам и лицам с ограниченными возможностями здоровья предоставляется доступ к печатным источникам, имеющимся в научной библиотеке ЧелГУ, с помощью специальных технических средств; доступ к электронным источникам, представленным в форме электронного документа в фонде научной библиотеки ЧелГУ или электронно-библиотечных системах, с помощью специальных технических и программных средств (рабочее место для незрячего пользователя с программным обеспечением экранного доступа с синтезом речи NVDA, рабочее место с компьютерным роллером и клавиатурой Clevy с большими кнопками и с разделяющей клавиши накладкой).
Учебно-методические материалы для обучающихся из числа инвалидов и лиц с ограниченными возможностями здоровья предоставляются в формах, адаптированных к ограничениям их здоровья и восприятия информации:
Для лиц с нарушениями зрения:
- в печатной форме увеличенным шрифтом,
- в форме электронного документа,
- в форме аудиофайла,
- в печатной форме шрифтом Брайля.
Для лиц с нарушениями слуха:
- в печатной форме,
- в форме электронного документа.
Для лиц с нарушениями опорно-двигательного аппарата:
- в печатной форме,
- в форме электронного документа,
- в форме аудиофайла.
Данный перечень может быть конкретизирован в зависимости от контингента обучающихся.
Для инвалидов и лиц с ограниченными возможностями здоровья освоение дисциплины может быть частично или полностью осуществлено с использованием дистанционных образовательных технологий (Moodle, Adobe Connect Pro и пр.).
В освоении дисциплины инвалидами и лицами с ограниченными возможностями здоровья используется индивидуальная работа. Под индивидуальной работой подразумевается две формы взаимодействия с преподавателем: индивидуальная учебная работа (консультации), т.е. дополнительное разъяснение учебного материала и углубленное изучение материала с теми обучающимися, которые в этом заинтересованы, и индивидуальная воспитательная работа. Индивидуальные консультации направлены на индивидуализацию обучения и установлению воспитательного контакта между преподавателем и обучающимся инвалидом или обучающимся с ограниченными возможностями здоровья.
При проведении процедуры оценивания результатов обучения инвалидов и лиц с ограниченными возможностями здоровья по дисциплине обеспечивается выполнение следующих дополнительных требований в зависимости от индивидуальных особенностей, обучающихся:
а) инструкция по порядку проведения процедуры оценивания предоставляется в доступной форме (устно, в письменной форме, в письменной форме шрифтом Брайля, устно с использованием услуг сурдопереводчика);
б) доступная форма предоставления заданий оценочных средств (в печатной форме, в печатной форме увеличенным шрифтом, в печатной форме шрифтом Брайля, в форме электронного документа, задания зачитываются ассистентом, задания предоставляются с использованием сурдоперевода);
в) доступная форма предоставления ответов на задания (письменно на бумаге, набор ответов на компьютере, письменно шрифтом Брайля, с использованием услуг ассистента, устно).
При проведении процедуры оценивания результатов обучения инвалидов и лиц с ограниченными возможностями

здоровья предусматривается использование технических средств, необходимых им в связи с их индивидуальными особенностями. Эти средства могут быть предоставлены ЧелГУ или могут использоваться собственные технические средства.
При необходимости инвалидам и лицам с ограниченными возможностями здоровья предоставляется дополнительное время для подготовки ответа на задания, процедура оценивания результатов обучения по дисциплине может проводиться в несколько этапов.
Проведение процедуры оценивания результатов обучения инвалидов и лиц с ограниченными возможностями здоровья допускается с использованием дистанционных образовательных технологий.

Лабораторная работа 1 Регулярные выражения

Цель: Ознакомиться с использованием регулярных выражений для работы с текстом, сравнить с обработкой методами класса String**.**

Теоретические материалы (см. Приложение 1)

Задания.

1) Всего 3 задачи. Для каждой задачи требуется написать и протестировать регулярное выражение. Ниже приводятся скрипты для тестирования регулярного выражения. Задание можно выполнять в GoogleCollab или в Eclipse.

2) Устно ответить на контрольный вопрос

1 Выделить автомобильные номера

```
### Your code goes here ###
### regexp =
############################

import re

# Тестирование
test_string = "РТУ220О КЕ800К П227НА С227НА Е199КБ Е512КМ О503ДП Р59ОТС А771ВЕ Х281ОН 112МНО Т51РУ Е390МВ М9000РТ М900РТУ В5РСХИ"
assert re.findall(regexp, test_string) == ['С227НА', 'Х281ОН', 'Е390МВ'], 'Проверьте регулярное выражение'
print("Тест пройден")
```

```
NameError                                 Traceback (most recent call last)
<ipython-input-2-20e522138be9> in <cell line: 5>()
      3 # Тестирование
      4 test_string = "РТУ220О КЕ800К П227НА С227НА Е199КБ Е512КМ О503ДП Р59ОТС А771ВЕ Х281ОН 112МНО Т51РУ Е390МВ М9000РТ М900РТУ В5РСХИ"
----> 5 assert re.findall(regexp, test_string) == ['С227НА', 'Х281ОН', 'Е390МВ'], 'Проверьте регулярное выражение'
      6 print("Тест пройден")

NameError: name 'regexp' is not defined
```

Листинг тестирования задачи 1

2 Почистить текст от html-тегов

```
! wget -q https://www.dropbox.com/s/dqp8yjmtu23fblj/moviescripts.html && ls -lah
total 60K
drwxr-xr-x 1 root root 4.0K Aug 27 12:03 .
drwxr-xr-x 1 root root 4.0K Aug 27 12:01 ..
drwxr-xr-x 4 root root 4.0K Aug 24 21:24 .config
-rw-r--r-- 1 root root  42K Aug 27 12:03 moviescripts.html
drwxr-xr-x 1 root root 4.0K Aug 24 21:25 sample_data
```

```
with open('./moviescripts.html', 'r', encoding='utf-8') as f:
  text = f.read()

import re

def clean_func(text):
  ### Your code goes here ###
  ### # Заменяем тег ...
  ### clean_text =
  ### # Удаляем ...
  ### clean_text =
  raise NotImplementedError
  ############################

  return clean_text.strip()  # Удаляем пробелы слева и справа

result = clean_func(text)
```

```
NotImplementedError                       Traceback (most recent
call last)
<ipython-input-5-e581e0503174> in <cell line: 14>()
     12   return clean_text.strip()  # Удаляем пробелы слева и
справа
     13
---> 14 result = clean_func(text)

<ipython-input-5-e581e0503174> in clean_func(text)
      7   ### # Удаляем ...
      8   ### clean_text =
----> 9   raise NotImplementedError
     10   ############################
     11

NotImplementedError:
```

```
# Тестирование
assert '\n' not in result, 'Результат должен выдаваться одной строкой!'
assert result[700:800] == 'мые прекрасные, остаются только идеями и сами по себе ценятся недорого. Иное дело – идеи, разработан'
assert result[-100:] == 'ты должны научиться спокойно расставаться со своими детищами и переключаться на настоящее и будущее.'
print("Тесты пройдены")
```

Листинг тестирования задачи 2

3. С помощью одного выражения с re.sub собрать в предложение набор токенов

```
import re

### Your code goes here ###
### make_sentence = lambda c: ...
############################
```

```
    # Тестирование
    assert make_sentence(["Привет", ",", "как", "дела", "?"]) == 'Привет, как дела?', 'Проверьте регулярное выражение'

    text = "Впервые передо мной забрезжил свет потенциальной осмысленности регэкспов: http://docs.python.org/dev/howto/regex.html понятно и доходчиво объясняет их функцию в роли трафарета,  – использование групп для выделения конкретных участков текста."
    tokens = ['Впервые', 'передо', 'мной', 'забрезжил', 'свет', 'потенциальной', 'осмысленности', 'регэкспов', ':', 'http://docs.python.org/dev/howto/regex.html', 'понятно', 'и', 'доходчиво', 'объясняет', 'их', 'функцию', 'в', 'роли', 'трафарета', ',', ' –', 'использование', 'групп', 'для', 'выделения', 'конкретных', 'участков', 'текста', '.']
    assert make_sentence(tokens) == text, 'Проверьте регулярное выражение'
    print("Тесты пройдены")
```

Листинг тестирования задачи 3

Контрольные вопросы

1 Что такое регулярные выражения?
2 Для чего они применяются?
3 Соответствие символов
4 Метасимволы, их значение и применение
5 Использование регулярных выражений
6 Компиляция регулярных выражений
7 Функции для работы с регулярными выражениями.
8 Какие существуют операции над строками в языке Python?
9 Какие существуют методы обработки строк?
10 Преимущества и недостатки использования для обработки строк методов и регулярных выражений.

Требования к оформлению отчета.

Отчет по ЛР состоит из

а) Титульный лист: название ЛР, ФИО студента и номер группы, № варианта.

б)Текст задания;

в) Исходный текст программы;

г) Скрины выполнения;

д) Выводы.

Критерии оценивания.

За решение каждой задачи вы можете получить до 2 баллов: задание выполнено полностью и правильно -2 балла; имеются незначительные ошибки -1 балл; приложение не работоспособно – 0 баллов;

Ответ на контрольный вопрос должен продемонстрировать понимание механизмов Java, за ответ вы можете оценку: ответ полный и правильный -1 балл; ответ не удовлетворительный – 0 баллов.

Итого, максимальная оценка - 7 баллов

Внимание! Полученная оценка автматически снижается на 2% за каждую полную неделю задержки сдачи отчета по работе, но не более, чем на 40%

Библиографический список

1. Маккинни, У. Python и анализ данных. Первичная обработка данных с применением pandas, NumPy и Jupiter : справочник / У. Маккинни ; перевод с английского А. А. Слинкина. – 3-е изд. – Москва : ДМК Пресс, 2023. – 536 с. – ISBN 978-5-93700-174-0. – Текст : электронный // Лань : электронно-библиотечная система. – URL: https://e.lanbook.com/book/348086 (дата обращения: 26.08.2023). – Режим доступа: для авториз. пользователей

2. Руководство по языку программирования Python. [Электронный ресурс].– URL: https://metanit.com/python/tutorial. – Доступ свободный

3. Методы интеллектуального анализа текстов [Электронный ресурс]. – URL: https://stud-sci.rudn.ru/course/view.php?id=264/ – Гостевой доступ (Зайти гостем)

Приложение 1.

Регулярные выражения в Python

Что такое Regex

Регулярные выражения (Regex) – это строки, задающие шаблон для поиска определенных фрагментов в тексте. Помимо поиска, с помощью специальных Regex-шаблонов можно манипулировать текстовыми фрагментами – удалять и изменять подстроки частично или полностью.

Регулярные выражения состоят из набора литералов (букв и цифр) и метасимволов и выглядят примерно так: r'(https?://)?(www\.)?youtube\.(com|nl)/watch\?v=([\w-]+)(&.*?)?(?=[^-\w&=%])'Используя метасимволы, можно создавать сложные шаблоны, содержащие специальные конструкции для работы с определенными последовательностями и группами символов.

Regex-выражения применяют для обработки текстовых данных, в том числе в скриптах для веб-скрапинга. Кроме того, Regex используют в составе OCR-приложений для очистки отсканированного текста.

Regex в Python

Большинство современных языков программирования поддерживают регулярные выражения, однако степень удобства использования Regex в разных языках варьируется. Python предоставляет простые и понятные методы для работы с регулярными выражениями. Все Regex инструменты находятся в модуле **re**, который входит в стандартный дистрибутив Python – достаточно импортировать его в свой проект:

```
import re
```

Для экранирования служебных символов в шаблонах поиска и замены используют два способа – обратный слэш \ и «сырые» строки r''. Второй метод предпочтительнее – он позволяет избежать нагромождения слэшей в шаблонах.

Основные функции Regex

re.match() – находит вхождение фрагмента **в начале** строки. Обычный формат использования – re.match(r'шаблон', строка):

```
import re
s = "утка крякает, кукушка кукует, петух кукарекает"
match = re.match(r'ку', s)
print(match)
```

Этот код вернет None, несмотря на то, что в строке есть 5 фрагментов «ку». Это происходит потому, что оба фрагмента расположены не в начале строки.

re.search() – находит первое вхождение фрагмента в любом месте и возвращает объект match. Если в строке есть другие фрагменты, соответствующие запросу, re.search их проигнорирует. У re.search есть дополнительные методы:

.span() – возвращает кортеж, содержащий начальную и конечную позиции искомого фрагмента.

.string – вернет строку, переданную в функцию re.search.

.group() – возвращает фрагмент строки, в котором было обнаружено совпадение.

```
#Пример использования re.search с дополнительными методами
import re
s = "от топота копыт пыль по полю летит"
match = re.search(r'по', s)
print(match, match.span(), match.string, match.group(),
sep='\n')

        #Вывод:
<re.Match object; span=(5, 7), match='по'>
(5, 7)
от топота копыт пыль по полю летит
по
```

re.findall() – находит все вхождения фрагмента, в любом месте. Функция re.findall() учитывает регистр символов. Чтобы в результат вошли фрагменты с символами в другом регистре, применяют флаг **re.IGNORECASE**:

```
import re
s = "Не видно, ликвидны акции или неликвидны."
match = re.findall(r'не', s, re.I)
print(match)
```

re.split() – расщепляет строку по заданному шаблону. Количество расщеплений задается флагом – в этом примере от строки отделяется только первое слово:

```
import re
s = "Обладаешь ли ты налогооблагаемой благодатью?"
res = re.split(r' ', s, 1)
print(res)
```

re.sub() – заменяет фрагмент в соответствии с шаблоном:

```
import re
s = "Коала какао лениво лакала"
res = re.sub(r'коала', 'макака', s, flags=re.I)
print(res)
```

re.compile() – создает объект из регулярного выражения. Применяется, если один и тот же поисковый шаблон используется в коде несколько раз:

```
import re
st = re.compile('угнал')
res1 = st.findall("Карл у Клары угнал Maclaren, а Клара
у Карла угнала Corvette.")
res2 = st.findall("Карл у Клары угнал кораллы, а Клара
у Карла угнала кларнет.")
print(res1, res2, sep='\n')
```

Все перечисленные выше примеры предназначены для выполнения самых простых задач по поиску и замене фрагментов текста. Возможности Regex в Python намного шире: шаблоны могут включать условия, учитывать (или игнорировать) группы символов и диапазоны значений. Для создания таких регулярных выражений используют специальные конструкции, состоящие из метасимволов.

Основные метасимволы в Regex

Мета-символ	Описание
[]	используется для указания набора или диапазона символов – re.findall(r'[с-я]', "Камер-юнкер юркнул в бункер", re.I),re.findall(r'[аж]', "ажиотаж, мандраж, багаж")
\	указывает на начало последовательности (мы рассмотрим их ниже) или экранирует служебные символы
.	выбирает любой символ, кроме новой строки \n.
^	проверяет, начинается ли строка с определенного символа / слова / набора символов. Например,r'^Привет' проверит, начинается ли строка с «Привет». Метасимвол ^ в наборе []имеет другое значение – проверяет, отсутствуют ли в строке определенные символы (подробнее об этом ниже).
$	проверяет, заканчивается ли строка в соответствии с шаблоном –r'До свиданья.$'
*	или больше совпадений с шаблоном –r'ко.*аборация'
?	ноль или одно совпадение –r'ф.?нтастика'. Кроме того, нейтрализует «жадность» выражений, которые используют ., *, + для выбора любых символов.
{}	точное число совпадений – r'Интерсте.{2} ар'.
|	любой из двух вариантов – r'уйду|останусь'

()	захватывает группу для дальнейших манипуляций – re.sub(r'(www)', r'\1.', "wwwwear-gear.com").
<>	создает именованную группу – re.search('(?P<группа1>\w+),(?P<группа2>\w+),(?P<группа3>\w+)', 'дом,улица,фонарь')

Последовательности

Знаком слэша \ обозначается специфическая последовательность символов.

Мета-символ	Описание
\A	проверяет, начинается ли строка с определенной последовательности символов. Например, re.findall(r"\AДом", txt), проверит, начинается ли предложение со слова «Дом».
\b	возвращает совпадение, если слово начинается или заканчивается нужной последовательностью символов. Выражение re.findall(r".com\b", s) проверит, есть ли в строке хотя бы одно доменное имя зоны **.com**.
\B	возвращает совпадение, если определенные символы есть в строке, но не в начале или не в конце слова – re.findall(r"\Bро", 'розовая от мороза'), re.findall(r'ин\B', 'синий апельсин').
\d	проверяет, что в строке есть цифры от 0 до 9 – re.findall("\d", 'при пожаре звоните 112').
\D	удостоверяет, что цифр в строке нет – re.findall("\D", 'цифр нет').
\s	проверяет наличие пробелов в строке – re.findall("\s", "один пробел").
\S	возвращает совпадение, если в строке есть любые символы, кроме пробелов – re.findall("\S", "непустая строка").
\w	проверяет, есть ли в строке «словесные» символы – знак нижнего подчеркивания, цифры и буквы – re.findall(r"\w", "_\\\\").
\W	возвращает совпадение по каждому «несловесному» символу – re.findall("\W", "здесь есть такие символы!").
\Z	проверит, заканчивается ли строка нужной последовательностью символов – re.findall("конец\Z", "это конец").

Наборы и диапазоны символов

Наборы и диапазоны в регулярных выражениях заключены в квадратные скобки

Последова-тельность	Описание
[есн]–	проверит, есть ли в строке любой из указанных символов е, с или н – re.findall("[есн]", "здесь есть несколько символов из набора"). Наличие любой цифры из набора проверяется так же –[0169].
[а-е]	вернет совпадения по каждому символу из алфавитного диапазона – re.findall("[а-е]", "здесь есть символы из диапазона"). Таким же образом возвращает совпадения по диапазону цифр – [5-9]. Чтобы не использовать флаг re.IGNORECASE, диапазон можно указывать так – [а-еА-Е].
[^абвгд]	проверит наличие в строке символов, кроме указанных в наборе – re.findall("[^абвгд]", "АБВГДейка – детская передача", re.I).

[0-5][0-9]	возвращает совпадения по двузначным цифрам от 00 до 59 – re.findall("[0-5][0-9]", "будильник сработает в 07:45").

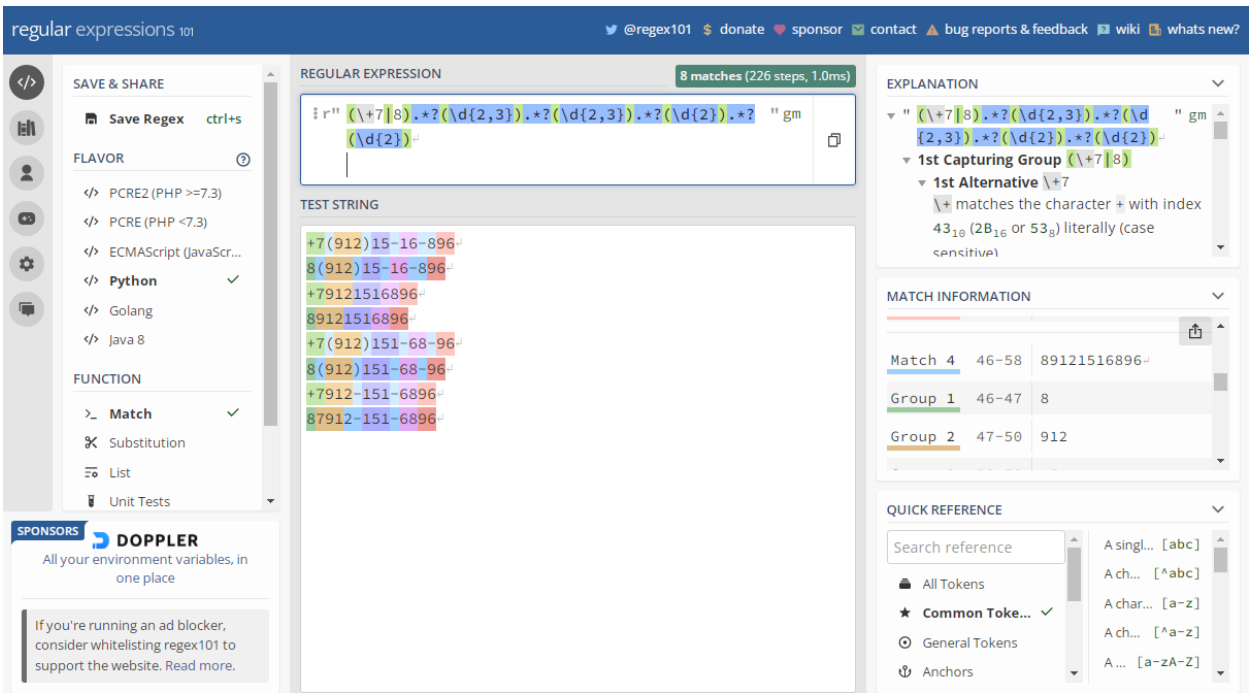
Флаги в Regex

Функциональность регулярных выражений расширяется за счет флагов:

Краткий синтаксис	Полный синтаксис	Назначение
re.A	re.ASCII	Возвращает совпадения только по ASCII-символам вместо всей таблицы Unicode.
re.I	re.IGNORECASE	Игнорирует регистр символов.
re.M	re.MULTILINE	Используется совместно с метасимволами ^ и $. В первом случае возвращает совпадения в начале каждой новой строки \n, во втором – в конце \n.
re.S	re.DOTALL	Заставляет метасимвол . возвращать совпадения по абсолютно всем символам, включая \n. Без этого флага точка . соответствует любому символу, кроме \n.
re.X	re.VERBOSE	Разрешает комментарии в Regex-выражениях.
re.L	re.LOCALE	Учитывает региональные настройки при использовании \w, \W, \b, \B, \s и \S. Используется **только** при работе с байтовыми строками, не совместим с re.ASCII.

Онлайн-конструкторы регулярных выражений

Чем сложнее регулярное выражение, тем труднее его правильно составить и протестировать. В интернете есть немало визуализаторов Regex, которые значительно упрощают эту задачу. Самый удобный ресурс – regex101. Сайт предоставляет справочную и отладочную информацию, позволяет визуально тестировать шаблоны для поиска и замены. Помимо Python, поддерживает PHP, Java, Golang и JavaScript.



Конструктор Regex

Примеры использования регулярных выражений в Python

Задача 1:

Написать регулярное выражение для извлечения из текста всех email-адресов.

Решение:

```
import re
s = 'По всем вопросам пишите на vasiliy-pupkin@gmail.com, или на
secondemail@yandex.ru, отвечу сразу. Или пишите моему ассистенту
secretary@gmail.com!'
emails = re.findall(r'[\w\.-]+@[\w\.-]+', s)
for email in emails:
    print(email)
```

Задача 2:

Имеется файл **transactions.txt**, в котором даты указаны в формате **MM/DD/YYYY**, при этом в некоторых случаях месяц обозначен первыми тремя буквами: **NOV**, **dec**, **JAN**. Нужно привести даты к формату **MM-DD-YYYY**.

```
    #формат дат в файле transactions.txt
nov/14/2021
dec/15/2021
12/16/2021
dec/17/2021
jan/03/2022
JAN/10/22
```

Решение:

```
import fileinput
import re
fn = "transactions.txt"
for line in fileinput.input(fn, inplace=True):
    new_line = re.sub('(\d{2}|[a-yA-Y]{3})\/(\d{2})\/(\d{2,
4})', r'\1-\2-\3', line)
    print(new_line)
```

```
        #Содержимое файла после выполнения кода:
nov-14-2021
dec-15-2021
12-16-2021
dec-17-2021
jan-03-2022
JAN-10-2022
```

Задача 3:

Вводится последовательность строк. Нужно вывести строки, в которых фрагмент «**кот**» присутствует в качестве **подстроки** не менее 2 раз.

```
        #Пример ввода
кот-кот
кот и кот
котофей
котейка кот
кот и котенок
```

Решение:

```
        import re
import sys
for line in sys.stdin:
    line = line.strip()
    if re.search(r"кот.*?кот", line):
        print(line)
```

Задача 4:

Дана последовательность строк. Нужно вывести те, в которых «**кот**» встречается в качестве **отдельного** слова.

```
        #Пример ввода:
кот в сапогах
кошка и кот
котофей
котяра
```

Решение:

```
        import re
import sys
for line in sys.stdin:
    line = line.rstrip()
    if re.search(r"\bкот\b", line):
        print(line)

        #Вывод
кот в сапогах
кошка и кот
```

Задача 5:

Вывести слова, состоящие из двух одинаковых слогов.

```
        #Пример ввода
тартар
тик-так
сносно
варвар
барабан
```

Решение:

```
        import re
import sys
for line in sys.stdin:
    line = line.strip()
    if re.search(r"\b(\w+)\1\b", line):
        print(line)

        #Вывод
тартар
сносно
варвар
```

Задача 6:

Вводится последовательность строк. В каждой строке нужно поменять местами две первые буквы в каждом слове, состоящем из двух и более букв.

```
        #Пример ввода
это пример текста
в котором нужно поменять буквы
```

Решение:

```
        import sys
import re
for line in sys.stdin:
    line = line.rstrip()
    print(re.sub(r'\b(\w)(\w)', r"\2\1", line))

        #Вывод
тэо рпимер еткста
в октором унжно опменять убквы
```

Задача 7:

Напишите функцию для валидации мобильного номера в международном формате. Корректным считается представление номера в таком виде:

```
+7(912)15-16-896, 8(912)15-16-896
+79121516896, 89121516896
+7(912)151-68-96, 8(912)151-68-96
+7912-151-6896, 87912-151-6896
```

Решение:

```
        import re
pattern =
re.compile(r'(\+7|8).*?(\d{2,3}).*?(\d{2,3}).*?(\d{2}).*?(\d{2})
')
def isValid(number):
    if re.match(pattern, number):
        print("ДА")
    else:
        print("НЕТ")
isValid(input())
```

Задача 8:

Напишите программу для парсинга номеров телефонов с тестовой страницы.

Решение:

```
        import urllib.request
from re import findall
url =
"http://www.summet.com/dmsi/html/codesamples/addresses.html"
response = urllib.request.urlopen(url)
data = response.read()
s = data.decode()
phones = findall("\(\d{3}\) \d{3}-\d{4}", s)
for number in phones:
    print(number)
```

Задача 9:

Нужно извлечь все имена и фамилии из текста.

Решение:

```
        import re
s = 'На встрече присутствовали: профессор Владимир Успенский,
физик-ядерщик Сергей Ковалев, президент клуба Владимир Медведев
и космонавт Юрий Титов.'
name = r"[А-Я][а-я]+,?\s+"
last_name = r"[А-Я][а-я]+"
persons = re.findall(name + last_name, s)
for item in persons:
    print(item)
```

Задача 10:

Нужно получить URL всех **png** и **jpg** изображений, использованных на главной странице **proglib.io**:

```
        import re
import requests
def getURL(text):
    urls = []
    results =
re.findall(r'(?:http\:|https\:)?\/\/.*\.(?:png|jpg)', text)
    for x in results:
        if not x.startswith('http:'):
            x = 'http:' + x
            urls.append(x)
    return urls
def getImages(url):
    resp = requests.get(url)
    urls = getURL(resp.text)
    print('urls', urls)
getImages('https://proglib.io')
```

Заключение

Regex в Python – мощный, гибкий, но достаточно сложный инструмент. Регулярные выражения сложно составлять, поддерживать и редактировать. При работе с текстовыми файлами Regex чаще всего можно заменить методами строк, а при парсинге, в большинстве случаев, использование XPath и CSS-селекторов окажется более эффективным.

Приложение 2.

Что такое Google Colab и кому он нужен

(текст взят с сайта https://blog.skillfactory.ru/chto-takoe-google-colaboratory-i-komu-on-nuzhen/#:~:text=%D0%9A%D0%B0%D0%BA%20%D0%BD%D0%B0%D1%87%D0%B0%D1%82%D1%8C%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%D1%82%D1%8C%20%D1%81%20Google%20Colab,-%D0%92%D1%81%D0%B5%20%D0%BF%D1%80%D0%BE%D1%81%D1%82%D0%BE%3A%20%D0%BD%D0%B0&text=%D0%9F%D0%BE%D1%81%D0%BB%D0%B5%20%D0%B7%D0%B0%D0%BF%D1%83%D1%81%D0%BA%D0%B0%20%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D1%8B%20Colab%20%D0%BF%D1%80%D0%B5%D0%B4%D0%BB%D0%BE%D0%B6%D0%B8%D1%82,Google%20Colab%20%D0%BF%D0%BE%D0%B4%D0%BA%D0%BB%D1%8E%D1%87%D0%B8%D1%82%D1%81%D1%8F%20%D0%BA%20%D1%85%D1%80%D0%B0%D0%BD%D0%B8%D0%BB%D0%B8%D1%89%D1%83**).**

Google Colab –облачный блокнот для программирования на Python

Содержание

Блокнот Colab – это бесплатная интерактивная облачная среда для работы с кодом на языке Python от Google в браузере. Принцип у нее такой же, как у остальных онлайн-офисов компании: она позволяет одновременно с коллегами работать с данными. Рассказываем, в чем преимущества Colab для написания кода в каких сферах он может быть полезен разработчикам и не только.

Что такое Google Colab

Google Colab – сервис, созданный Google, который предоставляет возможность работать с кодом на языке Python через Jupyter Notebook, не устанавливая на свой компьютер дополнительных программ. В Google Colab можно применять различные библиотеки на Python, загружать и запускать файлы, анализировать данные и получать результаты в браузере. Этот сервис особенно полезен для разработчиков и студентов, изучающих программирование на Python.

Кому нужен Google Colab

вообще всем, кто работает с Big Data;

аналитикам данных (сортировать данные в файлах за долгий период, делать визуализацию или выстраивать закономерности);

исследователям данных (разрабатывать и тестировать новые модели машинного обучения, составлять прогнозы);

инженерам данных (разрабатывать ПО, системы для хранения больших данных).

В основе «Колаборатории» – блокнот Jupyter для работы с кодом на языке Python, только с базой на Google Диске, а не на компьютере. Здесь те же ячейки (cells), которые поддерживают текст, формулы, изображения, разметку HTML и не только. То есть можно заниматься программированием на языке Python и не качать лишние файлы, кучу библиотек, не перегружать машину и не переживать, что место на жестком диске вот-вот закончится. Единственное условие – нужно иметь Google-аккаунт.

Главная особенность «Колаборатории» – бесплатные мощные графические процессоры GPU и TPU, благодаря которым можно заниматься не только базовой аналитикой данных, но и более сложными исследованиями в области машинного обучения. С тем, что CPU вычисляет часами, GPU или TPU справляются за минуты или даже секунды.

Anaconda и Jupyter Notebook

Дата-сайентисты часто работают с Jupyter Notebook – инструментом, позволяющим запускать и отлаживать алгоритмы небольшими фрагментами. Это удобно для работы с данными: нужно запустить фрагмент кода в ячейке – посмотреть результаты – изменить код – снова запустить и так далее.

Одна из популярных программ, в составе которых есть Jupyter Notebook, – Anaconda. Кроме Jupyter на ваш компьютер будут установлены языки Python и R, а также больше 250 разных программ и библиотек для анализа данных и разработки моделей машинного обучения.

Такое разнообразие нужно не всегда. Также не всегда можно установить все локально. В этом случае помогает Google Colaboratory.

CPU vs. GPU vs. TPU

CPU – центральный процессор – мозг компьютера, который выполняет операции с файлами. Настолько универсален, что может использоваться почти для всех задач: от записи фотографий на флешку до моделирования физических процессов.

GPU – графический процессор. Обрабатывает файлы быстрее, так как задачи выполняет параллельно, а не последовательно, как CPU. Он заточен исключительно под графику, поэтому на нем удобнее работать с изображением и видео, например заниматься 3D-моделированием или монтажом.

TPU – тензорный процессор, разработка Google. Он предназначен для тренировки нейросетей. У этого процессора в разы выше производительность при больших объемах вычислительных задач.

Сами процессоры дорогие, и не каждый может их себе позволить. Платформа Google Colaboratory дает возможность бесплатно и непрерывно пользоваться ими на протяжении 12 часов. Будьте внимательны: как только это время истечет, Colab сотрет все данные и файлы и придется начинать сначала.

Кроме того, Google отключает файлы блокнота после примерно 30 минут бездействия, чтобы не перегружать процессоры. Система Colab так устроена специально: например, многие факторы, в том числе время простоя, максимальная активность, общие ограничения на объем памяти иногда динамически меняются. Активным участникам ненадолго могут ограничить доступ к GPU, чтобы дать возможность использовать процессор другим.

Для чего используется Google Colab

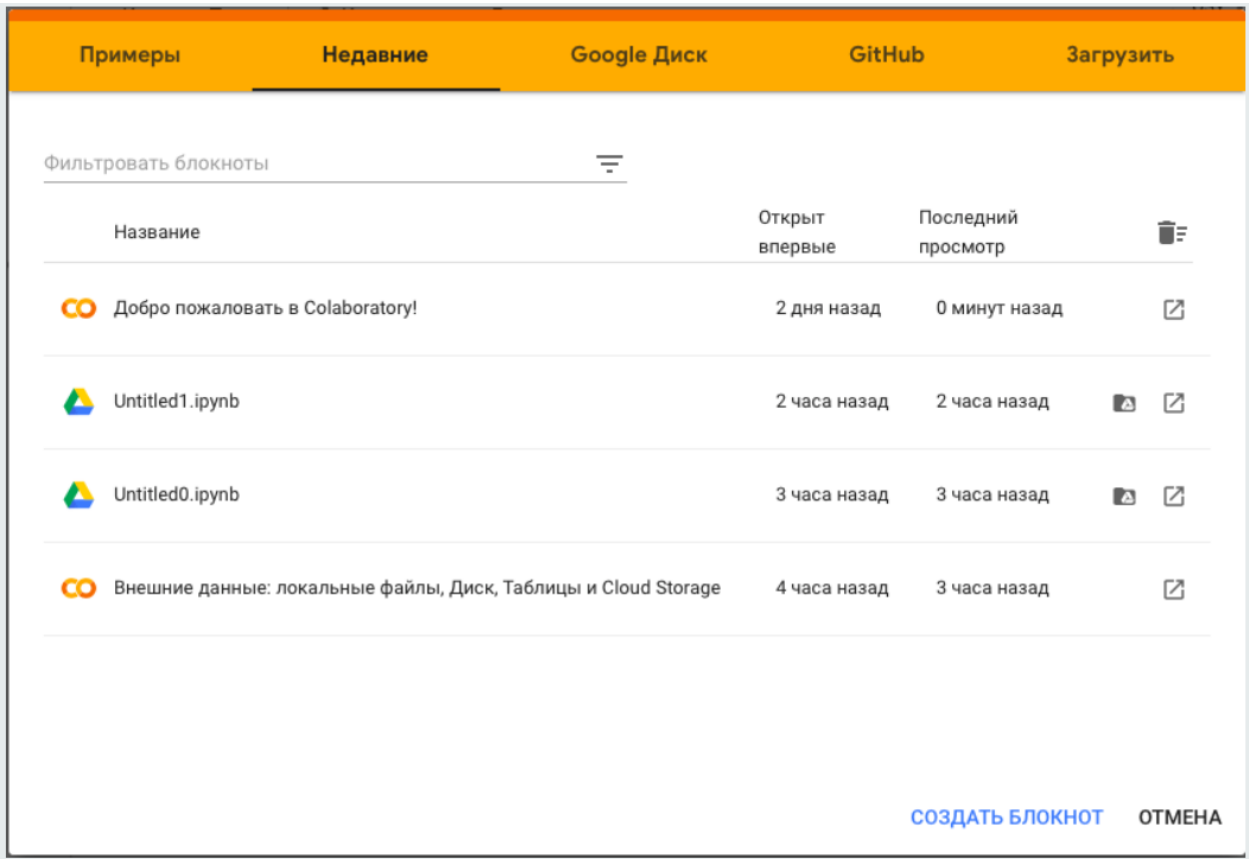
- знакомство с TensorFlow – открытой библиотекой на Python для машинного обучения;
- разработка нейронных сетей;
- эксперименты с TPU;
- распространение исследований в области искусственного интеллекта;
- создание руководств.

Несколько таких примеров есть в открытом доступе прямо в Colab.

Эта гибкость в управлении ограничениями позволяет Colab оставаться бесплатным для всех пользователей. Ну а чтобы все не упало в самый неожиданный момент, можно оформить подписку на Colab Pro за $9,99 в месяц. Там и памяти в два раза больше, и времени работы, и к тому же приоритетный доступ к TPU. Правда, пока Pro-подписка есть только в Канаде и США.

Как начать работать с Google Colab

Все просто: на сайте Google Colab сразу появляется экран с доступными блокнотами. Можно создавать новый или загружать уже разработанный Python-код из Google Диска.



Чтобы работать с файлами и кодом в Google Colab с личного диска, нужно использовать команду `mount()`:

```
   from google.colab import drive
drive.mount ('/content/drive')
```

После запуска команды Colab предложит ввести код авторизации. Открыв URL, вы должны предоставить сервису доступ к своему аккаунту. Тогда он выдаст код, который нужно будет вставить в поле, нажать ВВОД, и Google Colab подключится к хранилищу.

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989

Enter your authorization code:
```

Чтобы проверить, действительно ли Colab подключился, можно использовать команду `!ls "/content/drive/My Drive"`. Она покажет содержимое Google-диска.

Зачем использовать Google Colab

1) Как и Google Документы, он дает возможность работать с Python-библиотеками для анализа данных онлайн.

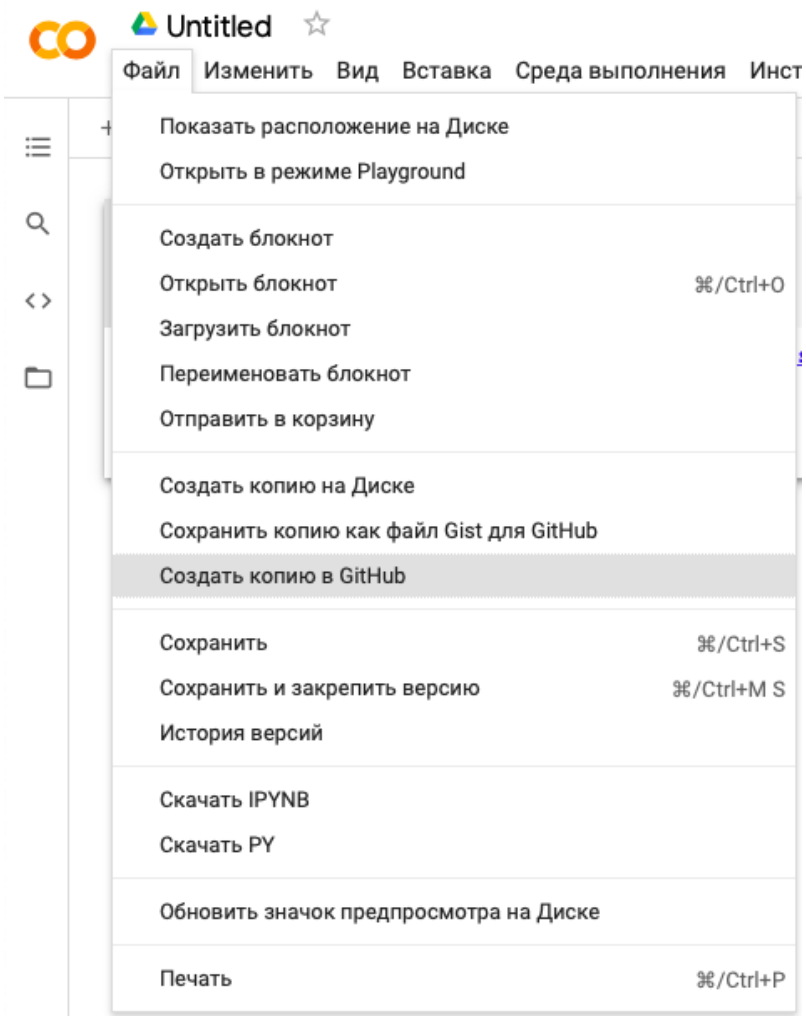
2) «Гугл Коллаб» предоставляет мощные процессоры для облачных вычислений. У него интуитивно понятный интерфейс, который позволяет не перегружать компьютер файлами и процессами и делать все вычисления быстро.

3) Все блокноты под рукой. В Google Colab сохраняется доступ к аккаунту с файлами с любых устройств. Правда, если вы с осторожностью

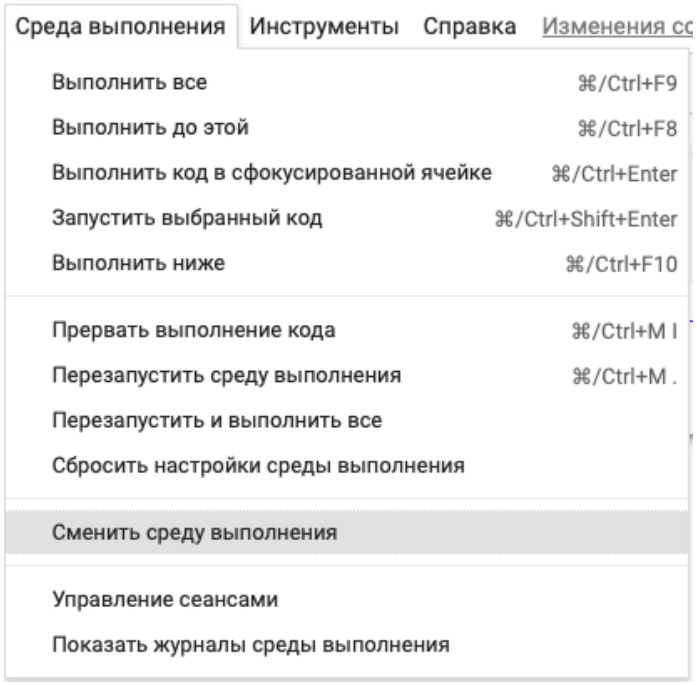
относитесь к своей конфиденциальности, Jupyter Notebook останется более предпочтительным вариантом.

4) Сердце Colab – это совместное использование. При работе над проектом в команде Colab дает возможность свободно править, комментировать и редактировать код с разных аккаунтов, даже если вы сидите на жестком локдауне где-нибудь в Лондоне.

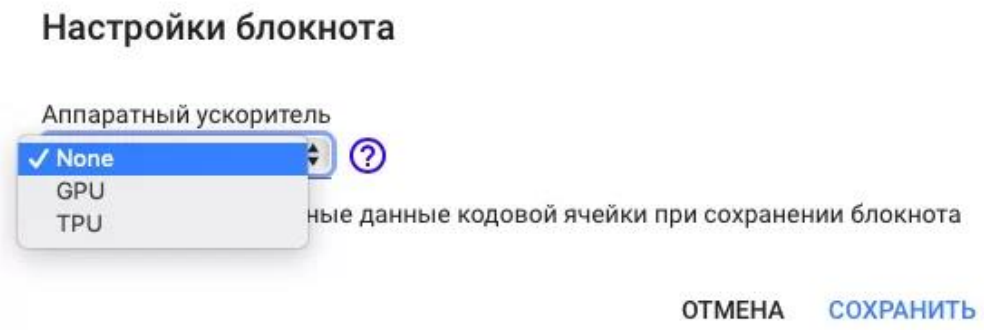
Еще одно достоинство Colab – интеграция с GitHub. Он открывает доступ к любому хранилищу, если ему предоставить профиль на сервисе.



Кроме того, для определенных задач в Google Colab можно выбрать подходящий по мощности процессор. Необходимо просто сменить среду выполнения в нужной вкладке



и уже в настройках блокнота выбрать между GPU и TPU.

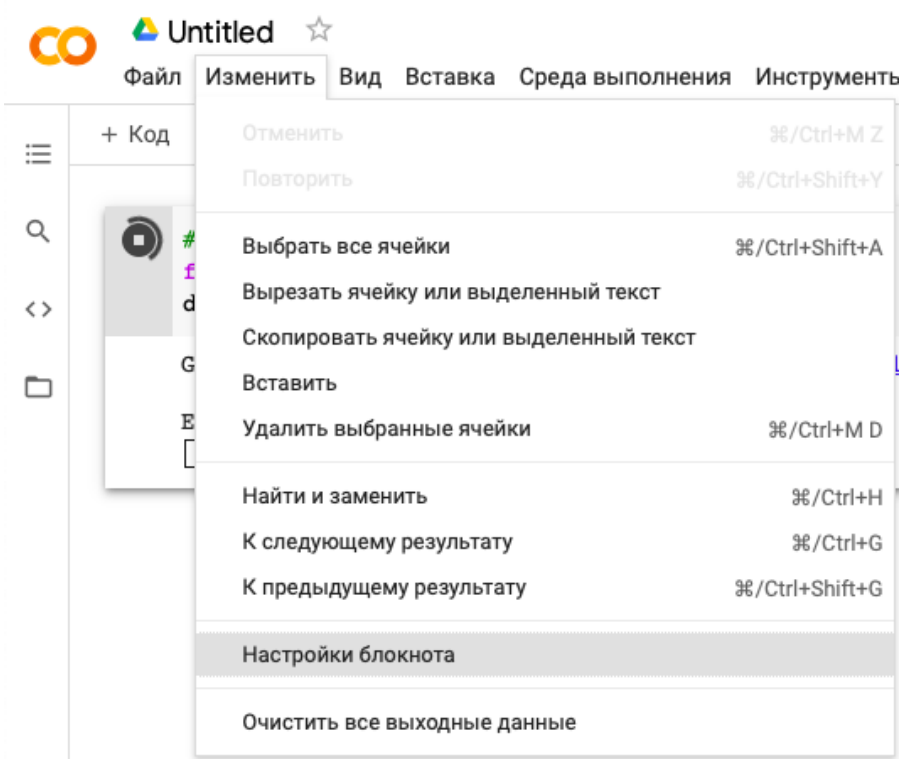


Не стоит работать с мощным процессором, когда не требуется работать с Big Data. Как мы уже говорили, Colab не любит, когда его ресурсы используются нерационально, поэтому любые перегрузки приведут к внезапному вылету из блокнота на неопределенное время.

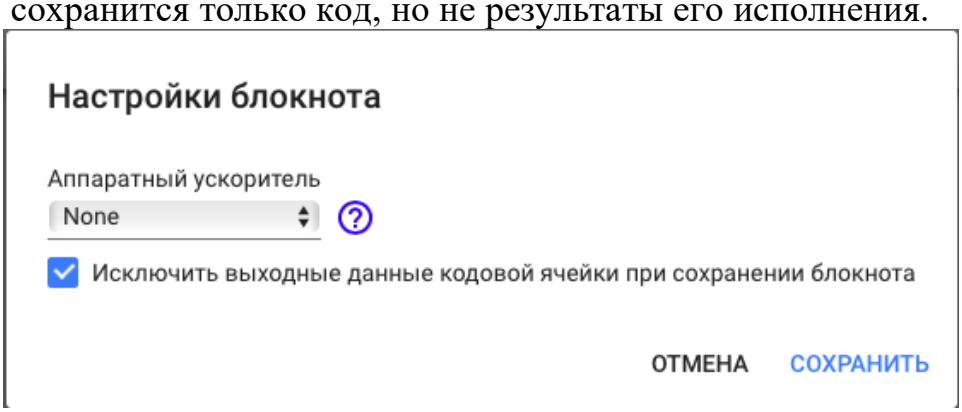
Google Colab максимально упростил все процессы: в нем есть и базовые библиотеки (NumPy, scikit-learn, Pandas), и более сложные (вроде Keras, TensorFlow или PyTorch), не нужно ставить программы и среды самостоятельно, можно просто сразу писать код. Если же базовых библиотек Google Colab недостаточно, всегда можно добавить необходимые с помощью установщика PIP и работать дальше:

```
%pip install emoji
```

В Colab можно делиться файлами с другими, оставлять комментарии, редакторские заметки и в целом делать все, что доступно в тех же Google Документах. Поэтому при общем доступе к блокноту все его содержимое будет доступно другим пользователям (текст, код, комментарии, выходные данные). Последнее можно отключить: нужно выбрать «Настройки блокнота» в меню «Изменить».



В появившемся окне Google Colab поставить галочку «Исключить выходные данные кодовой ячейки при сохранении блокнота», и тогда в блокноте сохранится только код, но не результаты его исполнения.



Вместе с тем открытый доступ к коду и его редактированию – отличная возможность найти интересные разработки по всему миру. У Google есть обширный репозиторий SeedBank, в котором можно исследовать множество блокнотов по Data Science или глубокому обучению, просто кликнув мышкой.

Облачные среды, похожие на Google Colab

– Yandex DataSphere – в отличие от Google Colaboratory это платный блокнот, в котором тарифицируется фактическое время вычислений. При регистрации на пробный период (60 дней) выдается грант в размере 4000 ₽ для резидентов РФ и 50 $ для нерезидентов РФ. Особенности использования сервиса можно изучить в документации.

– Kaggle Kernels – кроме Python, сервис Kaggle поддерживает R, интегрируется с Google Cloud Storage, BigQuery и AutoML. При этом время пользования процессорами – девять часов, на три меньше, чем у GC.

– Azure Notebooks – как и Google Colaboratory, среда тоже поддерживает другие языки (R, F#). Сервисы Microsoft Azure также, как и Яндекса, тарифицируются за фактическое время использования.

– CoCalc – предлагает и бесплатный, и платный (14 $) периоды. В расширенной версии больше памяти и времени простоя, приоритетный доступ к процессорам и техподдержке. Документация.

Лабораторная работа 2 Токенизация

Цель: Ознакомиться с задачами и библиотеками для проведения первичной обработки текстов в NLP – токенизацией, лемматизаций, стеммингом.

Теоретические материалы (см. Приложение 1)

Задания.

Выполнить 4 задания.

а. Изобразить облака слов для научных и новостных текстов (wordcloud, без стоп-слов).
б. Найти частотные n-граммы новостных и научных статей.
в. С помощью TF-IDF вывести ключевые слова для двух новостных текстов.
г. Сравнить качество жанровой классификации на TF-IDF-векторах с использованием двух различных токенизаторов (на словах и на частях слов)

2) Устно ответить на контрольный вопрос

Порядок выполнения работы.

Данные одни для всех заданий:

```
! wget -q
https://www.dropbox.com/s/ug0b4pvuynwj4pe/news_science.zip &&
unzip news_science.zip

import glob
import pandas as pd
from tqdm import tqdm


data = []
for path in glob.glob('./news_science/*'):
  texts = []
  for filename in tqdm(glob.glob(path + '/*.txt')):
    texts.append(open(filename, 'r').read().strip())

  data.append(pd.DataFrame({'text': texts}))
  data[-1]['genre'] = path.split('/')[-1][:3]  # 'new' или 'sci'

data = pd.concat(data)
```

Выходные данные:

```
100%|██████████| 49/49 [00:00<00:00, 5590.58it/s]
100%|██████████| 79/79 [00:00<00:00, 10038.17it/s]
```

```
data.sample(3)
```

Выходные данные:

	text	genre
70	Бывший сирийский член парламента и политически...	new
32	ПРЕДОТВРАЩЕНИЕ УГРОЗЫ ВИРУСНЫХ АТАК В АВТОМАТИ...	sci
37	И.А. Коськин \nМагистрант \nУчебно-научно-ис...	sci

1 Изобразить облака слов для научных и новостных текстов (wordcloud, без стоп-слов)

```
import nltk
nltk.download('stopwords')
```

результат:

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
from nltk.corpus import stopwords
from wordcloud import WordCloud
import matplotlib.pyplot as plt


### Your code goes here ###
### stopwords =
### wordcloud =
############################

plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
```

2 Найти частотные n-граммы новостных и научных статей

```
import string

# Инициализировать токенизатор
### Your code goes here ###
### import ...
### tokenizer = ...
############################

# 1. Токенизировать тексты выбранным токенизатором, приведя их к
нижнему регистру
### Your code goes here ###
### data['text_tokenized'] = data.text.map(lambda row: ...
############################

# 2. Удалить из списка токенов те, которые представлены
пунктуацией и числами

def is_symbol(tok):
    """ Функция определения, состоит ли токен только из знаков
пунктуации """
    ### Your code goes here ###
    ### result =
    ############################
    raise NotImplementedError
    return result

def clean_tokens(tokens):
```

```
    """ Функция возвращает из списка токенов только слова на
русском языке """
    ### Your code goes here ###
    ### tokens =
    ############################
    raise NotImplementedError
    return tokens

data['text_tokenized'] = data.text_tokenized.map(clean_tokens)
```

Создаем списки биграмм для текстов разных жанров

```
import nltk


def create_bigram(tokens):
    bigrams = nltk.bigrams(tokens)
    return list(bigrams)

# Пример для новостных текстов
tokens_news = [token for row in data[data.genre ==
'new'].text_tokenized.values for token in row]
bigrams_list = create_bigram(tokens_news)
bigrams_news = [" ".join(bigram) for bigram in list(bigrams_list)]

# Тестирование
assert bigrams_news[:3] == ['африка в', 'в последние', 'последние
недели'], 'проверьте функции фильтрации списка токенов'
print("Тест пройден")
```

Создаем счетчик частотностей биграмм

```
from collections import Counter
from collections import OrderedDict


def sort_by_frequency(list_of_ngrams, reverse=True):
    """
    Функция сортировки n-грамм по частотности вхождения в список
    Возвращает упорядоченный словарь (OrderedDict)
    """
    ### Your code goes here ###
    # data_with_freq = ...
    # data_sorted_by_freq = ...
    ############################

    return data_sorted_by_freq


bigrams_news_sorted = sort_by_frequency(birgrams_news)

# Тестирование
assert type(result) == OrderedDict, 'Словарь должен быть
упорядоченным'
assert result.get('национальной безопасности') == 4, 'Проверьте
счетчик биграмм'
```

```
print('Тесты пройдены')
```
В качестве результата выведите топ-20 частотных N-грамм.

https://colab.research.google.com/drive/14b4aqm10MF361n-cUox1o6T4o-8s47HN?pli=1#scrollTo=gij65BYqwn3k

Контрольные вопросы

1 Что такое регулярные выражения?
2 Для чего они применяются?
3 Соответствие символов
4 Метасимволы, их значение и применение
5 Использование регулярных выражений
6 Компиляция регулярных выражений
7 Функции для работы с регулярными выражениями.
8 Какие существуют операции над строками в языке Python?
9 Какие существуют методы обработки строк?
10 Преимущества и недостатки использования для обработки строк методов и регулярных выражений.

Требования к оформлению отчета.

Отчет по ЛР состоит из

а) Титульный лист: название ЛР, ФИО студента и номер группы, № варианта.

б)Текст задания;

в) Исходный текст программы;

г) Скрины выполнения;

д) Выводы.

Критерии оценивания.

За решение каждой задачи вы можете получить до 2 баллов: задание выполнено полностью и правильно -2 балла; имеются незначительные ошибки - 1 балл; приложение не работоспособно – 0 баллов;

Ответ на контрольный вопрос должен продемонстрировать понимание механизмов NLP, за ответ вы можете оценку: ответ полный и правильный -1 балл; ответ не удовлетворительный – 0 баллов.

Итого, максимальная оценка - 9 баллов

Внимание! Полученная оценка автматически снижается на 2% за каждую полную неделю задержки сдачи отчета по работе, но не более, чем на 40%

Библиографический список

1. Маккинни, У. Python и анализ данных. Первичная обработка данных с применением pandas, NumPy и Jupiter : справочник / У. Маккинни ; перевод с английского А. А. Слинкина. – 3-е изд. – Москва : ДМК Пресс, 2023. – 536 с. – ISBN 978-5-93700-174-0. – Текст : электронный // Лань : электронно-библиотечная система. – URL: https://e.lanbook.com/book/348086 (дата обращения: 26.08.2023). – Режим доступа: для авториз. пользователей

2. Руководство по языку программирования Python. [Электронный ресурс].– URL: https://metanit.com/python/tutorial. – Доступ свободный

3. Методы интеллектуального анализа текстов [Электронный ресурс]. – URL: https://stud-sci.rudn.ru/course/view.php?id=264/ – Гостевой доступ (Зайти гостем)

Приложение 1

Основы Natural Language Processing для текста

Автор оригинала: Ventsislav Yordanov
Материал с сайта: https://habr.com/ru/companies/Voximplant/articles/446738/

Что такое Natural Language Processing?

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

Python-библиотека NLTK

NLTK (Natural Language Toolkit) – ведущая платформа для создания NLP-программ на Python. У нее есть легкие в использовании интерфейсы для многих языковых корпусов, а также библиотеки для обработки текстов для классификации, токенизации, стемминга, разметки, фильтрации и семантических рассуждений. Ну и еще это бесплатный опенсорсный проект, который развивается с помощью коммьюнити.

Мы будем использовать этот инструмент, чтобы показать основы NLP. Для всех последующих примеров я предполагаю, что NLTK уже импортирован; сделать это можно командой `import nltk`

Основы NLP для текста

В этой статье мы рассмотрим темы:

1. Токенизация по предложениям.
2. Токенизация по словам.
3. Лемматизация и стемминг текста.
4. Стоп-слова.
5. Регулярные выражения.
6. Мешок слов.
7. TF-IDF.

1. Токенизация по предложениям

Токенизация (иногда – сегментация) по предложениям – это процесс разделения письменного языка на предложения-компоненты. Идея выглядит довольно простой. В английском и некоторых других языках мы можем вычленять предложение каждый раз, когда находим определенный знак пунктуации – точку.

Но даже в английском эта задача нетривиальна, так как точка используется и в сокращениях. Таблица сокращений может сильно помочь во время обработки текста, чтобы избежать неверной расстановки границ предложений.

В большинстве случаев для этого используются библиотеки, так что можете особо не переживать о деталях реализации.

Пример:

Возьмем небольшой текст про настольную игру нарды:

```
Backgammon is one of the oldest known board games. Its history can be traced back nearly 5,000 years to archeological discoveries in the Middle East. It is a two player game where each player has fifteen checkers which move between twenty-four points according to the roll of two dice.
```

Чтобы сделать токенизацию предложений с помощью NLTK, можно воспользоваться методом `nltk.sent_tokenize`

На выходе мы получим 3 отдельных предложения:

```
Backgammon is one of the oldest known board games.

Its history can be traced back nearly 5,000 years to archeological discoveries in the Middle East.

It is a two player game where each player has fifteen checkers which move between twenty-four points according to the roll of two dice.
```

2. Токенизация по словам

Токенизация (иногда – сегментация) по словам – это процесс разделения предложений на слова-компоненты. В английском и многих других языках, использующих ту или иную версию латинского алфавита, пробел – это неплохой разделитель слов.

Тем не менее, могут возникнуть проблемы, если мы будем использовать только пробел – в английском составные существительные пишутся по-разному и иногда через пробел. И тут вновь нам помогают библиотеки.

Пример:

Давайте возьмем предложения из предыдущего примера и применим к ним метод `nltk.word_tokenize`

Вывод:

```
['Backgammon', 'is', 'one', 'of', 'the', 'oldest', 'known', 'board', 'games', '.']

['Its', 'history', 'can', 'be', 'traced', 'back', 'nearly', '5,000', 'years', 'to', 'archeological', 'discoveries', 'in', 'the', 'Middle', 'East', '.']

['It', 'is', 'a', 'two', 'player', 'game', 'where', 'each', 'player', 'has', 'fifteen', 'checkers', 'which', 'move', 'between', 'twenty-four', 'points', 'according', 'to', 'the', 'roll', 'of', 'two', 'dice', '.']
```

3. Лемматизация и стемминг текста

Обычно тексты содержат разные грамматические формы одного и того же слова, а также могут встречаться однокоренные слова. Лемматизация и стемминг преследуют цель привести все встречающиеся словоформы к одной, нормальной словарной форме.

Примеры:

Приведение разных словоформ к одной:

```
dog, dogs, dog’s, dogs’ => dog
```

То же самое, но уже применительно к целому предложению: the boy’s dogs are different sizes => the boy dog be differ size

Лемматизация и стемминг – это частные случаи нормализации и они отличаются.

Стемминг – это грубый эвристический процесс, который отрезает «лишнее» от корня слов, часто это приводит к потере словообразовательных суффиксов.

Лемматизация – это более тонкий процесс, который использует словарь и морфологический анализ, чтобы в итоге привести слово к его канонической форме – лемме.

Отличие в том, что стеммер (конкретная реализация алгоритма стемминга – прим.переводчика) действует без знания контекста и, соответственно, не понимает разницу между словами, которые имеют разный смысл в зависимости от части речи. Однако у стеммеров есть и свои преимущества: их проще внедрить и они работают быстрее. Плюс, более низкая «аккуратность» может не иметь значения в некоторых случаях.

Примеры:

1. Слово good – это лемма для слова better. Стеммер не увидит эту связь, так как здесь нужно сверяться со словарем.
2. Слово play – это базовая форма слова playing. Тут справятся и стемминг, и лемматизация.
3. Слово meeting может быть как нормальной формой существительного, так и формой глагола to meet, в зависимости от контекста. В отличие от стемминга, лемматизация попробует выбрать правильную лемму, опираясь на контекст.

Теперь, когда мы знаем, в чем разница, давайте рассмотрим пример:

Вывод:

```
Stemmer: seen
Lemmatizer: see

Stemmer: drove
Lemmatizer: drive
```

4. Стоп-слова

Стоп-слова – это слова, которые выкидываются из текста до/после обработки текста. Когда мы применяем машинное обучение к текстам, такие слова могут добавить много шума, поэтому необходимо избавляться от нерелевантных слов.

Стоп-слова это обычно понимают артикли, междометия, союзы и т.д., которые не несут смысловой нагрузки. При этом надо понимать, что не существует универсального списка стоп-слов, все зависит от конкретного случая.

В NLTK есть предустановленный список стоп-слов. Перед первым использованием вам понадобится его скачать: `nltk.download("stopwords")`. После скачивания можно импортировать пакет stopwords и посмотреть на сами слова:

Вывод:

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

Рассмотрим, как можно убрать стоп-слова из предложения:

Вывод:

```
['Backgammon', 'one', 'oldest', 'known', 'board', 'games', '.']
```

Если вы не знакомы с list comprehensions, то можно узнать побольше здесь. Вот другой способ добиться того же результата:

Тем не менее, помните, что list comprehensions быстрее, так как оптимизированы – интерпретатор выявляет предиктивный паттерн во время цикла.

Вы можете спросить, почему мы конвертировали список во множество. Множество это абстрактный тип данных, который может хранить уникальные значения, в неопределенном порядке. Поиск по множеству гораздо быстрее поиска по списку. Для небольшого количества слов это не имеет значения, но

если речь про большое количество слов, то строго рекомендуется использовать множества. Если хотите узнать чуть больше про время выполнения разных операций, посмотрите на эту чудесную шпаргалку.

5. Регулярные выражения.

Регулярное выражение (регулярка, regexp, regex) – это последовательность символов, которая определяет шаблон поиска. Например:

- . – любой символ, кроме перевода строки;
- \w – один символ;
- \d – одна цифра;
- \s – один пробел;
- \W – один НЕсимвол;
- \D – одна НЕцифра;
- \S – один НЕпробел;
- [abc] – находит любой из указанных символов match any of a, b, or c;
- [^abc] – находит любой символ, кроме указанных;
- [a-g] – находит символ в промежутке от a до g.

Выдержка из документации Python:

Регулярные выражение используют обратный слеш (\) для обозначения специальных форм или чтобы разрешить использование спецсимволов. Это противоречит использованию обратного слеша в Python: например, чтобы буквально обозначить обратный слеш, необходимо написать '\\\\' в качестве шаблона для поиска, потому что регулярное выражение должно выглядеть как \\, где каждый обратный слеш должен быть экранирован.

Решение – использовать нотацию raw string для шаблонов поиска; обратные слеши не будут особым образом обрабатываться, если использованы с префиксом ‘r’. Таким образом, r”\n” – это строка с двумя символами (‘\’ и ‘n’), а “\n” – строка с одним символом (перевод строки).

Мы можем использовать регулярки для дополнительного фильтрования нашего текста. Например, можно убрать все символы, которые не являются словами. Во многих случаях пунктуация не нужна и ее легко убрать с помощью регулярок.

Модуль **re** в Python представляет операции с регулярными выражениями. Мы можем использовать функцию **re.sub**, чтобы заменить все, что подходит под шаблон поиска, на указанную строку. Вот так можно заменить все НЕслова на пробелы:

Вывод:

```
'The development of snowboarding was inspired by skateboarding  sledding  surfing and skiing '
```

Регулярки – это мощный инструмент, с его помощью можно создавать гораздо более сложные шаблоны. Если вы хотите узнать больше о регулярных выражениях, то могу порекомендовать эти 2 веб-приложения: regex, regex101.

6. Мешок слов

Алгоритмы машинного обучения не могут напрямую работать с сырым текстом, поэтому необходимо конвертировать текст в наборы цифр (векторы). Это называется извлечением признаков.

Мешок слов – это популярная и простая техника извлечения признаков, используемая при работе с текстом. Она описывает вхождения каждого слова в текст.

Чтобы использовать модель, нам нужно:

1. Определить словарь известных слов (токенов).
2. Выбрать степень присутствия известных слов.

Любая информация о порядке или структуре слов игнорируется. Вот почему это называется МЕШКОМ слов. Эта модель пытается понять, встречается ли знакомое слово в документе, но не знает, где именно оно встречается.

Интуиция подсказывает, что **схожие документы** имеют **схожее содержимое**. Также, благодаря содержимому, мы можем узнать кое-что о смысле документа.

Пример:

Рассмотрим шаги создания этой модели. Мы используем только 4 предложения, чтобы понять, как работает модель. В реальной жизни вы столкнетесь с бОльшими объемами данных.

1. Загружаем данные

Представим, что это наши данные и мы хотим загрузить их в виде массива:

```
I like this movie, it's funny.
I hate this movie.
This was awesome! I like it.
Nice one. I love it.
```

Для этого достаточно прочитать файл и разделить по строкам:

Вывод:

```
["I like this movie, it's funny.", 'I hate this movie.', 'This was awesome! I like it.', 'Nice one. I love it.']
```

2. Определяем словарь

Соберем все уникальные слова из 4 загруженных предложений, игнорируя регистр, пунктуацию и односимвольные токены. Это и будет наш словарь (известные слова). Для создания словаря можно использовать класс CountVectorizer из библиотеки sklearn. Переходим к следующему шагу.

3. Создаем векторы документа

Далее, мы должны оценить слова в документе. На этом шаге наша цель – превратить сырой текст в набор цифр. После этого, мы используем эти наборы как входные данные для модели машинного обучения. Простейший метод скоринга – это отметить наличие слов, то есть ставить 1, если есть слово и 0 при его отсутствии.

Теперь мы можем создать мешок слов используя вышеупомянутый класс CountVectorizer.

Вывод:

	awesome	funny	hate	it	like	love	movie	nice	one	this	was
0	0	1	0	1	1	0	1	0	0	1	0
1	0	0	1	0	0	0	1	0	0	1	0
2	1	0	0	1	1	0	0	0	0	1	1
3	0	0	0	1	0	1	0	1	1	0	0

Это наши предложения в векторнлй форме. Теперь мы видим, как работает модель «мешок слов».

```
I like this movie, it's funny.
I hate this movie.
This was awesome! I like it.
Nice one. I love it.
```

Еще пару слов про мешок слов

Сложность этой модели в том, как определить словарь и как подсчитать вхождение слов.

Когда размер словаря увеличивается, вектор документа тоже растет. В примере выше, длина вектора равна количеству известных слов. В некоторых случаях, у нас может быть неимоверно большой объем данных и тогда вектор может состоять из тысяч или миллионов элементов. Более того, каждый документ может содержать лишь малую часть слов из словаря.

Как следствие, в векторном представлении будет много нулей. Векторы с большим количеством нулей называются разреженным векторами (sparse vectors), они требуют больше памяти и вычислительных ресурсов. Однако мы можем уменьшить количество известных слов, когда используем эту модель, чтобы снизить требования к вычислительным ресурсам. Для этого можно использовать те же техники, что мы уже рассматривали до создания мешка слов:

- игнорирование регистра слов;
- игнорирование пунктуации;
- выкидывание стоп-слов;
- приведение слов к их базовым формам (лемматизация и стемминг);
- исправление неправильно написанных слов.

Другой, более сложный способ создания словаря – использовать сгруппированные слова. Это изменит размер словаря и даст мешку слов больше деталей о документе. Такой подход называется «N-грамма».

N-грамма это последовательность каких-либо сущностей (слов, букв, чисел, цифр и т.д.). В контексте языковых корпусов, под N-граммой обычно понимают последовательность слов. Юниграмма это одно слово, биграмма это последовательность двух слов, триграмма – три слова и так далее. Цифра N

обозначает, сколько сгруппированных слов входит в N-грамму. В модель попадают не все возможные N-граммы, а только те, что фигурируют в корпусе.

Пример:

Рассмотрим такое предложение:

```
The office building is open today
```

Вот его биграммы:

- the office
- office building
- building is
- is open
- open today

Как видно, мешок биграмм – это более действенный подход, чем мешок слов.

Оценка (скоринг) слов

Когда создан словарь, следует оценить наличие слов. Мы уже рассматривали простой, бинарный подход (1 – есть слово, 0 – нет слова). Есть и другие методы:

1. Количество. Подсчитывается, сколько раз каждое слово встречается в документе.
2. Частотность. Подсчитывается, как часто каждое слово встречается в тексте (по отношению к общему количеству слов).

7. TF-IDF

У частотного скоринга есть проблема: слова с наибольшей частотностью имеют, соответственно, наибольшую оценку. В этих словах может быть не так много информационного выигрыша для модели, как в менее частых словах. Один из способов исправить ситуацию – понижать оценку слова, которое часто встречается **во всех схожих документах**. Это называется TF-IDF.

TF-IDF (сокращение от term frequency — inverse document frequency) – это статистическая мера для оценки важности слова в документе, который является частью коллекции или корпуса.

Скоринг по TF-IDF растет пропорционально частоте появления слова в документе, но это компенсируется количеством документов, содержащих это слово.

Формула скоринга для слова X в документе Y:

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

Формула TF-IDF. Источник: filotechnologia.blogspot.com/2014/01/a-simple-java-class-for-tfidf-scoring.html

TF (term frequency — частота слова) – отношение числа вхождений слова к общему числу слов документа.

$$TF(\boldsymbol{term}) = \frac{Number\ of\ times\ \boldsymbol{term}\ appears\ in\ a\ document}{Total\ number\ of\ items\ in\ the\ document}$$

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

$$IDF(\boldsymbol{term}) = \log\left(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ \boldsymbol{term}\ in\ it}\right)$$

В итоге, вычислить TF-IDF для слова **term** можно так:

$$TFIDF(\boldsymbol{term}) = TF(\boldsymbol{term}) * \mathrm{IDF}(\mathbf{term})$$

Пример:

Можно использовать класс **TfidfVectorizer** из библиотеки sklearn, чтобы вычислить TF-IDF. Давайте проделаем это с теми же сообщениями, что мы использовали в примере с мешком слов.

```
I like this movie, it's funny.
I hate this movie.
This was awesome! I like it.
Nice one. I love it.
```

Вывод:

	awesome	funny	hate	it	like	love	movie	nice	one	this	was
0	0.000000	0.571848	0.000000	0.365003	0.450852	0.000000	0.450852	0.000000	0.000000	0.365003	0.000000
1	0.000000	0.000000	0.702035	0.000000	0.000000	0.000000	0.553492	0.000000	0.000000	0.448100	0.000000
2	0.539445	0.000000	0.000000	0.344321	0.425305	0.000000	0.000000	0.000000	0.000000	0.344321	0.539445
3	0.000000	0.000000	0.000000	0.345783	0.000000	0.541736	0.000000	0.541736	0.541736	0.000000	0.000000

Заключение

В этой статье были разобраны основы NLP для текста, а именно:

- NLP позволяет применять алгоритмы машинного обучения для текста и речи;

- NLTK (Natural Language Toolkit) – ведущая платформа для создания NLP-программ на Python;
- токенизация по предложениям – это процесс разделения письменного языка на предложения-компоненты;
- токенизация по словам – это процесс разделения предложений на слова-компоненты;
- лемматизация и стемминг преследуют цель привести все встречающиеся словоформы к одной, нормальной словарной форме;
- стоп-слова – это слова, которые выкидываются из текста до/после обработки текста;
- регулярное выражение (регулярка, regexp, regex) – это последовательность символов, которая определяет шаблон поиска;
- мешок слов – это популярная и простая техника извлечения признаков, используемая при работе с текстом. Она описывает вхождения каждого слова в текст.

Отлично! Теперь, зная основы выделения признаков, вы можете использовать признаки как входные данные для алгоритмов машинного обучения.

Лабораторнаяработа 3 Коррекция опечаток

Цель:Ознакомиться с методами библиотеками **Python** для исправления опечаток. Исследовать эти методы

Теоретическиематериалы (см.Приложение1)

Материалы для самостоятельного изучения

https://web.stanford.edu/~jurafsky/slp3/slides/2_EditDistance_Jan_08_2020.pdf

Задания.

Выполнить 4 задания.

1) Измерить время работы трех методов из готовых библиотек.
2) Написать функцию Дамерау-Левенштейна, измерить время работы.
3) Написать функцию генерации опечаток в соответствии с положением знаков на клавиатуре.
4) Написать функцию сравнения векторов, составленных из частотностей входящих в слово букв (CountVectorizer).

2)Устно ответить на контрольный вопрос

Порядок выполнения работы.

Данные одни для всех заданий:

```
examples = ["вот в инете откапал такую инеерсную статейку
предлагаю вашему внимани",
            "может и в правду лутше тебе молчать чем пытаться
сказать",
            "утром мы сидели как сычи а потом каааак начали
работать"]
```

Контрольныевопросы

1 Что такое TextMining? Основные задачи.
2 Что дает классификация по топологии естественных языков?
3 Что такое грамматическое значение слова?
4 Виды образования формы слова.
5 Библиотеки Pithon по коррекции опечаток для русского языка
6 Базовые способы коррекции
7 Что такое расстояние Левенштейна?
8 Какие вы знаете методы увеличения быстродействия приложений для исправления опечаток?

Требования к оформлению отчета.

Отчет по ЛР состоит из

а) Титульный лист:название ЛР, ФИО студента и номер группы, № варианта.

б)Текст задания;

в) Исходный текст программы;

г) Скрины выполнения;
д) Выводы.

Критерии оценивания.

За решение каждой задачи вы можете получить до 2 баллов: задание выполнено полностью и правильно -2 балла; имеются незначительные ошибки - 1 балл; приложение не работоспособно – 0 баллов;

Ответ на контрольный вопрос должен продемонстрировать понимание механизмов NLP, за ответ вы можете оценку: ответ полный и правильный -1 балл; ответ не удовлетворительный – 0 баллов.

Итого, максимальная оценка – 9 баллов

Внимание! Полученная оценка автматически снижается на 2% за каждую полную неделю задержки сдачи отчета по работе, но не более, чем на 40%

Библиографическийсписок

1. Маккинни,У.Python и анализ данных. Первичная обработка данных с применением pandas, NumPy и Jupiter:справочник/ У.Маккинни;. – Москва: ДМКПресс, 2023. – 536с. – ISBN978-5-93700-174-0. – Текст: электронный //Лань:э лектронно-библиотечная система. – URL:https://e.lanbook.com/book/348086(датаобращения:26.08.2023). – Режимдоступа:дляавториз.пользователей

2. Руководство по языку программирования Python. [Электронный ресурс].– URL: https://metanit.com/python/tutorial. – Доступ свободный

3. Методы интеллектуального анализа текстов [Электронный ресурс]. – URL: https://stud-sci.rudn.ru/course/view.php?id=264/ – Гостевой доступ (Зайти гостем)

Приложение 1

Исправление орфографии в Python с помощью TextBlob

Современные средства проверки правописания способны обрабатывать морфологию и использовать статистику для улучшения предложений.

Python предлагает множество модулей для этих целей, что делает написание простой проверки орфографии легким 20-минутным испытанием.

Одной из этих библиотек является TextBlob, которая используется для обработки естественного языка и предоставляет интуитивно понятный API для работы.

В этой статье мы рассмотрим, как реализовать исправление орфографии в Python с помощью `TextBlob`.

Установка

Во-первых, нам нужно установить TextBlob, поскольку он не предустановлен. Откройте консоль и установите его с помощью pip:

```
pip install textblob
```

Это должно установить все, что нам нужно для этого проекта. По окончании установки вывод консоли должен включать что-то вроде:

```
Successfully installed click-7.1.2 joblib-0.17.0 nltk-3.5 regex-
2020.11.13 textblob-0.15.3
```

TextBlob построен на основе NLTK, поэтому он также поставляется с установкой.

Функция correct()

Самый простой способ исправить вводимый текст - использовать метод `correct()`. В качестве примера мы будем использовать абзац из книги Чарльза Дарвина «О происхождении видов», которая является частью общественного достояния и упакована в файл с именем `text.txt`.

Кроме того, мы добавим несколько умышленных орфографических ошибок:

```
As far as I am abl to judg, after long attnding to the sbject, the
condiions of lfe apear to act in two ways—directly on the whle
organsaton or on certin parts alne and indirectly by afcting the
reproducte sstem. Wit respct to te dirct action, we mst bea in mid
tht in every cse, as Profesor Weismann hs latly insistd, and as I
have inidently shwn in my wrk on "Variatin undr Domesticcation,"
thcere arae two factrs: namly, the natre of the orgnism and the
natture of the condiions. The frmer sems to be much th mre
importannt; foor nealy siimilar variations sometimes aris under,
as far as we cn juddge, disimilar conditios; annd, on te oter
hannd, disssimilar variatioons arise undder conditions which
aappear to be nnearly uniiform. The efffects on tthe offspring
arre ieither definnite or in definite. They maay be considdered as
definnite whhen allc or neearly all thhe ofefspring off
```

```
inadividuals exnposed tco ceertain conditionas duriing seveal
ggenerations aree moodified in te saame maner.
```

Это полный орфографических ошибок текст, почти в каждом слове. Давайте напишем простой скрипт, используя TextBlob, чтобы исправить эти ошибки и распечатать их обратно в консоль:

```
from textblob import TextBlob

with open("text.txt", "r") as f:          # Opening the test file
with the intention to read
    text = f.read()                       # Reading the file
    textBlb = TextBlob(text)              # Making our first
textblob
    textCorrected = textBlb.correct()     # Correcting the text
    print(textCorrected)
```

Если вы раньше работали с TextBlob, этот алгоритм будет вам знаком. Мы прочитали файл и его содержимое и создали экземпляр `TextBlob`, передав содержимое конструктору.

Затем мы запускаем функцию `correct()` в этом экземпляре для исправления орфографии.

После запуска приведенного выше сценария вы должны получить примерно такой результат:

```
Is far as I am all to judge, after long attending to the subject,
the conditions of life appear to act in two ways—directly on the
while organisation or on certain parts alone and indirectly by
acting the reproduce system. It respect to te direct action, we
must be in mid the in every case, as Professor Weismann he lately
insisted, and as I have evidently shown in my work on "Variation
under Domesticcation," there are two facts: namely, the nature of
the organism and the nature of the conditions. The former seems to
be much th are important; for nearly similar variations sometimes
arms under, as far as we in judge, similar condition; and, on te
other hand, disssimilar variations arise under conditions which
appear to be nearly uniform. The effects on the offspring are
either definite or in definite. They may be considered as definite
when all or nearly all the offspring off individuals exposed to
certain conditions during several generations are modified in te
same manner.
```

Насколько верна коррекция орфографии TextBlob?

Как видим, в тексте все еще есть орфографические ошибки. Слова вроде `"abl"` должны были быть `"able"`, а не `"all"`. Хотя даже с ними все равно лучше оригинала.

Теперь возникает вопрос, насколько это лучше?

Следующий фрагмент кода представляет собой простой сценарий, который проверяет, насколько хорошо TextBlob исправляет ошибки, на основе этого примера:

```
from textblob import TextBlob
```

```
# A function that compares two texts and returns
# the number of matches and differences
def compare(text1, text2):
    l1 = text1.split()
    l2 = text2.split()
    good = 0
    bad = 0
    for i in range(0, len(l1)):
        if l1[i] != l2[i]:
            bad += 1
        else:
            good += 1
    return (good, bad)

# Helper function to calculate the percentage of misspelled words
def percentageOfBad(x):
    return (x[1] / (x[0] + x[1])) * 100
```

Теперь, используя эти две функции, давайте проведем быстрый анализ:

```
with open("test.txt", "r") as f1: # test.txt contains the same
typo-filled text from the last example
    t1 = f1.read()

with open("original.txt", "r") as f2: # original.txt contains the
text from the actual book
    t2 = f2.read()

t3 = TextBlob(t1).correct()

mistakesCompOriginal = compare(t1, t2)
originalCompCorrected = compare(t2, t3)
mistakesCompCorrected = compare(t1, t3)

print("Mistakes compared to original ", mistakesCompOriginal)
print("Original compared to corrected ", originalCompCorrected)
print("Mistakes compared to corrected ", mistakesCompCorrected,
"\n")

print("Percentage of mistakes in the test: ",
percentageOfBad(mistakesCompOriginal), "%")
print("Percentage of mistakes in the corrected: ",
percentageOfBad(originalCompCorrected), "%")
print("Percentage of fixed mistakes: ",
percentageOfBad(mistakesCompCorrected), "%", "\n")
```

Запустив его, вы распечатаете:

```
Mistakes compared to original  (126, 194)
Original compared to corrected  (269, 51)
Mistakes compared to corrected  (145, 175)

Percentage of mistakes in the test:  60.62499999999999 %
Percentage of mistakes in the corrected:  15.937499999999998 %
```

```
Percentage of fixed mistakes:  54.6875 %
```

Как мы видим, методу `correct` удалось уменьшить процент орфографических ошибок с 60,6% до 15,9%, что довольно неплохо, однако есть небольшая загвоздка. Он исправил 54,7% слов, так почему все еще остается 15,9% ошибок?

Ответ - чрезмерное исправление. Иногда он может изменить слово, которое написано правильно, например, первое слово в нашем примере текста, где `"As"` было исправлено `"Is"`. В других случаях ему просто не хватает информации о слове и контексте, чтобы сказать, какое слово пользователь намеревался ввести, поэтому он догадывается, что его следует заменить `"whl"` на `"while"` вместо `"whole"`.

Не существует идеального корректора орфографии, потому что большая часть разговорной речи зависит от контекста, так что имейте это в виду. В большинстве случаев ошибок гораздо меньше, чем в нашем примере, поэтому TextBlob должен работать достаточно хорошо для обычного пользователя.

Обучающий TextBlob с настраиваемыми наборами данных

Что, если вы хотите проверить орфографию на другом языке, который не поддерживается TextBlob из коробки? Или, может быть, вы хотите быть немного точнее? Что ж, может быть способ добиться этого. Все сводится к тому, как работает проверка орфографии в TextBlob.

TextBlob использует статистику использования слов на английском языке, чтобы делать разумные предложения по поводу того, какие слова следует исправить. Он хранит эту статистику в файле с именем en-spelling.txt, но также позволяет вам создать свой собственный файл статистики использования слов.

Попробуем сделать такой для нашего примера Дарвина. Мы будем использовать все слова из «Происхождения видов» для обучения. Вы можете использовать любой текст, просто убедитесь, что в нем достаточно слов, имеющих отношение к тексту, который вы хотите исправить.

В нашем случае остальная часть книги предоставит отличный контекст и дополнительную информацию, которая потребуется TextBlob для более точного исправления.

Перепишем скрипт:

```
from textblob.en import Spelling
import re

textToLower = ""

with open("originOfSpecies.txt","r") as f1:                # Open our
source file
      text = f1.read()                                          #
Read the file
      textToLower = text.lower()                               #
Lower all the capital letters
```

```
words = re.findall("[a-z]+", textToLower)                    # Find
all the words and place them into a list
oneString = " ".join(words)                                    # Join
them into one string

pathToFile = "train.txt"                                       # The
path we want to store our stats file at
spelling = Spelling(path = pathToFile)                        # Connect
the path to the Spelling object
spelling.train(oneString, pathToFile)                         # Train
```

Если мы заглянем в файл `train.txt`, то увидим:

```
a 3389
abdomen 3
aberrant 9
aberration 5
abhorrent 1
abilities 1
ability 4
abjectly 1
able 54
ably 5
abnormal 17
abnormally 2
abodes 2
...
```

Это означает, что слово `"a"` отображается как слово 3389 раз, а `"ably"`только 5 раз. Чтобы проверить эту обученную модель, мы будем использовать `suggest(text)` вместо `correct(text)`, который представляет собой список кортежей доверия слов. Первым элементом в списке будет слово, в котором он уверен, поэтому мы можем получить к нему доступ через `suggest(text)[0][0]`.

Обратите внимание, что это может быть медленнее, поэтому проверяйте орфографию слово за словом, так как сброс огромных объемов данных может привести к сбою:

```
from textblob.en import Spelling
from textblob import TextBlob

pathToFile = "train.txt"
spelling = Spelling(path = pathToFile)
text = " "

with open("test.txt", "r") as f:
      text = f.read()

words = text.split()
corrected = " "
for i in words :
     corrected = corrected +" "+ spelling.suggest(i)[0][0] # Spell
checking word by word
```

```
print(corrected)
```

И теперь это приведет к:

```
As far as I am all to judge after long attending to the subject
the conditions of life appear to act in two ways—directly on the
whole organisation or on certain parts alone and indirectly by
acting the reproduce system It respect to the direct action we
most be in mid the in every case as Professor Weismann as lately
insisted and as I have incidently shown in my work on "Variatin
under Domesticcation," there are two facts namely the nature of
the organism and the nature of the conditions The former seems to
be much th are important for nearly similar variations sometimes
arise under as far as we in judge dissimilar conditions and on
the other hand dissimilar variations arise under conditions which
appear to be nearly uniform The effects on the offspring are
either definite or in definite They may be considered as definite
when all or nearly all the offspring off individuals exposed to
certain conditions during several generations are modified in the
same manner.
```

Это исправляет примерно 2 из 3 слов с ошибками, что довольно хорошо, учитывая запуск без особого контекста.

Источник:

- Spelling Correction in Python with TextBlob
- Исправление орфографии в Python с помощью TextBlob
- #Python

Лабораторная работа 4 Морфологический анализ

Цель: Ознакомиться с методами библиотеками **Python** для морфологического анализа. Исследовать эти методы

Теоретические материалы (см.Приложение1)

Материалы для самостоятельного изучения

1. Syntactic analysis (5LN455). Transition-based dependency parsing. – https://cl.lingfil.uu.se/~sara/kurser/5LN455-2017/slides/5LN455-F9.pdf
2. Speech and Language Processing (3rd ed. draft).Dependency Parsing. – https://web.stanford.edu/~jurafsky/slp3/14.pdf
3. Боярский К.К. Введение в компьютерную лингвистику. Раздел 3.2. – https://books.ifmo.ru/file/pdf/1470.pdf
4. Библиотеки морфологического анализа | Обработка текста на русском языке. – https://www.youtube.com/watch?v=pF1RGPYyNHQ

Задания.

Выполнить 4 задания.

а. Маскировать все имена собственные в электронном письме.
б. Заменить с помощьюpymorphy выбранный объект в статье википедии на другой объект (со склонением).
в. Запустить два разных морфологических анализатора на одном тексте, привести три токена, для которых автоматически определились разные леммы; расшифровать предсказанные для них морфологические теги.
г. Визуализировать на двумерном графике 100 word2vec-токенов (нужно указывать часть речи) из новостных текстов с помощью алгоритмat-SNE.

2)Устно ответить на контрольный вопрос

Порядок выполнения работы.

4.1 Маскировать все имена собственные в электронном письме

```
email = """
Уважаемая Эльвира Геннадьевна,

Как шеф-повар, я должен выразить свою обеспокоенность слухами о появлении
плесени вблизи некоторых кухонь в корпусе 13 по адресу: Льва Толстого, 42.
Обращаю ваше внимание на то, что любые продукты, обрабатываемые,
приготавливаемые или потребляемые вблизи плесени, могут быть заражены и
небезопасны для употребления.

Если эти сообщения найдут подтверждения, мне придется закрыть несколько
столовых. Возможно распределение нагрузки на кухни в корпусах:
№9 площадь Гагарина, 99
№10 улица Южная, 10

С уважением,
```

```
Алексей Мартынов.
"""
```

4.2 Заменить pymorphy выбранный объект в статье википедии на другой объект (со склонением)

```
# Пример входных данных
orig_wikitext = """Стрекозы (лат. Odonáta) — отряд древних летающих
насекомых, насчитывающий в мировой фауне свыше 6650 видов. Это относительно
крупные насекомые, с подвижной головой, большими глазами, короткими
щетинковидными усиками, удлинённым стройным брюшком и четырьмя прозрачными
крыльями с густой сетью жилок. Стрекозы — активные специализированные
хищники, которые питаются насекомыми, пойманными на лету. Представители
отряда широко распространены по миру, встречаясь на всех материках, исключая
Антарктиду.
Все представители отряда ведут амфибионтный образ жизни — яйца и личинки
развиваются в водной среде, а имаго (взрослые) обитают на суше, освоив
воздушную среду и став прекрасными летунами. Взрослые стрекозы не
ограничиваются в выборе места обитания одними только берегами водоёмов и
могут улетать от них на значительные расстояния, встречаясь на лугах,
опушках лесов и даже в населённых пунктах. Развитие с неполным превращением:
имеются стадии яйца, личинки и имаго. Личинки (их называют также нимфами или
наядами) развиваются в водоёмах различных типов — главным образом в стоячих
озёрах, прудах и старицах рек, а также в ручьях, реках и болотистых
водоёмах, вплоть до созданных человеком прудов и канав с водой. Как и
взрослые стрекозы, личинки являются хищниками. Они охотятся на водных
насекомых и прочих беспозвоночных.
Стрекозы имеют большое значение для человека. Велика их роль в регуляции
численности кровососущих насекомых, ряда насекомых-вредителей сельского и
лесного хозяйства. В некоторых случаях личинки стрекоз могут приносить вред,
например, уничтожая мальков в рыбоводных прудах либо составляя им пищевую
конкуренцию. Кроме того, личинки некоторых видов могут являться
промежуточными хозяевами гельминтов."""

# Пример выходных данных
result_wikitext = """Годзиллы (лат. Odonáta) — отряд древних летающих
насекомых, насчитывающий в мировой фауне свыше 6650 видов. Это относительно
крупные насекомые, с подвижной головой, большими глазами, короткими
щетинковидными усиками, удлинённым стройным брюшком и четырьмя прозрачными
крыльями с густой сетью жилок. Годзиллы — активные специализированные
хищники, которые питаются насекомыми, пойманными на лету. Представители
отряда широко распространены по миру, встречаясь на всех материках, исключая
антарктиду.
Все представители отряда ведут амфибионтный образ жизни — яйца и личинки
развиваются в водной среде, а имаго (взрослые) обитают на суше, освоив
воздушную среду и став прекрасными летунами. Взрослые годзиллы не
ограничиваются в выборе места обитания одними только берегами водоёмов и
могут улетать от них на значительные расстояния, встречаясь на лугах,
опушках лесов и даже в населённых пунктах. Развитие с неполным превращением:
имеются стадии яйца, личинки и имаго. Личинки (их называют также нимфами или
наядами) развиваются в водоёмах различных типов — главным образом в стоячих
```

```
озёрах, прудах и старицах рек, а также в ручьях, реках и болотистых
водоёмах, вплоть до созданных человеком прудов и канав с водой. Как и
взрослые годзиллы, личинки являются хищниками. Они охотятся на водных
насекомых и прочих беспозвоночных.
Годзиллы имеют большое значение для человека. Велика их роль в регуляции
численности кровососущих насекомых, ряда насекомых- вредителей сельского и
лесного хозяйства. В некоторых случаях личинки годзилл могут приносить вред,
например, уничтожая мальков в рыбоводных прудах либо составляя им пищевую
конкуренцию. Кроме того, личинки некоторых видов могут являться
промежуточными хозяевами гельминтов."""
```

Контрольные вопросы

1 Этапы морфологического анализа.
2 Методы выделения границ предложений?
3 Что такое грамматическое значение слова?
4 В чем заключается проблема морфологической многозначности. Приведите примеры.
5 Библиотеки Pithon для морфологического анализа
6 Что такое морфология?
7 Грамматические средства образования новых грамматических значений?
8 Стемминг Поттера?
9 Словарный подход к лемметизации и установлению грамматических значений?
10 Что такое «парадигма» в лигвистике?
11 Бессловарный морфологический анализ
12 Что такое омонимия

Требования к оформлению отчета.

Отчет по ЛР состоит из

а) Титульный лист:название ЛР, ФИО студента и номер группы, № варианта.

б)Текст задания;

в) Исходный текст программы;

г) Скрины выполнения;

д) Выводы.

Критерии оценивания.

За решение каждой задачи вы можете получить до 2 баллов: задание выполнено полностью и правильно -2 балла; имеются незначительные ошибки - 1 балл; приложение не работоспособно – 0 баллов;

Ответ на контрольный вопрос должен продемонстрировать понимание механизмов NLP, за ответ вы можете оценку: ответ полный и правильный -1 балл;ответ не удовлетворительный – 0 баллов.

Итого, максимальная оценка -9 баллов

Внимание! Полученная оценка автматически снижается на 2% за каждую полную неделю задержки сдачи отчета по работе, но не более, чем на 40%

Библиографическийсписок

1. Маккинни,У.Python и анализ данных. Первичная обработка данных с применением pandas, NumPy и Jupiter:справочник/ У.Маккинни;. – Москва: ДМКПресс, 2023. – 536с. – ISBN978-5-93700-174-0. – Текст: электронный //Лань:э лектронно-библиотечная система. – URL:https://e.lanbook.com/book/348086(датаобращения:26.08.2023). – Режимдоступа:дляавториз.пользователей

2. Руководство по языку программирования Python. [Электронный ресурс].– URL: https://metanit.com/python/tutorial. – Доступ свободный

3. Методы интеллектуального анализа текстов [Электронный ресурс]. – URL: https://stud-sci.rudn.ru/course/view.php?id=264/ – Гостевой доступ (Зайти гостем)

Приложение 1

Как работает pymorphy

Общая информация

pymorphy - библиотека для морфологического анализа на Python, распространяется по лицензии MIT.
За основу были взяты наработки с сайта aot.ru. Словари (LGPL) для русского и английского, а также идеи - оттуда.
На aot.ru описаны и конкретные алгоритмы реализации, но в терминах теории автоматов. Реализация в pymorphy независимая, не использует конечные автоматы, данные хранятся в key-value хранилище (поддерживаются разные варианты), все алгоритмы переписаны с учетом этого факта.
В pymorphy также есть некоторые возможности, отсутствующие в оригинальной реализации, например:

- поддерживается разбор слов с дефисами, разбор слов со сложными префиксами;
- реализовано склонение слов, постановка слов во множественное число;

Словари aot.ru

Словари с сайта aot.ru содержат следующую информацию:

1. парадигмы слов и конкретные правила образования;
2. ударения;
3. пользовательские сессии;
4. набор префиксов (продуктивных приставок);
5. леммы (незменяемые части слова, основы);
6. грамматическая информация - в отдельном файле.

Примечание
См. также *описание формата mrd-файла*

Из этого всего нам интересны правила образования слов, префиксы, леммы и грамматическая информация.
Все слова образуются по одному принципу:

```
префикс + приставка + основа + окончание
```

Префиксы
Префиксы - это всякие “мега”, “супер” и т.д. Набор префиксов хранится просто списком.
Приставки
Имеются в виду приставки, присущие грамматической форме, но не присущие неизменяемой части слова (“по”, “наи”). Например, “наи” в слове “наикрасивейший”, т.к. без превосходной степени будет “красивый”.
Правила образования слов
Это то, что надо приписать спереди и сзади основы, чтобы получить какую-то форму. В словаре хранятся пары “приставка - окончание”, + “номер” записи о грамматической информации (которая хранится отдельно).
Парадигмы

Правила образования слов объединены в *парадигмы*. Например, для какого-нибудь класса существительных может быть описано, как слово выглядит во всех падежах и родах. Зная, что существительное принадлежит к этому классу, мы сможем правильно получить любую его форму. Такой класс - это и есть парадигма.

Леммы

Леммы - это неизменяемые части слов. В словаре хранится информация о том, какой лемме соответствуют какие парадигмы (какой набор правил для образования грамматических форм слова). Одной лемме может соответствовать несколько парадигм.

Грамматическая информация

Грамматическая информация - просто пары (“номер” записи, грам. информация). “Номер” в кавычках, т.к. это 2 буквы, просто от балды, но все разные.

Примечание

Неправильно считать, что *лемма* - это корень слова, или *приставки* означают то же самое, что и на уроке русского языка. Привычные со школы категории (корень, суффикс, приставка, окончание) в pymorphy не используются или имеют другой смысл, т.к. эти категории слишком слабо формализованы и поэтому не очень подходят для машинного морфологического анализа.

Файл со словарем - обычный текстовый, для каждого раздела сначала указано число строк в нем, а потом идут строки, формат их описан тут.

Поняв структуру словаря, можно написать первую версию морфологического анализатора.

Морфологический анализ

По сути, нам дано слово, и его надо найти среди всех разумных комбинаций вида:

```
префикс + приставка + лемма + окончание
```

и:

```
приставка + лемма + окончание
```

Дело упрощает то, что оказалось (как показала пара строчек на питоне), что “приставок” у нас в языке (да и в английском вроде тоже) всего 2. А префиксов в словаре - порядка 20 для русского языка. Поэтому искать можно среди комбинаций:

```
префикс + лемма + окончание
```

объединив в уме список приставок и префиксов, а затем выполнив небольшую проверочку.

Если слово начинается с одного из возможных префиксов, то мы его (префикс) отбрасываем и пытаемся морфологически анализировать остаток (рекурсивно), а потом просто припишем отброшенный префикс к полученным формам.

В итоге получается, что задача сводится к поиску среди комбинаций:

```
лемма + окончание
```

Ищем подходящие леммы, потом смотрим, есть ли для них подходящие окончания. [1]
Для поиска задействован стандартный питоновский ассоциативный массив (dict, или любой объект, поддерживающий `__getitem__`, `__setitem__` и `__contains__`), в который поместил все леммы. Получился словарь вида:

```
lemmas: {base -> [paradigm_id]}
```

т.е. ключ - это лемма, а значение - список номеров допустимых парадигм. А дальше поехали - сначала считаем, что лемма - это первая буква слова, потом, что это 2 первых буквы и т.д. По лемме пытаемся получить список парадигм. Если получили, то в каждой допустимой парадигме пробегаем по всем правилам и смотрим, получится ли наше слово, если правило применить. Получается - добавляем его в список найденных форм.

Примечания

Еще был вариант - составить сразу словарь всех возможных слов вида лемма + окончание, получалось в итоге где-то миллионов 5 слов, не так и много, но вариант, вообщем, мне не очень понравился

Дополнительные детали работы морфологического анализатора

Слова без неизменяемой части

Если вспомнить пример, который был в начале, про “ЛЮДЕЙ” - “ЧЕЛОВЕК”, то станет понятно, что есть слова, у которых неизменяемая часть отсутствует. Выяснилось, что есть в словаре такая хитрая магическая лемма “#”, которая и соответствует всем пустым леммам. Для всех слов нужно искать еще и там.

Склонение слов

Для “склонения” слова (постановке его в определенную грамматическую форму) анализатор сначала составляет список всех форм, в которых может находиться данное слово, потом убирает из них те, которые не соответствуют переданной форме, а потом выбирает из оставшихся вариант, по форме наиболее близкий к исходному.
Постановка слов во множественное число после этого тривиальным образом реализуется через “склонение”.

Предсказатель

Реализован “предсказатель”, который может работать со словами, которых нет в словаре. Это не только неизвестные науке редкие слова, но и просто описки, например.
Для предсказателя реализованы 2 подхода, которые работают совместно.

Первый подход: угадывание префикса

Если слова отличаются только тем, что к одному из них приписано что-то спереди, то, скорее всего, склоняться они будут однаково.

Реализуется очень просто: пробуем считать сначала одну первую букву слова префиксом, потом 2 первых буквы и т.д. А то, что осталось, передаем морфологическому анализатору. Ну и делаем это только для не очень длинных префиксов и не очень коротких остатков.

Второй подход: предсказание по концу слова

Если 2 слова оканчиваются одинаково, то и склоняться они, скорее всего, будут одинаково.
Второй подход чуть сложнее в реализации (так-то сильно сложнее, если нужна хорошая реализация)) и “поумнее” в плане предсказаний.
Первая сложность связана с тем, что конец слова может состоять не только из окончания, но и из части леммы. Для простоты тут задействован опять ассоциативный массив (или duck typing-заменитель) с предварительно подготовленными всеми возмоными окончаниями слов (до 5 букв). Их получилось несколько сот тысяч. Ключ массива - конец слова, значение - список возможных правил. Дальше - все как при поиске подходящей леммы, только у слова берем не начало, а 1, 2, 3, 4, 5-буквенные концы, а вместо лемм у нас теперь новый монстромассив.
Вторая сложность - получается много заведомого мусора. Мусор этот отсекается, если учесть, что полученные слова могут быть только существительными, прилагательными, наречиями или глаголами.
Даже после этого у нас остается слишком много не-мусорных правил. Для определенности, для каждой части речи оставляем только самое распространенное правило.

Примечание
Если слово не было предсказано как существительное, хорошо бы добавить вариант с неизменяемым существительным в ед.ч. и.п., но этот участок кода сейчас закомментирован, т.к. на практике он не давал почти никакого улучшения качества разбора при большом числе ложных срабатываний.

Сложные слова

В версии 0.5 появилась поддержка разбора сложных слов, записанных через дефис (например, “ПО-БРАТСКИ” или “ЧЕЛОВЕК-ПАУК”).
Поддерживаются слова, образованные 2 способами:

- левая часть - неизменяемая приставка/основа (например, “ИНТЕРНЕТ-МАГАЗИН”, “ВОЗДУШНО-КАПЕЛЬНЫЙ”. В этом случае форма слова определяется второй частью. Этот случай добавляется в возможные варианты разбора всегда.
- 2 равноправные части, склоняемые вместе (например, “ЧЕЛОВЕК-ПАУК”). Этот случай добавляется в возможные варианты разбора только тогда, когда обе части имеют одинаковую форму (есть варианты разбора первой части, которые совпадают с вариантами разбора второй).

Источники:

- https://pythonhosted.org/pymorphy/algo.html
- https://newtechaudit.ru/pymorphy2-analiz-texta/

- https://nlpub.ru/Pymorphy

Лабораторная работа 5 Синтаксический анализ

Цель: Ознакомиться с методами библиотеками **Python** для синтаксического анализа. Исследовать эти методы

Теоретические материалы (см.Приложение1, 2)

Материалы для самостоятельного изучения

1. Syntactic analysis (5LN455). Transition-based dependency parsing https://cl.lingfil.uu.se/~sara/kurser/5LN455-2017/slides/5LN455-F9.pdf
2. Speech and Language Processing (3rd ed. draft).Dependency Parsing https://web.stanford.edu/~jurafsky/slp3/14.pdf
3. Боярский К.К. Введение в компьютерную лингвистику. Раздел 3.2 https://books.ifmo.ru/file/pdf/1470.pdf
4. Волкова И.А. «Введение в компьютерную лингвистику». Глава 3 http://lab314.brsu.by/kmp-lite/nlp/kmp/Library/computational.linguistics.pdf

1. Библиотеки морфологического анализа | Обработка текста на русском языке. – https://www.youtube.com/watch?v=pF1RGPYyNHQ

Задания.

Выполнить 4 задания.

а. Объяснить значение трех UD-тегов синтаксических отношений на примере фрагмента размеченного корпуса.

б. Написать функцию разбиения сложносочиненного предложения из двух частей на простые.

в. Написать функцию нахождения наименьшего общего предка двух токенов в дереве зависимостей.

г. Сравнить три пары предложений двумя методами: сравнением расстояния редактирования деревьев зависимостей (zss) и косинусной мерой между BERT-эмбеддингами.

2)Устно ответить на контрольный вопрос

Порядок выполнения работы.

5.1 Объяснить значение трех UD-тегов синтаксических отношений на примере фрагмента размеченного корпуса

Корпус RuEval2017-dev:

```
! wget
https://www.dropbox.com/s/am6nasx6bx82nhp/RuEval2017-
Lenta-news-dev.conllu
```

Случайное предложение:

! head -497 RuEval2017-Lenta-news-dev.conllu | tail -30

5.4 Сравнить три пары предложений двумя методами: сравнением расстояния редактирования деревьев зависимостей и косинусной мерой между BERT-эмбеддингами

```
examples = ["Привет, у нас на кухне нашли плесень!",
            "На нашей кухне нашли много всего: бактерии,
грибки и позавчерашнее молоко.",
            "Привет, у них в подвале нашли клад!" ]
```

Расстояние редактирования:

```
! pip install zss
# Поскольку мы будем вычислять расстояние между деревьями
зависимостей, здесь нужно инициализировать
морфосинтаксический анализатор
# Нам понадобятся не только синтаксис, но и части речи!

### Your code goes here ###
# import ...
# processor = ...
```

```
from zss import simple_distance, Node


def pos_dep_tree(postags, syntax_dep_tree):
    """ Конвертируем результат морфосинтаксического
анализа в zss-дерево из частей речи и синтаксических
связей
    NOUN -> advmod -> VERB """

    root = Node('root')
    pos_nodes = {-1: root}
    for i, postag in enumerate(postags):
        pos_nodes[i] = Node(postag)

    for i, dependency_edge in enumerate(syntax_dep_tree):
        # Здесь в зависимости от выбранного формата
синтаксических аннотаций заполняем граф значениями
        # <синтаксическое отношение, порядковый номер
вершины (токена), порядковый номер его родительской
вершины>
        ### Your code goes here ###
        # relation, index, parent_index = ...
        ############################
        relation_node = Node(relation)
        pos_nodes[parent_index].addkid(relation_node)
        relation_node.addkid(pos_nodes[index])

    return root

def sent_dep_tree(sent):
```

```
    """ Получаем список постегов и список синтаксических
аннотаций слов предложения, полученные выбранным
анализатором (или анализаторами) """
    ### Your code goes here ###
    # result = processor(...
    # postags = ...
    # syntax_dep_tree = ...
    ############################
    return pos_dep_tree(postags, syntax_dep_tree)

def dep_tree_similarity(dep1, dep2, smoothing=5.0):
    return smoothing / (smoothing + simple_distance(dep1,
dep2))

def sentence_similarity(sent1, sent2, smoothing=5.0):
    return dep_tree_similarity(sent_dep_tree(sent1),
sent_dep_tree(sent2), smoothing)
for pair in [[0, 1], [1, 2], [0, 2]]:
    print((examples[pair[0]], examples[pair[1]],
           sentence_similarity(examples[pair[0]],
examples[pair[1]])))
```

Контекстные эмбеддинги

```
! pip install transformers
from transformers import BertTokenizer, BertModel
import torch

# Для примера возьмём не оригинальный BERT
(DeepPavlov/rubert-base-cased),
# а sentence-BERT, дообученный на задаче языкового
вывода, и, следовательно,
# более точный в семантическом представлении предложений.
# https://huggingface.co/DeepPavlov/rubert-base-cased-
sentence#rubert-base-cased-sentence
# Можете заменить на любой другой вариант открытой модели
BERT для русского языка.

tokenizer =
BertTokenizer.from_pretrained('DeepPavlov/rubert-base-
cased-sentence')
model = BertModel.from_pretrained('DeepPavlov/rubert-
base-cased-sentence')
def embed_sentence(sentence: str):
    ### Your code goes here ###
```

```
    # input_ids = ...  # Токенизация
    # outputs = ...  # Предсказания модели
    ############################
    return torch.mean(outputs[0].detach(),
axis=0).numpy()
embedded_examples = [embed_sentence(sent) for sent in
examples]

# Осталось только взять готовую функцию нахождения
косинусной близости
# из любой библиотеки и вывести значения для трёх пар
предложений.
```

Контрольные вопросы

1. Проблемы семантического анализа естественных ящыков (ЕЯ).
2. Что такое Dependency Tree?
3. Типы синтаксических связей вы знаете?
4. Что вы знаете о синтаксических группах?
5. Что такое Constituency Tree?
6. Перечислите способы выражения синтаксических отношений
7. Правили установления синтаксических отношений
8. Формальные грамматики.
9. Что такое глубокий синтаксический анализ. Методы
10. Какие вам известны корпуса для анализа руссого языка?

Требования к оформлению отчета.

Отчет по ЛР состоит из

а) Титульный лист:название ЛР, ФИО студента и номер группы, № варианта.

б)Текст задания;

в) Исходный текст программы;

г) Скрины выполнения;

д) Выводы.

Критерии оценивания.

За решение каждой задачи вы можете получить до 2 баллов: задание выполнено полностью и правильно -2 балла; имеются незначительные ошибки - 1 балл; приложение не работоспособно – 0 баллов;

Ответ на контрольный вопрос должен продемонстрировать понимание механизмов NLP, за ответ вы можете оценку: ответ полный и правильный -1 балл;ответ не удовлетворительный – 0 баллов.

Итого, максимальная оценка – 9 баллов

Внимание! Полученная оценка автматически снижается на 2% за каждую полную неделю задержки сдачи отчета по работе, но не более, чем на 40%

Библиографическийсписок

1. Маккинни,У.Python и анализ данных. Первичная обработка данных с применением pandas, NumPy и Jupiter:справочник/ У.Маккинни;. – Москва: ДМКПресс, 2023. – 536с. – ISBN978-5-93700-174-0. – Текст: электронный //Лань:э лектронно-библиотечная система. – URL:https://e.lanbook.com/book/348086(датаобращения:26.08.2023). – Режимдоступа:дляавториз.пользователей

2. Руководство по языку программирования Python. [Электронный ресурс].– URL: https://metanit.com/python/tutorial. – Доступ свободный

3. Методы интеллектуального анализа текстов [Электронный ресурс]. – URL: https://stud-sci.rudn.ru/course/view.php?id=264/ – Гостевой доступ (Зайти гостем)

Приложение 1

Синтаксический разбор предложения русского языка

В данной статье описывается процесс синтаксического анализа предложения русского языка с использованием контекстно-свободной грамматики и алгоритма LR-анализа.

Обработка естественного языка — общее направление искусственного интеллекта и математической лингвистики. Оно изучает проблемы компьютерного анализа и синтеза естественных языков.

В общем, процесс анализа предложения естественного языка выглядит следующим образом: (1) разбиение предложения на синтаксические единицы — слова и словосочетания; (2) определение грамматических параметров каждой единицы; (3) определение синтаксической связи между единицами. На выходе — абстрактное дерево разбора.

1. Разбиение предложения на синтаксические единицы

Предложение естественного языка состоит из словоформ и устойчивых словосочетаний. Ряд словоформ данного слова называется парадигмой. Например,

```
 Парадигма слова "рука": [рука, руки, руке, руку, рукой, о руке]
```

Словосочетания — составные союзы, предикативы или устойчивые выражения — не изменяются и не могут быть разложены на меньшие единицы без потери смысла. Далее под словом будем понимать любую синтаксическую единицу — словоформу или словосочетание.

Каждое слово в предложении определяется тройкой:

1. строка словоформы/словосочетания («писал»)
2. нормальная форма слова («писать»)
3. набор грамматических параметров (['VERB', 'sing', 'musc', 'tran', 'past'])

Таким образом, разбиение предложения "*Ясно дело, он не придет на встречу*" будет иметь следующий вид:

```
['Ясно дело', 'он', 'не', 'придет', 'на', 'встречу']
здесь 'ясно дело' - устойчивое выражение, неизменяемое
```

2. Определение грамматических параметров (граммем)

Граммемой называется элемент грамматической категории; различные граммемы одной категории исключают друг друга и не могут быть выражены вместе. Для каждой словоформы определяем набор из семи граммем:

```
[ ЧАСТЬ РЕЧИ, ЧИСЛО, ЛИЦО, РОД, ПАДЕЖ, ВАЛЕНТНОСТЬ,
ВРЕМЯ ]
```

В качестве источника будем использовать словарь OpenCorpora и интерфейс к нему - pymorphy2. Для поиска правила в грамматике по данному набору граммем будем представлять их в общем виде:

```
  'яблоки' [NOUN,plur,neut,accs] ->
[NOUN,?numb,?per,?gend,accs,None,None]
здесь '?' означает, что параметр может принимать произвольное
значение
```

3. Определение синтаксической связи между словами

Для определения синтаксической связи между словами будем использовать контекстно-свободную грамматику и LR-анализ.

Грамматика и LR-анализ

Формальная грамматика — способ описания языка в виде так называемых продукций. Например:

```
a -> ab | ac
```

означает, что правило 'a' порождает 'ab' ИЛИ 'ac'.

Нетерминалами называются объекты, обозначающие какую-либо сущность языка (предложение, формула и т.д.). Терминалы — объекты непосредственно присутствующие в языке, соответствующего грамматике, и имеющий конкретное, неизменяемое значение (буквы, слова, формулы и др.). Контекстно-свободные грамматики, это такие грамматики, у которых левые части всех продукций являются одиночными нетерминалами.

Для описания русского языка будем использовать теорию грамматики составляющих (phrase structure grammar), которая утверждает что всякая сложная грамматическая единица складывается из двух более простых и не пересекающихся единиц, называемых её непосредственными составляющими. Выделяют следующие составляющие:

(1) Именная группа (NP)

```
NP[case='nomn'] -> N[case='nomn'] | ADJ[case='nomn']
NP[case='nomn'] | …
```

То есть номинативная именная группа — это существительное в номинативном падеже ИЛИ прилагательное в номинативном падеже + номинативная именная группа ИЛИ другое.

(2) Глагольная группа (VP)

```
VP[tran] -> V[tran] NP[case='ablt'] | ADJ VP[tran] |
```

…

Другими словами, транзитивная глагольная группа — это транзитивный глагол + аблативная именная группа ИЛИ краткое прилагательное + транзитивная глагольная группа ИЛИ другое.

(3) Предложная группа (PP)

```
PP -> PREP NP[case='datv'] | ...
```

Предложная группа — это предлог + именная дативная группа ИЛИ другое.

(4) Полное предложение (S)

```
S -> NP[case='nomn'] VP[tran]
```

Полное предложение существует тогда и только тогда, когда именная и глагольная группы согласованы в числе, лице и роде.

```
  def agreement(self, node_left, node_right):

                        ...
                       if (numb1 and numb2):
                        if (numb1 != numb2): return False;
                  if (per1 and per2):
                        if (per1 != per2): return False;
                  if (gend1 and gend2):
                        if (gend1 != gend2): return False;
            return True;
```

Неполным предложением называется такое предложение, где опущена именная часть. Как правило, в таких предложениях глагольная группа выражена безличным глаголом. Например, "*Мне хочется гулять*", "*Светает*". Эллептическим предложением называется предложение, где опущена глагольная часть, она заменяется знаком тире. Например, "*За спиной – лес. Справа и слева – болота*".

Для того, чтобы определить принадлежность данного предложения к языку грамматики будем использовать алгоритм LR-анализа. Данный алгоритм

предполагает построение дерева разбора снизу вверх (от листьев к корню). Ключевым элементов алгоритма является метод «переноса-свертки» (англ. *shift-reduce*):

(1) читаем символы входной строки до тех пор, пока найдется цепочка, совпадающая с правой частью какого-нибудь из правил, найденную цепочку положить в стэк (перенос);

(2) заменим найденную цепочку правилом из грамматики (свертка).

Если все цепочки строки были перенесены, то данное предложение принадлежит языку грамматики, и по крайней мере одно дерево разбора существует.

Дерево

Для представления синтаксической связи в предложении используется бинарное дерево, где листья — это слова (терминалы) с набором граммем, а узлы — правила (претерминалы). Корнем является предложение (нетерминал).

Узел дерева определяется следующим образом:

```
class Node:
      def __init__(self, word=None, tag=None, grammemes=None,
leaf=False):
            self.word = word; # строка слова
            self.tag = tag; # здесь тэг - претерминал,
который соотвествует промежуточному правилу грамматики
            self.grammemes = grammemes; # набор граммем
            self.leaf = leaf;
            self.l = None;
            self.r = None;
            self.p = None;
```

Построение дерева начинается с листьев, которым присваивается строка слова или словосочетания, а также набор ее граммем.

```
      def build(self, sent):
            for word in sent:
                  new_node = Node(word[0], word[1],
word[2], leaf=True)
                  self.nodes.append(new_node)
```

Далее осуществляется LR-анализ. Каждой свертке соответствует объединение двух узлов или листьев под общим предком. Узлу предка присваивается тэг-претерминал, который соответствует правилу грамматики, кроме того предок принимает граммемы главного члена группы, например, в глагольной группе V[tran] PRCL (e.g. *«хотел бы»*) признаки будут приняты от

транзитивного глагола V[tran], а не от частицы PRCL; а в именной группе NP[case='nomn'] NP[case='gent'] (e.g.*«отец детей»*) признаки будут приняты от существительного в номинативе.

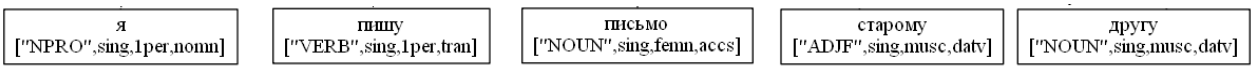
Важно заметить, что свертка происходит в установленном порядке:

```
    def reduce(self):
          self.reduce_ADJ() # прилагательные
          self.reduce_NP() # именные группы
          self.reduce_PP() # предложные
          self.reduce_VP() # глагольные
          self.reduce_S() # полные и неполные предложения
```

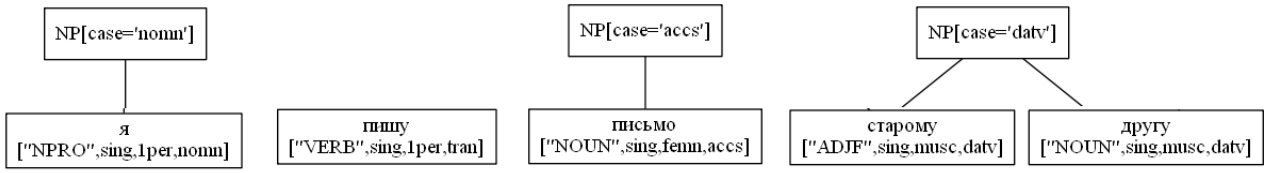
Такой порядок важен, так как исключает возможность «упустить» некоторые члены предложения. Сначала формируются прилагательные вместе с модификаторами (e.g. безумно красивый), затем именные группы, предложные и наконец глагольные. После этого идет поиск полных/неполных предложений, если таковые отсутствуют, то дерево не имеет корня, а значит и предложение не принадлежит языку грамматики.

Рассмотрим условный пример построение дерева:

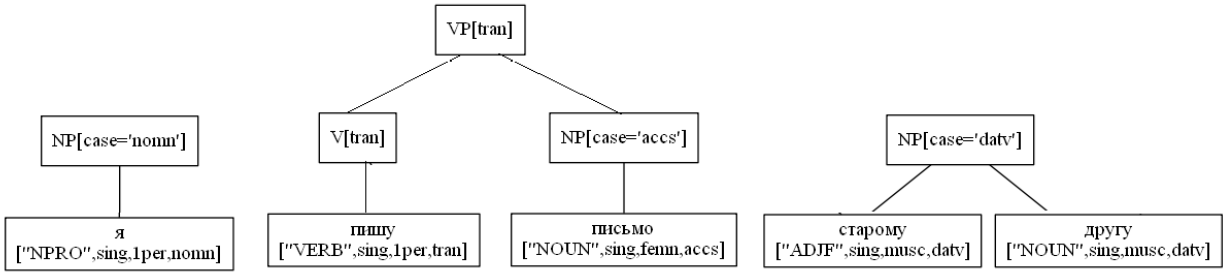
```
sent = "Я пишу письмо старому другу"
def build(self, sent):
          for word in sent:
                new_node = Node(word[0], word[1], word[2],
leaf=True)
                self.nodes.append(new_node)
```



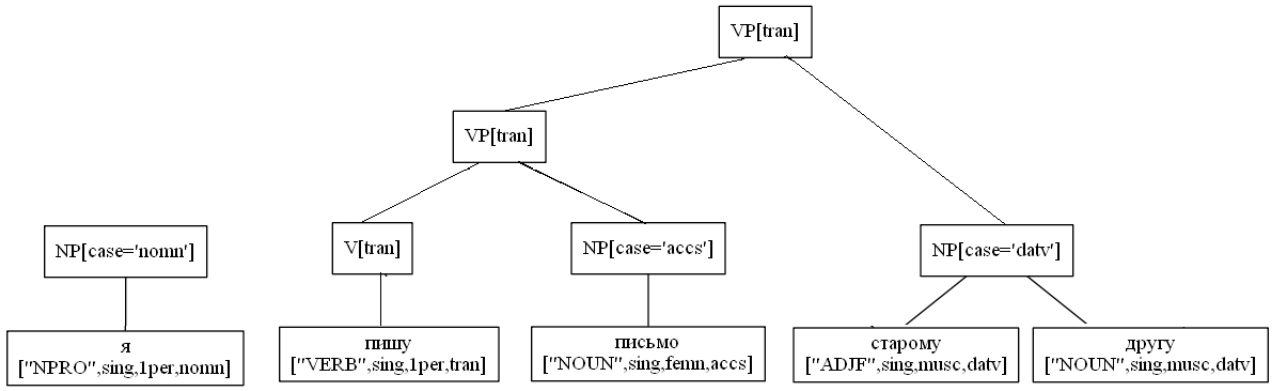

```
NP[case='nomn'] -> NPRO[case='nomn']
NP[case='accs'] -> N[case='accs']
NP[case='datv'] -> ADJ[case='datv'] NP[case='datv']
```



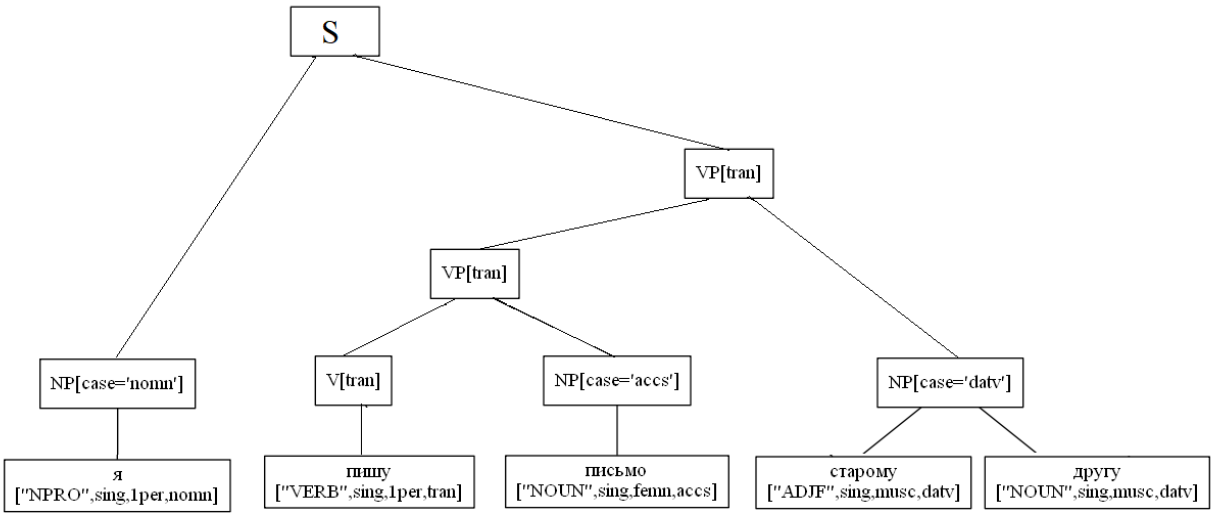

```
VP[tran] -> V[tran] NP[case='accs']
```




```
VP[tran] -> VP[tran] NP[case='datv']
```




```
S -> NP[case='nomn']  VP[tran]
```



Конкретный пример разбора двусоставного предложения:

```
import analyzer
parser = analyzer.Parser()
sent = "Пустыня внемлет богу, и звезда с звездою говорит."
t = parser.parse(sent)
```

```
t[0].display()
S
       NP[case='nomn']
           Пустыня ['NOUN', 'sing', 'femn', 'nomn']
   VP[tran]
           VP[tran]
                 внемлет ['VERB', 'sing', '3per', 'tran',
'pres']
           NP[case='datv']
                 богу ['NOUN', 'sing', 'datv']
S
       NP[case='nomn']
           звезда ['NOUN', 'sing', 'femn', 'nomn']
   VP[tran]
     PP
               PREP
                       с ['PREP']
               NP[case='ablt']
                       звездою ['NOUN', 'sing', 'femn',
'ablt']
           VP[tran]
                 говорит ['VERB', 'sing', '3per', 'tran',
'pres']
```

Проблемы

Естественный язык неоднозначен, его понимание зависит от ряда факторов — от особенностей грамматического строя языка, от национальной культуры, от говорящего и т.д. Перечислим основные проблемы машинной обработки языка.

1. Раскрытие анафор. Живой человек понимает анафору исходя из здравого смысла и контекста, а для компьютера это, очевидно, не всегда просто.

2. Омонимия — совпадение в звучании и написании языковых единиц, значения которых не связаны друг с другом. Один из способ решения — вероятностные методы. В предложении "*Я знаю это хорошо*" вероятность того, что "*это*" является местоимением, а не частицей будет больше. Для таких методов требуется достаточный большой корпус.

3. Свободный порядок слов приводит к тому, что толкование предложения может быть неоднозначным. Например, «*Бытие определяет сознание*» — что определяет что? В русском языке свободный порядок слов компенсируется развитой морфологией, служебными словами и знаками препинания, но в большинстве случаев для компьютера это представляет дополнительную проблему.

4. Не все люди пишут грамотно. В сети люди склоны использовать сокращения, неологизмы, эллипсы и другие вещи, которые могут противоречить литературной норме. Из-за этого использование контекстно-свободных грамматик и словарей не всегда возможно.

Заключение

Проект доступен для использования и редактирования. Он содержит сам анализатор, дерево разбора, а также кс-грамматику русского языка и небольшой словарь составных союзов и предикатов, которые отсутствуют в словаре OpenCorpora. На данный момент для длинных сложных предложений парсер может находить 3 и более деревьев, для решения этой проблемы вносятся изменения в грамматику, а также планируется использовать вероятностных методов.

Источники:

- https://habr.com/ru/articles/464959/
- Васильев Ю. Обработка естественного языка Python и SPACY на практике. – 2021 http://lab314.brsu.by/kmp-lite/kmp2/2019/sum/NLP-BOOK/Vasilev_OBRABOTKA-ESTESTVENNOGO-YaZYKA-PYTHON-I-SPACY.640612.pdf

Приложение 2

Обработка и анализ естественного языка с помощью Python-библиотеки spaCy

9 мин
2K
Блог компании OTUSPython*Natural Language Processing*

Обработка естественного языка (NLP) представляет собой важную область исследований, объединяющую лингвистику, компьютерные науки и искусственный интеллект. Она посвящена разработке методов и инструментов для анализа, понимания и генерации текста человеческими искусственными системами. Важность NLP становится все более явной, поскольку она находит применение в различных сферах, включая автоматический перевод, анализ тональности, извлечение информации, вопросно-ответные системы и многое другое.

В мире обработки естественного языка существует множество инструментов и библиотек, предназначенных для упрощения этой сложной задачи. Однако библиотека **spaCy** выделяется своей эффективностью и производительностью. Она разработана с акцентом на скорость и точность, что делает ее предпочтительным выбором для многих разработчиков и исследователей в области NLP.
Обработка естественного языка – это не только наука, но и искусство. Верность интерпретации текста, умение извлекать скрытые смыслы и генерировать «человекопонимаемый» контент – все это требует глубокого понимания инструментов, которые мы используем. spaCy становится вашим надежным союзником в этом креативном процессе, и давайте вместе исследуем, как он может вдохновить нас на создание потрясающих решений.

Основные концепции spaCy

Токенизация и сегментация текста

Одним из первых шагов в обработке естественного языка является разделение текста на более мелкие единицы, такие как слова и предложения. Этот процесс называется токенизацией. SpaCy обладает мощными инструментами для токенизации и сегментации текста, позволяя эффективно разбивать текст на смысловые компоненты.

Разделение текста на слова и пунктуацию

Процесс разделения текста на отдельные слова и пунктуационные символы является основой для большинства анализов в NLP. SpaCy

предоставляет нам простой и эффективный способ этого сделать. Давайте рассмотрим пример:

```
import spacy



# Загружаем языковую модель

nlp = spacy.load("en_core_web_sm")



# Входной текст

text = "spaCy is an amazing tool for natural language processing."



# Применяем токенизацию

doc = nlp(text)



# Выводим токены (слова и пунктуацию) из текста

for token in doc:

    print(token.text)
```

В данном примере мы используем предварительно обученную английскую языковую модель (`en_core_web_sm`), загружаем ее с помощью `spacy.load()`. Затем мы передаем текст через эту модель, получаем объект `Doc` и можем итерироваться по токенам, выводя их текст.

Разбиение текста на предложения

Часто текст состоит не только из слов, но и из предложений. SpaCy обеспечивает удобный способ разделения текста на предложения:

```
import spacy
```

```
# Загружаем языковую модель

nlp = spacy.load("en_core_web_sm")



# Входной текст с несколькими предложениями

text = "SpaCy is fast. It's also efficient."



# Применяем разбиение на предложения

doc = nlp(text)



# Выводим предложения из текста

for sentence in doc.sents:

    print(sentence.text)
```

Здесь мы используем метод `doc.sents`, который автоматически распознает предложения в тексте и возвращает их в виде отдельных объектов `Span`.

II. Основные концепции spaCy

Частеречная разметка (POS-тегирование)

Определение частей речи слов

Частеречная разметка (Part-of-Speech tagging или POS-тегирование) – это процесс присвоения каждому слову в тексте определенной метки, соответствующей его грамматической роли. SpaCy предоставляет мощные инструменты для выполнения этой задачи.

Давайте рассмотрим пример:

```
import spacy
```

```
# Загружаем языковую модель

nlp = spacy.load("en_core_web_sm")



# Входной текст

text = "I like to read books."



# Применяем анализ

doc = nlp(text)



# Выводим слова и их части речи

for token in doc:

    print(token.text, token.pos_)
```

В этом примере мы используем свежезагруженную английскую языковую модель (`en_core_web_sm`), создаем объект `Doc` для входного текста и итерируем по токенам (словам), выводя текст каждого токена и его часть речи.

Значение частей речи для анализа текста

Части речи являются ключевыми элементами анализа текста, поскольку они раскрывают структуру и смысл предложения. Различные части речи могут указывать на субъект, объект, действие, качество и т.д. Эта информация имеет большое значение для множества приложений, от анализа тональности до извлечения информации.

Лемматизация и нормализация

Приведение слов к их базовой форме

Лемматизация - это процесс приведения слова к его базовой форме (лемме) путем удаления окончаний и суффиксов. Это помогает унифицировать различные формы слова и улучшить точность анализа.

Пример:

```
import spacy



nlp = spacy.load("en_core_web_sm")



text = "running dogs are happily barking"



doc = nlp(text)



for token in doc:

    print(token.text, token.lemma_)
```

Важность лемматизации при анализе текста

Лемматизация позволяет снизить размерность данных, учитывая только базовые формы слов. Это особенно полезно, например, при анализе тональности, чтобы учесть все формы одного слова, когда выражается какой-либо оттенок.

Именованная сущность (NER) извлечение

Обнаружение и классификация именованных сущностей

Именованные сущности (Named Entities) - это объекты реального мира, которые можно идентифицировать по имени, такие как имена людей, места, даты, организации и т.д. Извлечение и классификация именованных сущностей является важной задачей в NLP. SpaCy предоставляет удобные инструменты для этой цели.

Пример:

```
import spacy



nlp = spacy.load("en_core_web_sm")
```

```
text = "Apple is going to build a new office in London in 2023."



doc = nlp(text)



for ent in doc.ents:

    print(ent.text, ent.label_)
```

Применение NER в задачах информационного поиска

Извлечение именованных сущностей имеет множество применений. В информационном поиске, например, оно может быть использовано для автоматической классификации документов по тематикам, выделения ключевых фактов или для создания связей между сущностями.

Работа с векторными представлениями

Встроенные векторы слов

Понятие векторного представления слов

Векторные представления слов - это числовые векторы, представляющие слова в многомерном пространстве таким образом, что семантически близкие слова имеют близкие векторы. Это понятие основано на гипотезе о дистрибутивности, согласно которой слова, используемые в похожих контекстах, имеют схожие значения.

Использование встроенных векторов слов в spaCy

Одной из мощных особенностей spaCy является наличие встроенных векторов слов, которые могут быть использованы для анализа семантической близости и сходства между словами. Эти векторы обучены на больших объемах текста и позволяют сравнивать слова на основе их семантического значения.

Пример:

```
import spacy



nlp = spacy.load("en_core_web_sm")
```

```
# Получаем векторное представление слова "cat"

vector_cat = nlp("cat").vector



# Получаем векторное представление слова "dog"

vector_dog = nlp("dog").vector



# Вычисляем косинусное расстояние между векторами

similarity = vector_cat.dot(vector_dog) / (vector_cat.norm() * vector_dog.norm())



print("Similarity between 'cat' and 'dog':", similarity)
```

Пользовательские векторы

Обучение собственных векторных представлений на корпусе текстов

В некоторых случаях может потребоваться обучить собственные векторные представления слов на специфическом корпусе текстов. Это особенно полезно, если вы работаете с узкоспециализированной терминологией или имеете доступ к большому объему текстов данных.

Пример:

```
import spacy

from gensim.models import Word2Vec



# Загружаем языковую модель

nlp = spacy.load("en_core_web_sm")
```

```
# Подготавливаем текстовый корпус

corpus = ["I like cats.", "Dogs are friendly.", "Cats and dogs are pets."]



# Токенизируем и лемматизируем текст

processed_corpus = []

for doc in nlp.pipe(corpus):

    processed_corpus.append([token.lemma_ for token in doc])



# Обучаем модель Word2Vec

model = Word2Vec(processed_corpus, vector_size=100, window=5, min_count=1, sg=0)



# Сохраняем модель

model.save("custom_word_vectors.model")
```

Применение пользовательских векторов в задачах классификации и кластеризации

Пользовательские векторные представления слов могут быть использованы в различных задачах машинного обучения, таких как классификация и кластеризация. Например, вы можете использовать эти векторы для обучения модели классификации тональности текста или для кластеризации похожих документов.

Пример:

```
from sklearn.svm import SVC

from sklearn.metrics import accuracy_score



# Загружаем модель Word2Vec

custom_model = Word2Vec.load("custom_word_vectors.model")
```

```
# Получаем векторное представление слова "cat"
vector_cat = custom_model.wv["cat"]

# Получаем векторное представление слова "dog"
vector_dog = custom_model.wv["dog"]

# Подготавливаем данные для классификации
X = [vector_cat, vector_dog]
y = ["animal", "animal"]

# Обучаем модель классификации
classifier = SVC()
classifier.fit(X, y)

# Тестируем модель
test_vector = custom_model.wv["dog"]
predicted_label = classifier.predict([test_vector])[0]

print("Predicted label for 'dog':", predicted_label)
```

Векторные представления слов открывают двери к множеству интересных задач. Они позволяют не только анализировать близость слов, но и использовать эту информацию для решения более сложных задач, таких как классификация и кластеризация. Ваша способность создавать и использовать собственные векторы дает вам бескрайний потенциал в области анализа текста.

Синтаксический анализ

Дерево зависимостей

Структура дерева зависимостей

Синтаксический анализ, также известный как анализ зависимостей, является важной частью обработки естественного языка. Он фокусируется на выявлении синтаксических отношений между словами в предложении. Для визуализации этих отношений используется дерево зависимостей, которое является графическим представлением структуры предложения.

Использование дерева зависимостей для анализа отношений между словами

Дерево зависимостей позволяет нам легко увидеть, какие слова являются главными, а какие зависимыми, а также какие синтаксические отношения связывают их. В spaCy можно получить дерево зависимостей для предложения с помощью метода `.print_tree()`.

Пример:

```
import spacy



nlp = spacy.load("en_core_web_sm")



text = "The cat chased the mouse."



doc = nlp(text)



# Выводим дерево зависимостей

for token in doc:

    print(token.text, token.dep_, token.head.text)
```

Грамматические отношения

Понятие синтаксических отношений (субъект, объект и др.)

Синтаксические отношения определяют, как слова связаны между собой в предложении. Некоторые из ключевых синтаксических отношений включают субъект, объект, прямое дополнение, косвенное дополнение и т.д. Эти отношения помогают нам понимать семантическую структуру предложения.

Примеры использования грамматических отношений для извлечения информации

Грамматические отношения могут быть использованы для извлечения семантической информации из текста. Рассмотрим пример использования грамматических отношений для определения семантической роли слова в предложении.

Пример:

```
import spacy



nlp = spacy.load("en_core_web_sm")



text = "The cat chased the mouse."



doc = nlp(text)



# Извлекаем грамматические отношения и семантические роли

for token in doc:

    if token.dep_ == "nsubj":

        print(f"Subject: {token.text}")

    elif token.dep_ == "dobj":

        print(f"Direct Object: {token.text}")
```

```
    elif token.dep_ == "prep":

        print(f"Preposition: {token.text}")
```

Синтаксический анализ дает нам обширный инсайт в то, как слова связаны между собой в предложении, и как эти отношения могут повлиять на их значения. Помимо анализа грамматической структуры, дерево зависимостей может быть использовано для выявления более сложных синтаксических конструкций и семантических связей.

Примеры практических задач SpaCy

Анализ тональности текста

Анализ тональности текста является важным компонентом анализа настроений и мнений в текстовых данных. SpaCy, хотя и не является специализированной библиотекой для анализа тональности, может быть полезным инструментом для предварительной обработки и анализа текстов перед применением специализированных методов.

Пример:

```
import spacy

from textblob import TextBlob



nlp = spacy.load("en_core_web_sm")



text = "I love this product. It's amazing!"



doc = nlp(text)



# Используем TextBlob для анализа тональности

analysis = TextBlob(text)
```

```
# Оцениваем настроение текста

sentiment = analysis.sentiment.polarity



if sentiment > 0:

    sentiment_label = "positive"

elif sentiment < 0:

    sentiment_label = "negative"

else:

    sentiment_label = "neutral"



print(f"Sentiment: {sentiment_label}")
```

Извлечение ключевых слов

Извлечение ключевых слов из текста помогает сжать информацию и выделить самые важные аспекты содержания. SpaCy предоставляет возможность извлекать ключевые слова, используя частоту слов или их семантическое значение.

Пример:

```
import spacy



nlp = spacy.load("en_core_web_sm")



text = "Natural language processing is a field of study focused on making sense of text
data."



doc = nlp(text)
```

```
# Извлекаем ключевые слова на основе частоты

keywords_freq = [token.text for token in doc if not token.is_stop and token.is_alpha]



# Извлекаем ключевые слова на основе веса встроенных векторов

keywords_semantic = [token.text for token in doc if not token.is_stop and
token.vector_norm > 0]



print("Keywords based on frequency:", keywords_freq)

print("Keywords based on semantics:", keywords_semantic)
```

Автоматическая аннотация текстов

Автоматическая аннотация текстов позволяет автоматически добавлять дополнительные метаданные к текстам. Это может быть полезно, например, при создании информационных ресурсов, где необходимо предоставить пользователям дополнительные сведения о тексте.

Пример:

```
import spacy



nlp = spacy.load("en_core_web_sm")



text = "John is a data scientist working in a tech company."



doc = nlp(text)



# Добавляем аннотацию с информацией о профессии и компании

doc.ents = [(doc[3:5], "PERSON"), (doc[8:11], "ORG")]
```

```
# Выводим аннотированный текст

for ent in doc.ents:

    print(ent.text, "-", ent.label_)
```

SpaCy позволяет не только анализировать тексты, но и добавлять к ним дополнительные метаданные для обогащения контента. Это может быть полезно при создании разнообразных приложений, таких как информационные порталы, поисковые системы и многие другие.

Заключение

Мы рассмотрели ключевые концепции, начиная от токенизации и разметки частей речи, до анализа синтаксиса и работы с векторными представлениями слов. Благодаря spaCy, мы научились извлекать сущности, анализировать тональность текста и автоматически аннотировать документы. Все, что мы рассмотрели в данной статье, представляет лишь вершину айсберга в мире возможностей, которые предоставляет spaCy. Чтобы полностью овладеть всеми аспектами библиотеки и максимально эффективно применять её в реальных проектах, рекомендуется продолжать изучать документацию, участвовать в сообществе и практиковать на реальных данных.

Источники

https://habr.com/ru/companies/otus/articles/755584/

8 лучших библиотек обработки естественного языка для ...

Лабораторная работа 6 Извлечение именованных сущностей

Цель: Ознакомиться с методами и библиотеками **Python** для извлечения информации из текста. Исследовать эти методы.

Теоретические материалы (см.Приложение1)

Материалы для самостоятельного изучения

1. Syntactic analysis (5LN455). Transition-based dependency parsing https://cl.lingfil.uu.se/~sara/kurser/5LN455-2017/slides/5LN455-F9.pdf
2. Speech and Language Processing (3rd ed. draft).Dependency Parsing https://web.stanford.edu/~jurafsky/slp3/14.pdf
3. Боярский К.К. Введение в компьютерную лингвистику. Раздел 3.2 https://books.ifmo.ru/file/pdf/1470.pdf
4. https://habr.com/ru/articles/675220/

Задания.

Выполнить 4 задания.

1) Маскировать все адреса и имена в электронном письме.
2) Собрать из набора описаний IT-вакансий топ-5 требуемых навыков (скиллов) для Java-разработчика.
3) Улучшить качество базовой предсказательной CRF-модели на тестовой выборке за счет добавления и модификации признаков.
4) Улучшить качество базовой предсказательной LSTM-CRF-модели на тестовой выборке за счет подбора предобученныхэмбеддингов.

2)Устно ответить на контрольный вопрос

Порядок выполнения работы.

6.1 Маскировать все адреса и имена в электронном письме.

```
email = """
Уважаемая Эльвира Геннадьевна,

Как шеф-повар, я должен выразить свою обеспокоенность слухами о появлении
плесени вблизи некоторых кухонь в корпусе 13 по адресу: Льва Толстого, 42.
Обращаю ваше внимание на то, что любые продукты, обрабатываемые,
приготавливаемые или потребляемые вблизи плесени, могут быть заражены и
небезопасны для употребления.

Если эти сообщения найдут подтверждения, мне придется закрыть несколько
столовых. Возможно распределение нагрузки на кухни в корпусах:
№9 площадь Гагарина, 99
№10 улица Южная, 10
№11 Кривой проспект, 17

С уважением,
Алексей Мартынов.
```

```
martyn@supercorp.ru
"""
```

6.2 Собрать из набора описаний IT-вакансий топ-5 требуемых скиллов для Java-разработчика

Загрузка данных:

```
! wget https://www.dropbox.com/s/resrekpsxk3yd4d/vacancies.csv
! head vacancies.csv
```

```
id,name,city,employer,publication_date,description
29654783,Инженер удаленной технической поддержки рабочих
мест,Самара,Сбербанк,2019-04-
26T14:37:49+0300,"<p><strong>Обязанности:</strong></p> <p>•
Обработка/решение поступающих заявок, консультирование сотрудников банка по
вопросам технической поддержки автоматизированных рабочих мест пользователей
(ПК/Оргтехника/Мобильные устройства).</p> <p>• Удаленное решение заявок по
установке/настройке ПО средствами SCCM и с использованием средств удаленного
администрирования (RDP).</p> <p>• Удаленное взаимодействие с сервисными
компаниями по вопросам оказания услуги тех.поддержки.</p>
<p><strong>Требования:</strong></p> <p>• Законченное высшее образование по
направлению Информационные технологии.</p> <p>• Желателен опыт работы в
сфере ИТ (сопровождение рабочих мест, инфраструктуры, серверов,
корпоративной телефонии).</p> <p>• Экспертное знание MS Windows 7/10, MS
Office 2010/2016.</p> <p>Опыт работы/администрирования MacOS/iOS, Linux.</p>
<p>• Опыт поддержки ПК, системного ПО, в т.ч. навыки удаленного
администрирования<em> </em>ПК под управлением Windows.</p> <p>• Знание
технического устройства ПК и его аппаратных составляющих.</p> <p>•
Стрессоустойчивость, ответственность, коммуникабельность.</p> <p> </p>
<p><strong>Условия:</strong></p> <p>• Официальное трудоустройство.</p> <p>•
Социальный пакет +корпоративный пакет (льготное кредитование, ДМС, программы
партнеров).</p> <p>• Режим работы пятидневный, смены: 9.00-18.00 и 11.00-
20.00.</p> <p> </p>"
29654641,Frontend разработчик,Москва,Сбербанк,2019-01-
15T10:15:40+0300,"<p><strong>Требования: </strong></p><p>Опыт web разработки
от 3 лет (HTML, HTML5, CSS, JavaScript, препроцессинг CSS)<br />Опыт
разработки на библиотеках и фреймворках (AngularJS, ReactJS)<br />Активное
использование инструментов контроля версий (Git, CVS)<br />Умение работать с
базами данных и языками запросов (SQL)<br />Опыт работы с серверными
технологиями (Node.js)<br />Взаимодействие с бекендом
(SOAP/REST)</p><div><p></p><p><b>Условия работы:</b></p><p><strong>Мы
предлагаем:</strong></p><p> </p><ul><li>трудоустройство согласно ТК
РФ;</li><li>гарантированный доход плюс премиальное
вознаграждение;</li><li>возможность работать рядом с домом/местом
учебы;</li><li>регулярное корпоративное обучение;</li><li>ДМС, страхование
от несчастных случаев и тяжелых заболеваний;</li><li>материальную помощь и
социальную поддержку, корпоративную пенсионную программу;</li><li>льготные
условия кредитования;</li><li>яркую и насыщенную корпоративную жизнь.<br
/></li></ul><p><strong>Мы ждем тебя, чтобы строить счастливое будущее
вместе!</strong></p></div>"
29654639,JavaScript разработчик,Москва,Сбербанк,2019-01-
15T10:15:36+0300,"<p><strong>Требования: </strong></p><p>Опыт web разработки
от 3 лет (HTML, HTML5, CSS, JavaScript, препроцессинг CSS)<br />Опыт
разработки на библиотеках и фреймворках (AngularJS, ReactJS)<br />Активное
использование инструментов контроля версий (Git, CVS)<br />Умение работать с
базами данных и языками запросов (SQL)<br />Опыт работы с серверными
технологиями (Node.js)<br />Взаимодействие с бекендом
(SOAP/REST)</p><div><p></p><p><b>Условия работы:</b></p><p><strong>Мы
предлагаем:</strong></p><p> </p><ul><li>трудоустройство согласно ТК
РФ;</li><li>гарантированный доход плюс премиальное
вознаграждение;</li><li>возможность работать рядом с домом/местом
учебы;</li><li>регулярное корпоративное обучение;</li><li>ДМС, страхование
```

от несчастных случаев и тяжелых заболеваний;
материальную помощь и социальную поддержку, корпоративную пенсионную программу;
льготные условия кредитования;
яркую и насыщенную корпоративную жизнь.
 Мы ждем тебя, чтобы строить счастливое будущее вместе! "
29654618,Ведущий инженер,Москва,Сбербанк,2019-01-15T10:15:10+0300," Сбербанк объявляет конкурс на позицию Ведущий Инженер в Фабрике данных: сопровождение портала «Супермаркет данных» - внутренней площадки распространения данных в режиме self-service, позволяющей получить данные Банка и их описание для исследования и промышленных процессов
Обязанности: Администрирование и сопровождение портала – 2-я линия сопровождения;
Решение инцидентов, обработка запросов на обслуживание и изменения;
Взаимодействие с разработчиком автоматизированной системы и подразделениями заказчика по вопросам сопровождения портала;
Выполнение работ по настройке и поддержке тестового и промышленного ландшафтов;
Проведение ежедневного мониторинга, подготовка документации (Confluence);
Оперативное реагирование и решение нештатных ситуаций связанных с работой портала.
Организация внедрения и проведение приемо-сдаточных и нагрузочных испытаний портала;
Настройка и поддержка интеграционных решений со смежными системами;
Требования: Высшее техническое образование (желательно в области ИТ);
Знание ITSM, ITIL;
Опыт работы в поддержке АС как 2-я линия сопровождения
Опыт работы с Jira, Confluence;
(желательно) Опыт использования и построения отчётов в Grafana, Zabbix,
(желательно) Опыт использования технологий DevOps и версионного контроля (Jenkins, Bitbucket);
Опыт работы с реляционными БД (Oracle, PostgreSQL) и NoSQL решениями
Опыт работы с BigData решений на стеке технологий Cloudera Hadoop
Опыт работы с Apache Maven
Опыт работы по Agile методологии
Инициативность, коммуникабельность, конструктивность; Функции
Обрабатывает нетиповые запросы пользователей в рамках своего направления и/или кросс-функциональных запросов
Осуществляет мониторинг предоставления ИТ услуги и работу с отклонениями по услуге по своему направлению
Координирует работу по высокоприоритетным инцидентам
Готовит предложения по оптимизации подпроцессов по своему направлению Осуществляет сбор предложений и проводит первичный анализ проблем по своему направлению Выявляет и регистрирует операционные риски по ИТ услугам, выполняет мероприятия по их минимизации Подготавливает и проводит приемо-сдаточных испытаний по ИТ услугам Отвечает за соблюдение целевых значений и КПЭ по этапам процессов и подпроцессам в рамках своего направления Отвечает и выполняет работы по внедрению изменений по предоставляемым ИТ услугам Координирует взаимодействие между подразделениями по своему направлению
Готовит сводные отчеты по инцидентам по своему направлению
Разрабатывает предложения по изменению регламентов по предоставлению услуги в своем направлении Готовит технические регламенты по своему направлению и организует работу по их внедрению Проводит обучение сотрудников по своему направлению Организует работу функциональных подчиненных
 Образование Высшее в Технические науки или Математика Опыт работы Опыт работы (минимальный) 4 года Опыт работы (функциональный) 3-5 лет Профессиональные компетенции
Организация взаимодействия (средний уровень) Согласовывает время, ресурсы, задачи сторон в небольшом

проекте или процессе, учитывая их интересы и существующие процедуры Банка. В случае возникновения сложных ситуаций предлагает варианты решения для согласования с руководством. Организация сервиса (средний уровень) Делает выводы о потребностях клиентов на основании опыта и предлагает решения в рамках существующих процедур и практик Банка. При возникновении нестандартных клиентских проблем предлагает решения для утверждения руководством. Анализ и моделирование данных (средний уровень) Обладает базовой теоретической подготовкой в области ИТ. Знает характеристики систем анализа и моделирования данных. Может самостоятельно выбирать оптимальный вариант решения задач по разработке, доработке или сопровождению таких систем в рамках существующих процедур. Навыки Знание процессов и стандартов Знание информационных систем Интерпретация и презентация данных Управление ИТ-услугами Иностранные языки Сертификаты Условия работы: Условия: Интересные и масштабные задачи!
Место работы: м. Тульская; Даниловский форт
Достойный уровень заработной платы, премия по итогам работы за год
График работы с 9:00 до 18:00, пятница - до 16:45.
Оформление согласно ТК РФ;
Социальный пакет: ДМС, спортзал , возможность обучения за счет компании, льготные условия кредитования.
"
29668611,Главный инженер по сопровождению,Москва,Сбербанк,2019-02-12T09:00:00+0300," Обязанности: · Администрирование и сопровождение решений на базе платформы SAS; · Администрирование и сопровождение решений на базе платформы Pega; · Работа с инцидентами и задачами в рамках 2-го уровня поддержки (управление правами доступа, установка и конфигурирование, troubleshooting); · Установка патчей на производственные среды; · Обеспечение процедур резервирования, вынесение предложений по совершенствованию и модернизации инфраструктуры; оптимизации потоков загрузки данных; · Мониторинг; · Организация и поддержание в актуальном состоянии базы знаний по поддерживаемым системам и сервисам; · Контроль срока действия лицензий и обеспечение их своевременного продления. Навыки: · Знание методологии ITSM и сервисного подхода; · Опыт работы с платформой SAS на уровне администратора; · Понимание типовых архитектур решений SAS; · Знания продуктов SAS DI, LSF, SAS MC и языка программирования SAS BASE (SAS Macro); · Знание SQL; · Навыки и практический опыт написания документации для администраторов и пользователей, описания процессов и архитектуры; · Навыки программирования и написания скриптов (bash/VBS/VBA/powershell/); · Знание Linux; · Личностные качества: активность, системное мышление, умение работать в команде и проактивно найти нужных людей и получить требуемую информацию; · Знание английского языка, достаточное для чтения инструкций. Условия работы: Условия: Место работы: Кутузовский проспект, ул.Поклонная 3А
График работы 5/2;
Оформление согласно ТК РФ;
Социальный пакет: ДМС, спортзал (фитнес, йога), спортивные и культурно-массовые мероприятия, возможность обучения за счет компании, льготные условия кредитования;
Отличная профессиональная команда
Годовая премия "
28868190,Full stack developer,Санкт-Петербург,Сбербанк,2019-03-04T11:40:54+0300," <u>Задачи:</u> поддержка и разработка приложений для обмена информацией с корпоративными клиентами <u>Требования:</u> Желательно, фул-стек (JAVA + UI). Если нет UI, то было бы хорошо, если есть желание развиваться в эту сторону. Для Java:
Опыт работы с технологиями: Spring, Hibernate, Gradle, JMS, JAX-WS, Git,

Jenkins, UML, Junit. Для UI:
Знание JavaScript (es6/es7: классы, импорты декораторы, arrow-функции, генераторы, промисы)
Экспертное знание HTML, HTML5, SASS/LESS/CSS
• Опыт работы с js-фреймовками (например, Backbone, Angular, React, Reflux).
• Опыт работы с Webpack: уметь собрать проект с картинками и стилями, скриптами.
• Опыт работы с REST-сервисами
• Понятие о принципах разработки ПО, ООП, паттернах проектирования
• Умение работать с Git Будет плюсом
• Знание React Опыт работы с Bootstrap, Ionic, Foundation и т.д.
• Умение разрабатывать прототипы UI с помощью, например, Axure, Pixate, Invision, Proto.io и т.д.
<u>Условия:</u> официальное трудоустройство;
возможность профессионального и карьерного развития;
премирование (годовое);
социальный пакет (ДМС; санаторно-курортный отдых и т.д.);
корпоративное обучение;
активная корпоративная жизнь Банка (спортивные мероприятия, внутренние конкурсы и пр.)
Требования Желательно, Full Stack (JAVA + UI). Если отсутствуют компетенции UI, необходимо желание развиваться в данном направлении. Для Java:Опыт работы с технологиями: Spring, Hibernate, Gradle, JMS, JAX-WS, Git, Jenkins, UML, Junit. Для UI:Знание JavaScript (es6/es7: классы, импорты декораторы, arrow-функции, генераторы, промисы) Экспертное знание HTML, HTML5, SASS/LESS/CSS • Опыт работы с js-фреймовками (например, Backbone, Angular, React, Reflux). • Опыт работы с Webpack: уметь собрать проект с картинками и стилями, скриптами. • Опыт работы с REST-сервисами • Понятие о принципах разработки ПО, ООП, паттернах проектирования • Умение работать с Git Будет плюсом• Знание React Опыт работы с Bootstrap, Ionic, Foundation и т.д. • Умение разрабатывать прототипы UI с помощью, например, Axure, Pixate, Invision, Proto.io и т.д. Обязанности Поддержка и разработка приложений для обмена информацией с корпоративными клиентами Условия Интересные и масштабные задачи, работа в команде профессионалов; Возможность обучения и сертификации за счет компании; Крупные и уникальные проекты в развивающейся компании; Официальное оформление по ТК РФ; Рыночный фикс и годовая премия (обсуждается с успешным кандидатом); Социальный пакет + ДМС; Отсутствие строгого дресс-кода; Корпоративная программа лояльности (льготное кредитование, скидки в компаниях-партнерах).
"
29666393,"Аналитик HCM (PA, OM)",Москва,Сбербанк,2019-01-15T15:00:18+0300," Требования: - От 1 года работы в консалтинге - Знание PA, OM, PT - на среднем уровне. Любое знание PY – плюс, но не обязательно. - Понимание HR процессов. Опыт работы в HR подразделении или на соответствующих проектах. - Уверенное знание Excel - Высокие коммуникативные навыки
Обязанности: - Развитие собственного Z-продукта (контроль качества данных в SAP HCM) - Написание документации: спецификации на разработку / настройку, инструкции. - Взаимодействие с бизнесом (консультации по ведению данных, анализ потребностей пользователей) - Взаимодействие с консультантами (по процессам других команд для развития нашего продукта) - Анализ данных SAP HCM и внесение предложений по проверке этих данных - Постановка задач на разработку ABAP-специалистам Условия работы: Мы предлагаем: трудоустройство согласно ТК РФ;
гарантированный доход плюс премиальное вознаграждение;
возможность работать рядом с домом/местом учебы;
регулярное корпоративное обучение;
ДМС, страхование от несчастных случаев и тяжелых заболеваний;
материальную помощь и социальную поддержку, корпоративную пенсионную программу;
льготные условия кредитования;
яркую и насыщенную корпоративную жизнь.
 Мы ждем тебя, чтобы строить счастливое будущее вместе! "

29665306,Дизайнер UX|UI,Москва,Сбербанк,2019-01-15T14:30:26+0300," Роль: дизайнер-проектировщик пользовательского взаимодействия Специализация: web-интерфейс систем корпоративного ДБО Обязанности: Сбор и уточнение требований к продукту совместно с аналитиком; Предложение схемы пользовательского взаимодействия, обсуждение с заинтересованными сторонами; Перевод схемы в дизайн, работа с готовыми компонентами дизайн-системы, разработка недостающих; Обсуждение и уточнение механики пользовательского взаимодействия с разработчиками; Подготовка интерактивных прототипов в случае необходимости; Документирование задачи на канбан-доске. Технические компетенции: Понимание принципов и технологий разработки пользовательских интерфейсов; Системное мышление, способность видеть продукт частью системы; Экспертиза в проектировании взаимодействия, визуальном дизайне, юзабилити; Знание композиции, типографики; создание прототипов высокой степени детализации; Уверенное владение sketch, InVision; Умение пользоваться Photoshop, Axure, Illustrator. Дополнительные требования: Доказанный опыт (портфолио) в дизайне информационных веб-приложений. Мы предлагаем: Работа среди профессионалов банковского дела; Профессиональное обучение и развитие; Мотивирующая система оплаты труда; Расширенный социальный пакет, корпоративные льготы (в т.ч. по внутренним программам кредитования и программам лояльности партнёров); Участие в интересных и инновационных проектах; Отличные возможности для самореализации, результаты, которыми можно гордиться; Высокий уровень корпоративной культуры; Отличный новый офис «Sbergile Home» в шаговой доступности от ст. Кутузовская (метро и МЦК); Многофункциональный спортивный зал (бесплатно); Вместительный подземный паркинг. "
29664693,Разработчик Big Data,Иннополис,Сбербанк,2019-01-15T14:15:14+0300," Проект-разработка сервиса Доставки Данных. Цель проекта – написание фреймворка по управление заданиями на кластере. Работая в данном проекте, разработчик получит возможность использовать в своей работе самые передовые технологии и фреймоворки BigData и стать одним из разработчиков такого фреймворка. Функции: Разработка новых фич сервиса Доставки Данных (ДД).
Работа с дефектами ПО сервиса Доставки Данных (выявление причин, исправление, оценки, консультации).
Консультации по внутренней работе сервиса Доставки Данных.
Участие во всех командных мероприятиях (планирование, ретроспектива, stand-up, демо).
Взаимодействие с разработчиками и другими специалистами по тестированию, проектными и продуктовыми менеджерами. Требования: Опыт разработки на Java / Scala от 3 лет.
Знание Linux на уровне продвинутого пользователя.
Знание платформы Hadoop (HDFS, Spark, HBase, MapReduce, etc.).
Опыт работы с Git, Bitbucket.
Знание жизненного цикла разработки ПО.
Знание SQL.
Опыт работы с bugtracking системами и тестовой документацией. Приветствуются знания / опыт: Опыт работы с Jenkins или любым другим аналогичным инструментом.
Знание языков Python и/или Scala.
Умение

```
работать по гибким методологиям разработки.</li><li>Знание современных
инструментов devops (Jenkins).</li></ul><div></div>"
```

6.3, 6.4 Улучшить качество базовой предсказательной CRF-модели на тестовой выборке за счет добавления и модификации признаков

Загрузка данных:

```
! wget https://www.dropbox.com/s/iuwsx5pmfhkk0w2/ner_dataset.csv
```

Baseline для 6.3

```
import pandas as pd

data = pd.read_csv("ner_dataset.csv", encoding="latin1")
data = data.fillna(method="ffill")

agg_func = lambda s: [[w, p, t] for w, p, t in
zip(s["Word"].values.tolist(),
                                                    s["POS"].values.tolist(),

 s["Tag"].values.tolist())]
grouped = data.groupby("Sentence #").apply(agg_func).values.tolist()

X_list = [[word[:2] for word in sentence] for sentence in grouped]
y_list = [[word[2] for word in sentence] for sentence in grouped]
from sklearn.model_selection import train_test_split

data_train, data_test, y_train, y_test = train_test_split(X_list, y_list,
test_size=0.2, random_state=1337)
def word2features(sent, i):
    word = sent[i][0]
    postag = sent[i][1]

    features = {
        'word.lower()': word.lower(),
        'word.isupper()': word.isupper(),
        'word.istitle()': word.istitle(),
        'word.isdigit()': word.isdigit(),
        'postag': postag,
        ### Your code goes here ###
    }
    if i > 0:
        word1 = sent[i-1][0]
        postag1 = sent[i-1][1]
        features.update({
            '-1:word.lower()': word1.lower(),
            '-1:word.istitle()': word1.istitle(),
            '-1:word.isupper()': word1.isupper(),
            ### Your code goes here ###
        })
    else:
        features['BOS'] = True
```

```
    if i < len(sent)-1:
        word1 = sent[i+1][0]
        postag1 = sent[i+1][1]
        features.update({
            '+1:word.lower()': word1.lower(),
            '+1:word.istitle()': word1.istitle(),
            '+1:word.isupper()': word1.isupper(),
            ### Your code goes here ###
        })
    else:
        features['EOS'] = True

    return features

def sent2features(sent):
    return [word2features(sent, i) for i in range(len(sent))]
X_train = [sent2features(s) for s in data_train]
X_test = [sent2features(s) for s in data_test]
### Your code goes here ###
# Инициализировать и обучить модель, оценить её качество, прокомментировать,
получается ли лучшее качество с вашими признаками или без них и почему.
```

Контрольные вопросы

1 Основные методы извлечения информации из текстов.
2 Разметки последовательностей в NER
3 Какие нейронные сети для распознавания именованных сущностей вы знаете?
4 Методы оценки качества моделей в ИИ?

Требования к оформлению отчета.

Отчет по ЛР состоит из

а) Титульный лист:название ЛР, ФИО студента и номер группы, № варианта.

б)Текст задания;

в) Исходный текст программы;

г) Скрины выполнения;

д) Выводы.

Критерии оценивания.

За решение каждой задачи вы можете получить до 2 баллов: задание выполнено полностью и правильно -2 балла; имеются незначительные ошибки - 1 балл; приложение не работоспособно – 0 баллов;

Ответ на контрольный вопрос должен продемонстрировать понимание механизмов NLP, за ответ вы можете оценку: ответ полный и правильный -2 балла;ответ не полный – 1 балл; ответ не удовлетворительный – 0 баллов.

Итого, максимальная оценка – 10 баллов

Внимание! Полученная оценка автматически снижается на 2% за каждую полную неделю задержки сдачи отчета по работе, но не более, чем на 40%

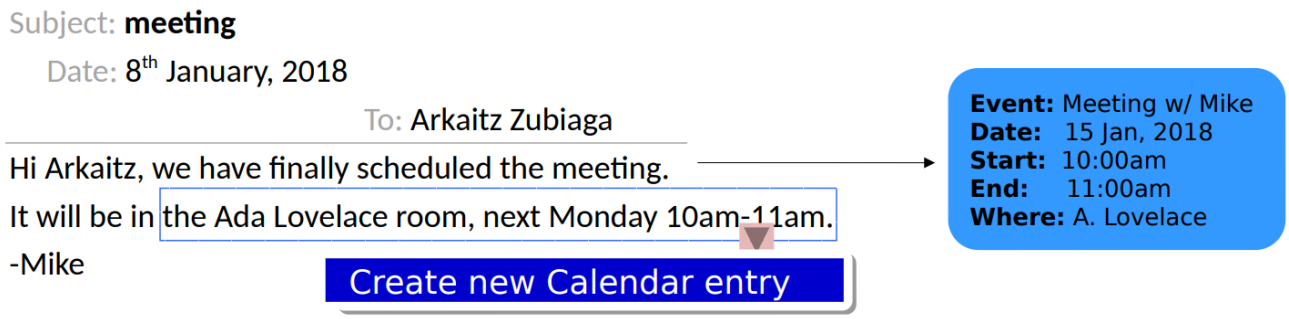
Библиографический список

1. Ботов, Д. С. Извлечение информации с использованием нейросетевых моделей языка на примере анализа вакансий в системах онлайн-рекрутмента / Д. С. Ботов, Ю. Д. Кленин, И. Е. Николаев // ВЕСТНИК ЮГОРСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА 2018 г. Выпуск 3 (50). С. 37–48. – URL: https://cyberleninka.ru/article/n/izvlechenie-informatsii-s-ispolzovaniem-neyrosetevyh-modeley-yazyka-na-primere-analiza-vakansiy-v-sistemah-onlayn-rekrutmenta
2. Чупин В.О., Метод выявления и анализа тональности наиболее обсуждаемых тем в социальных сетях /В.О.,Чупин, Т.Ю.Оленчикова // Южно-Уральская молодежная школа по математическому моделированию: сб. тр. VI Всеросс. студ. науч.-практ. конф. – Челябинск: Издательский центр ЮУрГУ, 2023., С.101-109/ – URL: https://elib.susu.ru/cgi-bin/koha/opac-detail.pl?biblionumber=489422&query_desc=%D0%A3%D1%80%D0%B0%D0%BB%D1%8C%D1%81%D0%BA%D0%B0%D1%8F%20%D0%BC%D0%BE%D0%BB%D0%BE%D0%B4%D0%B5%D0%B6%D0%BD%D0%B0%D1%8F%20%D1%88%D0%BA%D0%BE%D0%BB%D0%B0

Приложение 1

Введение в извлечение сущностей из текста и NER

Извлечение информации означает создание структурированных данных из неструктурированного текста. На практике задача может выглядеть так: нужно автоматически создать запись в календаре исходя из текста письма, как на рисунке ниже.



Статьи Википедии, например, включают оба вида информации – окно со структурированным текстом справа и неструктурированный текст слева:

Используя техники извлечения информации, мы можем автоматически заполнить это окно необходимыми данными. При этом мы извлекаем из текста ограниченный объем его семантического содержимого. Можно рассматривать этот процесс как автоматическое заполнение опросника или шаблона (к примеру, таблицы базы данных).

Извлечение информации стало важной задачей обработки естественного языка еще в 90-е годы и сейчас делится на несколько актуальных подзадач:

Извлечение именованных сущностей (Named Entity Recognition, или NER).
Извлечение отношений.
Извлечение темпоральных выражений.
Разрешение кореференции.
Извлечение событий.
Заполнение слотов.
Связывание сущностей.

Мы рассмотрим первый пункт.

Извлечение именованных сущностей

Как правило, это имена собственные. Обычно выделяют три категории именованных сущностей – люди, места и организации, но в специализированных решениях часто рассматриваются и другие категории, такие как названия продуктов и произведений искусства. Также к именованным сущностям часто относят и численные выражения (цены, даты, время).

Вот пример расширенного списка:

Type	Tag	Sample Categories
People	PER	Individuals, fictional characters, small groups
Organization	ORG	Companies, agencies, political parties, religious groups, sports teams
Location	LOC	Physical extents, mountains, lakes, seas
Geo-Political Entity	GPE	Countries, states, provinces, counties
Facility	FAC	Bridges, buildings, airports
Vehicles	VEH	Planes, trains, and automobiles

Задача извлечения именованных сущностей сводится к выявлению отрезков текста, которые являются именами собственными, и их категоризации по типу сущности. Результаты решения можно в дальнейшем использовать для снижения разреженности данных при классификации текста, для выявления объекта в рамках анализа тональности и в вопросно-ответных системах.

Но вернемся к NER. Какие тут встречаются проблемы?

Первая – это неоднозначность сегментации. Например, является ли данное слово сущностью и, если да, то где границы этой сущности? Вот как здесь:

I'm on the Birmingham New Street-London Euston train.
location location

Вторая же проблема – неоднозначность типа сущности: к примеру, *Georgia* в английском – это имя или страна?

NER как задача разметки последовательностей

Стандартный подход к NER – использование пословной разметки последовательности слов. В этой задаче первый шаг – расставить метки по BIO-нотации: B-метка (beginning) проставляется для обозначения начала интересующей нас сущности, I (inside) – для обозначения слова внутри нее, а O (outside) – это любое слово за ее пределами. Вот предложение для примера:

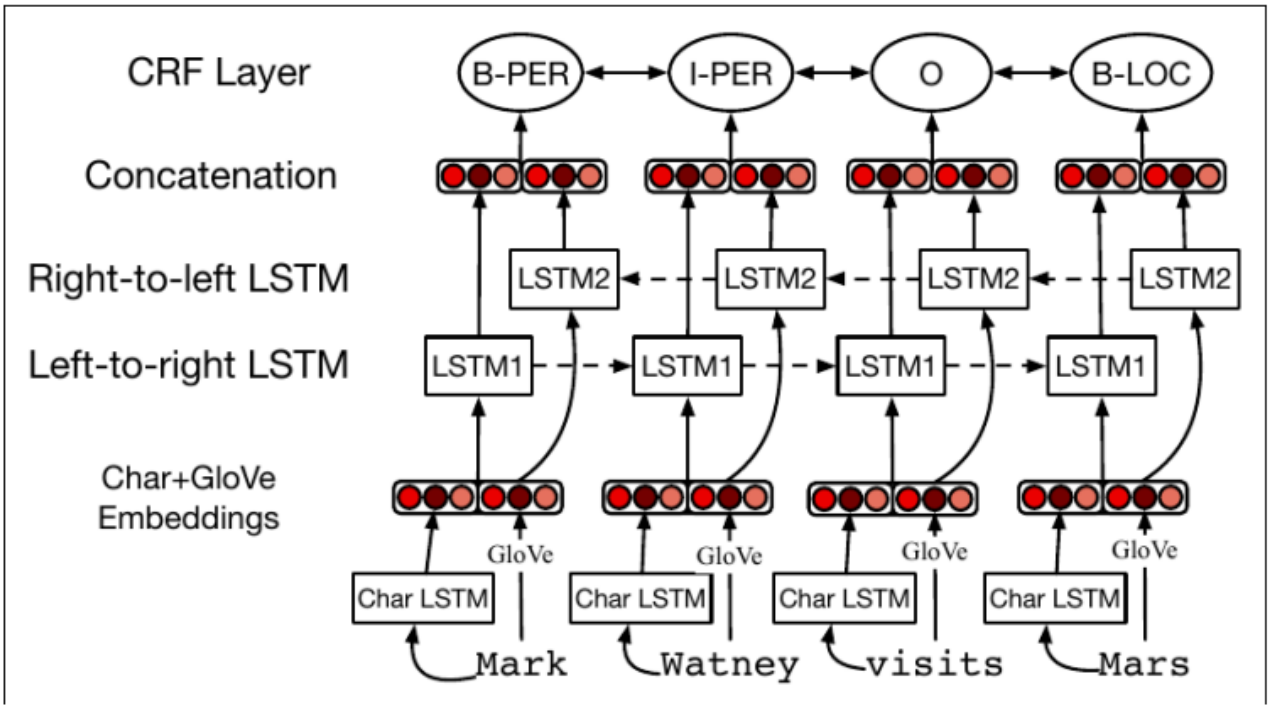
[$_{ORG}$ American Airlines], a unit of [$_{ORG}$ AMR Corp.], immediately matched the move, spokesman [$_{PERS}$ Tim Wagner] said.

А вот так выглядит разметка в нотации BIO:

Words	BIO Label	IO Label
American	B-ORG	I-ORG
Airlines	I-ORG	I-ORG
,	O	O
a	O	O
unit	O	O
of	O	O
AMR	B-ORG	I-ORG
Corp.	I-ORG	I-ORG
,	O	O
immediately	O	O
matched	O	O
the	O	O
move	O	O
,	O	O
spokesman	O	O
Tim	B-PER	I-PER
Wagner	I-PER	I-PER
said	O	O
.	O	O

Такая нотация позволяет нам выделить в тексте границы именованных сущностей и их тип.

Модель, которая лучше всего справляется с задачей распознавания именованных сущностей, – Bi-LSTM. Это двунаправленная LSTM, которая сочетает в себе две нейронных сети, одна из которых работает справа налево, а вторая идет в обратном направлении – слева направо, захватывая таким образом весь контекст предложения.



В задаче распознавания именованных сущностей обучается модель MEMM (Maximum-Entropy Markov Model) или CRF (Conditional Random Field), которая определяет, является ли токен частью сущности или нет. Для присвоения меток этим токенам часто используются газетиры – многомиллионные готовые

списки сущностей, такие как GeoNames или National Street Gazetteer (для Великобритании).

Оценка качества

Что касается оценки качества моделей, то тут используются стандартные метрики – точность, полнота и F-мера. Единицей для расчета этих метрик считается сущность, а не слово, то есть точность определяется как отношение количества правильно распознанных сущностей и количества распознанных сущностей, а полнота – как отношение правильно распознанных сущностей и количества сущностей в золотом стандарте. F-мера объединяет в себе две предыдущие метрики.

Из-за сегментации оценка качества модели осложняется, поскольку при обучении моделей мы используем слова, а для оценки – целые сущности. По этой причине если в нашем корпусе есть выражение *Leamington Spa*, а система определяет только *Leamington* – это уже ошибка.

Коммерческие системы распознавания сущностей

В научной работе обычно используются статистические модели для работы с последовательностями. Коммерческие же системы извлечения именованных сущностей часто гибридные и основаны на правилах и методах машинного обучения. Как правило, делается несколько проходов по тексту, и результаты первого прохода отражаются на следующем. Сначала для разметки однозначных сущностей в таких системах используются правила, обеспечивающие высокую точность, потом осуществляется поиск соответствий в части уже выявленных сущностей. Для идентификации прочих сущностей используются специальные списки из интересующей нас области и техники вероятностной разметки последовательностей, также использующие метки с предыдущих стадий.

Источники:

- https://habr.com/ru/articles/675220/
- Васильев Ю. Обработка естественного языка Python и SPACY на практике. – 2021 http://lab314.brsu.by/kmp-lite/kmp2/2019/sum/NLP-BOOK/Vasilev_OBRABOTKA-ESTESTVENNOGO-YaZYKA-PYTHON-I-SPACY.640612.pdf

Вопросы к зачету

по дисциплине **«Интеллектуальный анализ текстов»**

направление 01.03.02 Прикладная математика и информатика (Бакалавр)

Профиль/специализация Прикладная математика и искусственный интеллект

1. Что такое регулярные выражения? Для чего они применяются?
2. Компиляция регулярных выражений
3. Функции Python для работы с регулярными выражениями.
4. Какие существуют операции над строками в языке Python?
5. Какие существуют методы обработки строк?
6. Преимущества и недостатки использования для обработки строк методов и регулярных выражений.
7. Что такое сегментация слов.
8. Проблемы токенизации слов в тексте.
9. Сегментация предложений в тексте. Бинарный классификатор выделения предложений.
10. Нормализация и лемматизация слов.
11. Стемминг текста. Особенности использования стеммера Портера
12. Токенизация слов с помощью библиотеки nltk python. Функции модуля nltk.tokenize.
13. Использование регулярних выражений для выделения токенов.
14. Каким методом объекта класса MorphAnalyzer() можно получить морфологический разбор заданного слова? Что является параметром данного метода?
15. Что собой представляет морфологический разбор, возвращаемый данным методом?
16. Какими свойствами и методами обладает полученный морфологический разбор?
17. Как получить тег морфологического разбора? Что собой представляет полученный тег?
18. Как получить нормальную форму слова?
19. Что подразумевается под понятием “граммема” в pymorphy2? Как получить значение граммемы для конкретного морфологического разбора?
20. Как получить кириллические названия граммем?
21. Для чего используется свойство грамматического разбора score?
22. Как определяется параметр score в морфологическом разборе pymorphy2?
23. Основные способы формализации синтаксической структуры предложения.
24. Деревья синтаксической зависимости. Проективные деревья синтаксической зависимости. Размеченные деревья синтаксической зависимости.
25. Формальное определение грамматики. Контекстно-свободные грамматики (КСГ).
26. Синтаксический анализ. Задача синтаксического анализа.
27. Дерево разбора для КСГ
28. Использование контекстно-свободных грамматик для реализации размеченных систем составляющих.
29. Неоднозначные контекстно-свободные грамматики и синтаксическая омонимия.
30. Показать многозначность контекстно-свободной грамматики, распознающей следующие предложения:
    - Japanese eat sushi with the hands.
    - Japanese eat sushi with pickled ginger.
31. Показать два способа формального выражения синтаксической структуры следующих предложений:
    - Дипломная работа студента включает четыре раздела.
    - The small boy put the tortoise on the colored rug.

Пример билета на зачет

Челябинский государственный университет

Кафедра **Теории управления и оптимизации**
Специальность: **01.03.02 Прикладная математика и информатика**
Дисциплина: **Интеллектуальный анализ текстов**

Билет № 6

1. Токенизация слов с помощью библиотеки nltk python. Функции модуля nltk.tokenize.
1. Показать два способа формального выражения синтаксической структуры следующих предложений:
   - Дипломная работа студента включает четыре раздела.
   - The small boy put the tortoise on the colored rug.

Преподаватель _____
(подпись)

Заведующий кафедрой _____
(подпись)

01.03.02 Прикладная математика и искусственный интеллект, направление подготовки Прикладная математика и информатика, РПД по дисциплине "Интеллектуальный анализ текстов", 2024 года набора, очная форма обучения

Проректор по учебной работе	утверждено	В.Е. Федоров

Ученым советом математического факультета

Протокол заседания № 3 от 19.10.2023

Председатель Ученого совета математического факультета	согласовано	Е.А. Сбродова

Заседанием кафедры теории управления и оптимизации

Протокол заседания № 4 от 12.10.2023

Заведующий кафедрой	согласовано	И. В. Изместьев
Автор (составитель)		Е. С. Красильникова

Структура рабочей программы соответствует приказу ректора ФГБОУ ВО «ЧелГУ» от «13» апреля 2021 г. № 247-1