Министерство науки и высшего образования Российской Федерации

Национальный исследовательский университет «МИЭТ»

Л.Г. Гагарина, А.В. Городилов, А.И. Кононова

Лабораторный практикум по курсу «Математическая лингвистика и обработка естественных языков»

Утверждено редакционно-издательским советом университета

Москва 2024

Введение

Учебное пособие предназначено для студентов, изучающих дисциплину «Математическая лингвистика и обработка естественных языков».

Пособие включает в себя задания, выполнение которых позволит получить навыки работы с современным инструментарием работы с естественными языками — библиотеками, языковыми средствами, фреймворками. Но основное внимание уделяется не освоению конкретных средств, а методов и алгоритмов, которые в настоящий момент широко используются и реализованы в инструментарии. Студентам и преподавателям рекомендуется проявлять творчество и свободу в выборе средств для выполнения заданий.

К обработке естественных языков есть несколько актуальных подходов, каждый из которых имеет свои сильные и слабые стороны и область применения. В данном пособии рассматриваются: математическая лингвистика, классическое машинное обучение и глубокое обучение, в том числе нейронные сети.

Выполнение заданий позволит на практике увидеть сильные и слабые стороны каждой из рассматриваемых методик, что в дальнейшем позволит делать разумный и обоснованный выбор методов для решения практических задач.

Данное пособие оставляет студентам возможности проявлять творческий подход и экспериментировать с предложенными задачами и реализацией решений. Студентам рекомендуется активно и смело этим пользоваться, а преподавателям — поощрять это. Детальные рекомендации по работе с пособием и оцениваю работ даны в заключительной части.

Общие теоретические сведения

Обработка естественных языков (NLP) включает в себя анализ текста и генерацию текстов. В лабораторных работах подробно рассматриваются технологии, используемые для анализа и генерации.

Курс опирается на знания, получаемые в других дисциплинах прогрмм подготовки магистров и бакалавров. Связь с другими дисциплинами описана в приложении 2. Рекомендуется знакомиться с данным приложением, если вы изучаете дисциплину не в рамках магистерской программы, или если курс кажется сложным или непонятным.

Основные методы и технологии NLP можно условно разделить на три группы: математическая лингвистика, классическое машинное обучение и глубокое обучение.

Математическая лингвистика

Это междисциплинарная область, которая объединяет методы и концепции математики и лингвистики для изучения и моделирования языковых явлений. Цель этой науки заключается в создании формальных моделей, которые могут объяснить структуры и процессы, характерные для естественных языков. Математическая лингвистика находит применение как в теоретических, так и в прикладных исследованиях.

Основные направления математической лингвистики это:

Формальные грамматики: Эти грамматики используются для описания синтаксических структур языков. Самые известные виды — это контекстно-свободные грамматики, которые применяются для моделирования структуры предложений в естественных языках.

Автоматы и языки: Изучение формальных языков и автоматов, таких как конечные автоматы и стековые автоматы, помогает понять, как можно распознавать и порождать синтаксически правильные последовательности.

Энтропия и избыточность: Анализ текстов с точки зрения теории информации позволяет измерять энтропию и избыточность языков. Эти понятия помогают понять, как информация передается и как она может быть сжата без потери смысла.

Кодирование и передача информации: Изучение различных методов кодирования информации для оптимальной передачи текстов и данных.

Корпусная лингвистика: Использование больших корпусов текстов для анализа частотности слов и грамматических конструкций.

Модели на основе вероятностей: Байесовские сети, скрытые марковские модели и другие статистические модели используются для анализа и предсказания языковых структур.

Алгоритмы синтаксического анализа: Разработка эффективных алгоритмов для синтаксического анализа предложений, таких как алгоритмы на основе деревьев и графов.

Математическая лингвистика имеет широкий спектр применений в различных областях, в том числе в разработке систем обработки естественного языка, включая автоматические переводчики, системы синтеза и распознавания речи, а также интеллектуальные поисковые системы. В

лингвистических исследованиях она помогает создавать формальные описания языков и разрабатывать теории грамматики и семантики.

Машинное обучение

Методы машинного обучения позволили значительно расширить возможности обработки текста, и их развитие привело к окончанию «зимы искусственного интеллекта» - периода, когда наступило разочарование в искусственном интеллекте, и развитие этой отрасли замедлилось.

Машинное обучение позволило эффективно решать следующие задачи, для каждой из которых подходят различные методы:

Анализ тональности и мнений: используются логистическая регрессия, SVM и наивный Байесовский классификатор для определения эмоциональной окраски текста.

Тематическое моделирование: применение методов кластеризации и разложения матриц для выявления скрытых тем в больших корпусах текстов.

Идентификация и разметка сущностей: скрытые марковские модели и логистическая регрессия используются для выделения именованных сущностей и теггинга частей речи.

Классификация текста: использование наивного Байесовского классификатора, SVM и деревьев решений для распределения текстов по категориям.

Снижение размерности и визуализация данных: разложение матриц и методы понижения размерности (PCA, t-SNE) помогают в анализе и визуализации текстовых данных.

Таким образом, даже без нейронных сетей существует множество инструментов и подходов для обработки и анализа естественного языка. Эти методы могут быть оптимальными для решения множества задач.

Глубокое обучение

Рекуррентные нейронные сети (RNN) предназначены для моделирования последовательных данных, таких как текст, и позволяют учитывать контекст. В этой категории выделяются LSTM (Long Short-Term Memory), которые способны запоминать информацию на долгосрочной основе и эффективно обрабатывать длинные последовательности, а также GRU (Gated Recurrent Unit), упрощенная версия LSTM, требующая меньше вычислительных ресурсов, но также хорошо работающая с длинными последовательностями.

Сверточные нейронные сети (CNN) используются для извлечения локальных признаков и паттернов из текстов, что важно для задач классификации и анализа. Они адаптированы для обработки текстовых данных, позволяя эффективно выделять ключевые признаки на уровне слов и фраз.

Модели внимания (Attention) улучшают способность модели фокусироваться на значимых частях входных данных. В этой области выделяются Self-Attention, использующий внимание внутри одной последовательности для выделения важных элементов контекста, и Bahdanau Attention, позволяющий модели фокусироваться на соответствующих частях входного текста при генерации выходного.

Трансформеры позволяют обрабатывать последовательности данных параллельно и более эффективно, чем RNN. Ключевая архитектура - Transformer, основанная на механизме внимания, эффективно обрабатывающая длинные последовательности и улавливающая сложные зависимости. Среди трансформеров выделяются BERT (Bidirectional Encoder Representations from Transformers), предобученная модель, которая извлекает двунаправленные контекстуальные представления слов, и GPT (Generative Pre-trained Transformer), предназначенная для генерации текста и способная создавать осмысленные и связные тексты на основе заданного контекста.

Генеративные модели, такие как Variational Autoencoders (VAE) и Generative Adversarial Networks (GAN), используются для создания новых текстов или дополнения существующих. VAE применяют вероятностный подход для генерации данных, тогда как GAN обучают генератор создавать данные, способные обмануть дискриминатор, что способствует созданию более реалистичных текстов.

Предобученные языковые модели, такие как ULMFiT (Universal Language Model Fine-tuning), позволяют адаптировать предобученные модели для специфических задач. Среди них выделяются модели OpenAI GPT и GPT-2, GPT-3, обученные на огромных объемах текстов и способные решать различные задачи генерации и анализа текста.

Как уже было сказано выше, эти методы используются для двух основных задач — анализа текста и генерации текста. Результаты анализа и генерации находят применение в большом количестве прикладных задач.

Анализ текста

Анализ текста используется для множества целей в различных областях, включая бизнес, науку, здравоохранение, медиа и многое другое.

Одной из ключевых целей анализа текста является определение настроений и мнений. Это позволяет понять эмоциональную окраску текста, будь то позитивная, негативная или нейтральная. Такой анализ активно применяется для изучения отзывов клиентов, соцсетей, комментариев и опросов, что помогает компаниям и организациям понимать общественное мнение о своих продуктах, услугах или событиях.

Еще одной важной задачей является извлечение информации, которое позволяет автоматически выделять ключевые данные из больших объемов текста. Этот метод используется для идентификации значимых деталей, таких как имена людей, названия организаций, географические места, даты и числа в новостях, статьях и документах.

Классификация текста позволяет автоматически распределять тексты по категориям. Это полезно для сортировки электронной почты (например, определение спама), классификации новостных статей по темам и организации контента на веб-сайтах.

Суммаризация текста создаёт краткие версии длинных текстов, сохраняя основное содержание. Эта техника применяется для генерации кратких отчетов, создания анонсов и резюме статей или документов, что помогает пользователям быстро ознакомиться с большим объемом информации.

Поиск информации и ответы на вопросы помогают быстро находить релевантную информацию в текстовых данных. Такие методы используются в поисковых системах, системах вопросов и ответов (например, Siri, Google Assistant), базах знаний и системах поддержки клиентов.

Тематическое моделирование определяет скрытые темы и паттерны в текстах. Это полезно для анализа больших текстовых корпусов, таких как исследование научных публикаций, анализ отзывов пользователей и выявление трендов в социальных медиа.

Распознавание речи и синтез текста включают преобразование устной речи в текст и наоборот. Эти технологии используются в голосовых ассистентах, системах управления голосом, для автоматических субтитров и переводов, а также создания аудиокниг и других мультимедийных продуктов.

Обнаружение аномалий и мошенничества помогает выявлять необычные или подозрительные паттерны в текстовых данных. Это важно для обнаружения мошенничества в банковских транзакциях, анализа сетевого трафика для выявления кибератак и мониторинга социальных медиа для выявления экстремистского контента.

Диалоговые системы и чат-боты создаются для ведения осмысленных диалогов с пользователями. Они применяются в виртуальных ассистентах, чат-ботах для поддержки клиентов, интерактивных обучающих системах и системах автоматизации задач.

Таким образом, анализ текста предоставляет инструменты для понимания, обработки и использования текстовой информации, что делает его незаменимым в современном мире, где объемы текстовых данных постоянно растут.

Генерация текста

Принципы генерации текста изучают с целью развития и совершенствования различных областей и технологий, таких как:

Автоматизация общения:

Чат-боты и виртуальные ассистенты: Улучшение взаимодействия с пользователями, предоставление ответов на запросы, помощь в выполнении задач и решении проблем.

Клиентская поддержка: Автоматизированные системы могут обрабатывать запросы клиентов, предоставлять решения типичных проблем и передавать сложные случаи человеческим операторам.

Контент-генерация:

Создание текстов: Генерация новостных статей, отчетов, резюме и другого контента, что может сократить время и затраты на производство текстовых материалов.

Креативное письмо: Генерация сценариев, историй, стихов и других форм художественного текста.

Образование и обучение:

Интерактивные учебные материалы: Создание адаптивных учебных пособий и материалов, подстраивающихся под уровень знаний и интересы учащихся.

Поддержка преподавателей: Генерация тестовых заданий, конспектов лекций и другой вспомогательной информации.

Перевод и локализация:

Машинный перевод: Улучшение качества автоматического перевода текстов с одного языка на другой, что способствует глобальной коммуникации и доступности информации.

Анализ данных и прогнозирование:

Системы рекомендаций: Персонализация контента и рекомендаций на основе анализа текстовых данных и предпочтений пользователей.

Социальные сети и маркетинг:

Создание рекламных кампаний: Автоматическая генерация текстов для рекламных сообщений, постов и других маркетинговых материалов.

Лабораторная работа №1. Регулярные выражения - поиск

Цель работы:

научиться составлять шаблоны с использованием синтаксиса регулярных выражений и использовать регулярные выражения для поиска в тексте.

Теоретические основы

Поиск по тексту является часто используемой функцией при обработке текста на всех уровнях и для любых целей. Практически все текстовые редакторы имеют встроенную функцию поиска по тексту, которыми читателю наверняка приходилось пользоваться. Также во многих языках программирования высокого уровня есть функции поиска подстроки в строке.

Тем не менее поиск по целой подстроке является очень ограниченным инструментом. Например, нельзя произвести поиск сразу по двум словам — приходится сначала искать по одному, затем по другому слову и объединять результаты. Нельзя искать фрагменты текста, которые имеют известное начало и конец, но разные середины. Можно привести ещё много примеров того, когда поиск по целой подстроке не является удобным. Тем не менее им всё равно приходиться пользоваться, но результаты поиска надо дополнительно обрабатывать либо с помощью дополнительного программного кода, либо вручную.

Существует распространённый инструмент, расширяющий возможности поиска — поиск по маске. В маске знак «*» обозначает любое количество

любых символов, а знак «?» - любой один символ. Поиск по маске имеет очень простые правила, что позволяет его использовать неспециалистам, но его возможности всё ещё очень ограниченны.

Регулярные выражения также являются способом расширить возможности поиска. Они используют более сложный синтаксис, чем поиск по маске. Тем не менее они широко распространены, их синтаксис является де-факто стандартом. Стоит отметить, что регулярные выражения также используют символы «*» и «?», но они имеют другой смысл. Изучающие регулярные выражения часто путаются в значении этих символов из-за их использования в поиске по маске. Чтобы избежать данной ошибки рекомендуется освоить эти символы и их функционал в первую очередь.

Регулярные выражения имеют много возможностей и правил. Однако знание даже небольшой части возможностей и синтаксиса позволяет начать их эффективно использовать. Это делает кривую изучения регулярных выражений очень пологой. Не следует их бояться.

Правила, с которых рекомендуется начать изучение:

Знак «*» используется после символа, и означает, что этот символ может повторяться в искомом фрагменте любое количество раз, в том числе 0 раз.

Например регулярное выражение «сан*ый» будет находить слова «санный», «саный», «саый» и «саннннный». Обратите внимание, что * вовсе не означает, что между «н» и «ы» могут быть любые символы. То есть слово «сановый» не будет находиться с помощью регулярного выражения «сан*ый»,

но находится с помощью маски «сан*ый». Обязательно достигните осознания причин этого, прежде чем читать дальше.

Знак «+» означает, что символ перед ним встречается один или более раз, а знак «?» - ноль или один раз. Выражение «сл+он» будет находить слова «слон» и «слллон», но не будет находить слово «сон». Выражение «сл?он» будет находить слова «слон» и «сон», но не будет находить «сллон».

Фигурные скобки с числом в них означают, что символ встречается ровно столько раз, сколько написано скобках. Также в фигурных скобках может стоять диапазон: минимальное и максимальное число повторов, разделенных дефисом. Например выражение «длин{1-2}а» будет находить слова «длинна» и «длина», но не будет находить «длиннна».

Квадратные скобки означают один из символов внутри скобок. Например, выражение «с[тл]он» будет искать слова «стон» и «слон», но не будет искать слова «стлон» или «слтон». В квадратных скобках также можно указать диапазон — в таком случае будут включены все символы, которые идут между началом и концом по алфавиту (или точнее — в кодировке).

Любой из упомянутых и других специальных символов может быть экранирован «\», чтобы снять с него специальный смысл, например если надо искать подстроку со скобками или знаком +.

Рекомендуется попробовать указанные правила, и затем продолжить изучение других правил.

Регулярные выражения, регулярные грамматики и конечные автоматы тесно связаны между собой, так как все они описывают одни и те же классы языков, известные как регулярные языки. Регулярные выражения предоставляют синтаксис для определения шаблонов, которые соответствуют множествам строк, используя символы, операции объединения, конкатенации и

замыкания Клини. Каждое регулярное выражение может быть преобразовано в конечный автомат (НКА), который распознает тот же язык. В свою очередь, любой конечный автомат может быть преобразован в эквивалентное регулярное выражение.

Конечные автоматы бывают двух типов: детерминированные и недетерминированные. Детерминированные конечные автоматы однозначно определяют переход в следующее состояние для каждого входного символа, тогда как недетерминированные конечные автоматы могут иметь несколько возможных переходов. Несмотря на различия, любой недетерминированный конечный автомат может быть преобразован в эквивалентный детерминированный конечный автомат, что подчеркивает их эквивалентность в выразительной мощности.

Регулярные грамматики представляют собой другой способ описания регулярных языков с помощью правил, определяющих, как строки могут быть образованы из начального символа. Существуют праволинейные и леволинейные регулярные грамматики, которые могут быть преобразованы друг в друга и в конечные автоматы. Это значит, что любая регулярная грамматика может быть преобразована в конечный автомат, распознающий тот же язык, и наоборот.

Основное ограничение этих инструментов заключается в том, что они могут описывать только регулярные языки, которые являются наименее

мощным классом языков в иерархии Хомского. Регулярные языки не могут включать конструкции с вложенными или рекурсивными структурами, такие как правильно сбалансированные скобки или синтаксис языков программирования. Для описания таких более сложных языков требуются более мощные формализмы, такие как контекстно-свободные грамматики и соответствующие автоматы — стековые автоматы.

Таким образом, регулярные выражения, регулярные грамматики и конечные автоматы являются взаимосвязанными инструментами, ограниченными в своей выразительной способности только регулярными языками, но все они играют важную роль в теории формальных языков и автоматов.

Задание на лабораторную работу 1:

1. Изучить синтаксис регулярных выражений.

2. Потренироваться в использовании регулярных выражений, решая задания tutorial и beginner на сайте https://regexcrossword.com/

3. Написать регулярные выражения для одной из следующих задач. Номер варианта выбрать по первой букве фамилии в соответствии с таблицей:

Первая буква фамилии	Вариант
А-Г	1
Е-К	2

Л-О	3
П-У	4
У-Ц	5
Ч-Я	6

Вариант 1: Проверка адреса электронной почты

Напишите регулярное выражение, которое проверяет, является ли строка корректным адресом электронной почты.

Простой вариант: проверяет наличие символа @ и точки . в правильных позициях.

Сложный вариант: включает дополнительные правила для проверки, что email начинается с буквы или цифры и не содержит специальных символов в домене.

Вариант 2: Поиск и замена телефонных номеров

Напишите регулярное выражение, которое ищет телефонные номера в формате XXX-XXX-XXXX и заменяет их на (XXX) XXX-XXXX.

Пример:

Входная строка: "Мой номер телефона 123-456-7890."

Выходная строка: "Мой номер телефона (123) 456-7890."

Вариант 3: Валидация пароля

Напишите регулярное выражение для проверки паролей. Пароль должен содержать:

Минимум 8 символов

Как минимум одну заглавную букву

Как минимум одну строчную букву

Как минимум одну цифру

Вариант 4: Извлечение дат из текста

Напишите регулярное выражение для извлечения дат в формате DD-MM-YYYY из текста.

Пример:

Входная строка: "Сегодня 31-05-2023, а завтра будет 01-06-2023."

Выход: ["31-05-2023", "01-06-2023"]

Вариант 5: Проверка URL

Напишите регулярное выражение для проверки корректности URL. URL должен начинаться с http:// или https://, содержать доменное имя и может иметь путь.

Вариант 6: Выделение комментариев из кода

Напишите регулярное выражение для выделения комментариев из кода на C++.

Однострочные комментарии начинаются с //

Многострочные комментарии начинаются с /* и заканчиваются */

Программные средства и инструменты

Распространенные языки программирования высокого уровня как правило имеют поддержку регулярных выражений в стандартных библиотеках.

В некоторых языках требуется использование внешних библиотек, а в некоторых поддержка является нативной.

Для выполнения первых двух лабораторных работ рекомендуется использовать язык программирования, с которым вы имеете наибольший опыт работы, чтобы вы могли сосредоточиться на регулярных выражениях, а не на самом языке.

Если такого языка нет — выбирайте perl или python. Perl поддерживает регулярные выражения на уровне языка, и программа будет максимально короткой. Python пригодиться в других работах. Также вы можете посмотреть примеры, приведенные далее, и выбрать то, что более понятно.

Далее кратко рассмотрена поддержка регулярных выражений в различных языках.

В языке C++ поддержка регулярных выражений была введена в стандартной библиотеке с версии C++11 через заголовочный файл <regex>.

Основные классы и функции для работы с регулярными выражениями:

std::regex — класс, представляющий регулярное выражение.

std::smatch и std::cmatch — классы для хранения результатов поиска.

std::regex_search — функция для поиска подстрок, соответствующих регулярному выражению.

std::regex_match — функция для проверки полного соответствия строки регулярному выражению.

std::regex_replace — функция для замены подстрок, соответствующих регулярному выражению.

Пример поиска адреса электронной почты в тексте:

```
#include <iostream>
#include <regex>
#include <string>
int main() {
   std::string text = "My email is example@example.com.";
   std::regex email_pattern(R"(\w+@\w+\.\w+)");
   std::smatch matches;
   if (std::regex_search(text, matches, email_pattern)) {
      std::cout << "Found email: " << matches[0] << std::endl;
   } else {
      std::cout << "No email found." << std::endl;
   }

   return 0;
}
```

В Python регулярные выражения поддерживаются модулем re.

```
import re
```

```
text = "Мой email example@example.com."
pattern = r"\w+@\w+\.\w+"

match = re.search(pattern, text)
if match:
    print("Найден email:", match.group())
else:
    print("Email не найден.")
```

В Perl регулярные выражения встроены в язык и используются с операторами =~ и s///.

```
my $text = "Мой email example\@example.com.";
if ($text =~ /(\w+\@\w+\.\w+)/) {
    print "Найден email: $1\n";
} else {
    print "Email не найден.\n";
}
```

В PHP регулярные выражения поддерживаются с помощью функций preg_match и preg_replace.

```
$text = "Мой email example@example.com.";
$pattern = "/\w+@\w+\.\w+/";

if (preg_match($pattern, $text, $matches)) {
   echo "Найден email: " . $matches[0];
} else {
   echo "Email не найден.";
}
```

В Java регулярные выражения поддерживаются через классы Pattern и Matcher.

```
import java.util.regex.*;

public class Main {
   public static void main(String[] args) {
      String text = "Мой email example@example.com.";
      String pattern = "\\w+@\\w+\\.\\w+";

      Pattern compiledPattern = Pattern.compile(pattern);
      Matcher matcher = compiledPattern.matcher(text);

      if (matcher.find()) {
         System.out.println("Найден email: " + matcher.group());
      } else {
         System.out.println("Email не найден.");
      }
```

```
    }
}
```

Лабораторная работа №2. Регулярные выражения - замена

Цель работы:

научиться составлять сложные регулярные выражения, пользоваться группами захвата в регулярных выражениях, встраивать регулярные выражения в код на языках программирования.

Дополнительная цель:

научиться определять границы применимости регулярных выражений, отличать задачи, для которых данный инструмент не подходит.

Теоретические основы

При обработке текста часто приходится не только искать в тексте нужный фрагмент, но и заменять его на другой. Для поиска фрагмента для замены можно использовать, в том числе, регулярные выражения. Но так как регулярные выражения позволяют найти неидентичные фрагменты текста — то и замена часто должна быть произведена не на фиксированный текст.

Регулярные выражения в своей современной форме содержат правила, которые позволяют в некоторой степени менять и тот текст, который вставляется вместо найденного. К сожалению, возможности по созданию такого текста не очень велики. С помощью регулярных выражений можно указать, какая именно часть исходной строки должна быть заменена, а какая должна остаться неизменной. Кроме того, в исходной строке можно выделить

фрагменты, которые можно фиксировано переупорядочить и вставить. Если требуются более сложные преобразования — то регулярные выражения, как правило, становятся неподходящим инструментом.

Для замены используются два шаблона — шаблон поиска и шаблон замены. Шаблон поиска может включать себя фрагменты, которые называются «группами захвата» - они выделяются круглыми скобками. В шаблоне замены такие группы обозначаются знаком доллара и номером группы. На место такой конструкции вставляется содержимое группы захвата без изменений.

Например, если шаблон поиска «(пере)([а-я])», а шаблон замены «недо$2», то например слово «перегиб» будет заменено на «недогиб». Первая группа захвата не фигурирует в замене. Поэтому она выкидывается, а вторая группа будет вставлена в замененный фрагмент.

Основные шаги для использования регулярных выражений для замены включают: определение шаблона, нахождение соответствующих участков текста и замена этих участков на новый текст.

Например, у нас есть текст "The rain in Spain falls mainly in the plain." и нам нужно заменить слово "rain" на "snow". В этом случае шаблоном будет \ brain\b, который находит слово "rain", ограниченное границами слов. Замена будет "snow". Результатом замены будет строка "The snow in Spain falls mainly in the plain.".

Еще один пример - замена всех цифр в строке "My phone number is 123-456-7890." на знак "#". Шаблоном будет \d, который находит любую цифру. Замена будет знаком "#". Результатом замены будет строка "My phone number is ###-###-####.".

Регулярные выражения также позволяют использовать группы для более сложных замен. Например, если нужно поменять местами две группы текста в строке "John Smith", шаблоном будет (\w+) (\w+), который находит два слова, разделенные пробелом. Замена будет \2, \1, что означает, что нужно поменять местами найденные группы: вторую поставить на первое место, а первую — на второе, добавив между ними запятую и пробел. Результатом замены будет строка "Smith, John".

Задание на лабораторную работу 2:

Написать следующие регулярные выражения:

a. Заменить в арифметическом выражении все числа на буквы

Например: (2+3)*4 → (x+x)*x

Для каждого из усложняющих вариантов написать либо регулярное выражение, либо причину, почему это не получилось:

b. в числах может быть десятичная точка 2.5+4 → a+b

c. одинаковые числа заменяются на одинаковые буквы 2+(2+3) → a+(a+b)

d. множители перед и после скобок не заменяются 2(3+4) → 2(a+b)

e. двузначные и более числа заменяются, а однозначные — нет 2*13+7*56 → 2*a+7*b

f. первый множитель в каждой паре не заменяется, удаляется лишний знак умножения 2*3 + 4*5 → 2a+4b

g. арифметическое выражение переводится из инфиксной в постфиксую запись 2*3+1 → ab*c+

Если регулярное выражение написать не получилось, но это в принципе возможно, то это не считается ошибкой, при условии, что вы привели ход ваших рассуждений до того момента, как встали в тупик, и эти рассуждения не точно повторяют рассуждения того, кто перед вами защищался. Рекомендуется записать рассуждения до защиты своими словами — так они будут звучать оригинальнее. Доказывать, что регулярное выражение действительно невозможно написать, не обязательно, так как это сложно. Но если получится, то это, конечно, приветствуется.

Программные средства и инструменты

Библиотеки, поддерживающие работу с регулярными выражениями, также поддерживают замену. Обычно для замены существует отдельная функция или метод, которая отличается от поиска количеством аргументов.

Пример замены в тексте адреса электронной почты на строку [email was here] в языке C++:

```
#include <iostream>
#include <regex>
```

```
#include <string>

int main() {
    std::string text = "My email is example@example.com.";
    std::regex email_pattern(R"(\w+@\w+\.\w+)");
    std::string replacement = "[email was here]";

    std::string result = std::regex_replace(text, email_pattern, replacement);
    std::cout << "Modified text: " << result << std::endl;

    return 0;
}
```

Python

В Python для замены подстрок используется функция re.sub.

```
import re
text = "Мой email example@example.com."
pattern = r"\w+@\w+\.\w+"
replacement = "[скрытый email]"
result = re.sub(pattern, replacement, text)
print("Изменённый текст:", result)
```

Perl

В Perl для замены подстрок используется оператор s///.

```
my $text = "Мой email example\@example.com.";
```

```
my $pattern = qr/\w+\@\w+\.\w+/;
my $replacement = "[скрытый email]";

$text =~ s/$pattern/$replacement/;
print "Изменённый текст: $text\n";
```

В PHP для замены подстрок используется функция preg_replace.

```
$text = "Мой email example@example.com.";
$pattern = "/\w+@\w+\.\w+/";
$replacement = "[скрытый email]";

$result = preg_replace($pattern, $replacement, $text);
echo "Изменённый текст: $result";
```

В Java для замены подстрок используется метод replaceAll класса String.

```
import java.util.regex.*;

public class Main {
   public static void main(String[] args) {
      String text = "Мой email example@example.com.";
      String pattern = "\\w+@\\w+\\.\\w+";
      String replacement = "[скрытый email]";
```

```
        Pattern compiledPattern = Pattern.compile(pattern);
        Matcher matcher = compiledPattern.matcher(text);

        String result = matcher.replaceAll(replacement);
        System.out.println("Изменённый текст: " + result);
    }
}
```

Лабораторная работа №3. Контекстно-свободные грамматики

Цель работы:

научиться использовать контекстно-свободные грамматики для описания синтаксиса искусственного языка и фрагментов синтаксиса естественного языка. Научиться использовать библиотеки для работы с контекстно-свободной грамматикой.

Дополнительная цель:

понять границы применимости контекстно-свободных грамматик и других формальных грамматик для работы с естественными языками.

Теоретические основы

Формальная грамматика — это система описания синтаксической структуры языка. Она состоит из множества правил, которые определяют, какие последовательности символов могут образовывать допустимые конструкции в данном языке.

В зависимости от вида правил грамматики она может относиться к одному из четырех типов иерархии Хомского. Самый ограниченный тип —

регулярные грамматики. Этот тип эквивалентен по возможности регулярным выражениям, которые были изучены на предыдущих лабораторных. К его ограничениям регулярных выражений, в частности, относится необходимость анализа в один проход. Чтобы снять часть из этих ограничений — необходимо использовать следующий по ограниченности тип — контекстно-свободные грамматики. С помощью контекстно-свободных грамматик можно описать практически любой искусственный язык, в частности языки программирования. Но ограничения не позволяют описать полностью синтаксис естественных языков.

Для описания контекстно-свободных грамматик существует множество распространенных нотаций, в частности формы Бекуса-Наура и синтаксические диаграммы. Но в данной работе предлагается использовать описание формальной грамматики, не привязанное конкретно к контекстно-свободному типу.

Формальная грамматика состоит из следующих элементов.

Набор терминалов (токенов) - это символы, которые присутствуют в языке и не могут быть дальше разложены на более простые элементы.

Набор нетерминалов (переменных) - это символы, которые могут заменяться другими символами в процессе разбора.

Начальный символ - это специальный нетерминал, с которого начинается процесс разбора.

Правила вывода или продукции определяют, какие символы можно заменить другими символами в процессе вывода строки языка.

Контекстно-свободная грамматика — это грамматика, в которой в левой части правила вывода должен находиться только один нетерминал. Это ограничение отличает контекстно-свободную грамматику от менее

ограниченных типов. Ограничений на правую часть нет — это отличает контекстно-свободные грамматики от автоматных, в которых должны соблюдаться ограничения на вид правой части.

Пример контекстно свободной грамматики для расстановки круглых скобок:

S -> (S) S

S -> ε

«S» — это начальный нетерминальный символ.

«(» и «)» — это терминальные символы.

«ε» - обозначает пустую строку

«->» - отделяет левую часть продукции (или правила вывода) от правой.

Такая грамматика описывает сбалансированные круглые скобки, между которыми ничего нет. Например «()», «(()())», «((()))» и подобные строки. Стоит отметить, что этот «язык» - то есть множество всех возможных сбалансированных расстановок скобок — невозможно описать регулярными выражениями.

Задание на лабораторную работу 3:

С помощью контекстно-свободной грамматики описать часть синтаксиса языка C++ или любого другого языка программирования.

Важные условия:

Набор правил должен воспринимать как корректную программу «hello world».

Вы должны сами написать все правила, а не найти готовые в интернете, поэтому:

1. Набор правил не должен быть полным;

2. Смысл каждого правила в наборе должен быть вам ясен — вы должны мочь его объяснить и привести пример его исполнения и нарушения. На защите я могу попросить вас удалить любое из правил, а потом добавить в проверяемый код то, что сделает код «некорректным», но он снова станет корректным, если правило вернуть.

Программные средства и инструменты

Для минимального выполнения работы допустимо использовать этот сайт: https://web.stanford.edu/class/archive/cs/cs103/cs103.1156/tools/cfg/ .

Ограничением данного сайта является невозможность использовать перевод строки в анализируемом коде, а также использование только заглавных латинских букв в качестве нетерминалов, что ограничивает число правил 26ю.

Существуют библиотеки, которые используются для разработки языков программирования, трансляторов и компиляторов. Их использование позволит снять указанные ограничения и получить опыт работы с применяемыми на практике инструментами. Однако работа с искусственными языками выходит за рамки этого курса.

Самыми распространенными средствами являются yacc и GNU/Bison для C++. GNU/Bison фактически является реализацией yacc, поэтому код очень похож для обеих средств.

Пример грамматики арифметических выражений:

%{

```
#include <stdio.h>
#include <stdlib.h>

int yylex(void);
void yyerror(const char *s);
%}

%token NUMBER

%%
expr:   expr '+' term     { $$ = $1 + $3; }
   |   expr '-' term      { $$ = $1 - $3; }
   |   term               { $$ = $1; }
   ;

term:   term '*' factor   { $$ = $1 * $3; }
   |   term '/' factor    { $$ = $1 / $3; }
   |   factor             { $$ = $1; }
   ;

factor: '(' expr ')'      { $$ = $2; }
     | NUMBER             { $$ = $1; }
```

```
    ;

%%

int main(void) {
    return yyparse();
}

void yyerror(const char *s) {
    fprintf(stderr, "Error: %s\n", s);
}

#include <ctype.h>

int yylex(void) {
    int c;
    while ((c = getchar()) == ' ' || c == '\t') {
        /* Игнорируем пробелы */
    }
    if (isdigit(c)) {
        ungetc(c, stdin);
        scanf("%d", &yylval);
        return NUMBER;
    }
    if (c == '\n' || c == EOF) {
        return 0;
```

```
    }
    return c;
}
```

Для языка python существует библиотека PLY (pyhon lex and yacc), которая тоже является аналогом yacc (и lex).

Это пример той же самой грамматики арифметических выражений с использованием PLY.

```
import ply.lex as lex
import ply.yacc as yacc

tokens = ('NUMBER',)

t_ignore = ' \t'
t_NUMBER = r'\d+'

def t_newline(t):
    r'\n'
    t.lexer.lineno += 1
    return t
```

```
def t_error(t):
    print(f"Illegal character '{t.value[0]}'")
    t.lexer.skip(1)

lexer = lex.lex()

def p_expr_add(p):
    'expr : expr "+" term'
    p[0] = p[1] + p[3]

def p_expr_sub(p):
    'expr : expr "-" term'
    p[0] = p[1] - p[3]

def p_expr_term(p):
    'expr : term'
    p[0] = p[1]

def p_term_mul(p):
    'term : term "*" factor'
    p[0] = p[1] * p[3]

def p_term_div(p):
    'term : term "/" factor'
    p[0] = p[1] / p[3]
```

```
def p_term_factor(p):
    'term : factor'
    p[0] = p[1]

def p_factor_num(p):
    'factor : NUMBER'
    p[0] = int(p[1])

def p_factor_expr(p):
    'factor : "(" expr ")"'
    p[0] = p[2]

def p_error(p):
    print("Syntax error")

parser = yacc.yacc()

while True:
    try:
        s = input('calc > ')
    except EOFError:
        break
```

```
if not s:
    continue
result = parser.parse(s)
print(result)
```

Лабораторная работа №4. Токенизация текста

Цель работы:

Научиться методам предварительной обработки текста, в частности токенизации — разбиения текста на слова или другие субъединицы — токены.

Теоретические основы

Токенизация - разбиение текста на предложения, слова, слоги и другие субъединицы.

С токенизации обычно начинается обработка текста на естественном языке.

Токенизация помогает преобразовать текст в формат, который проще анализировать и обрабатывать известными компьютерными алгоритмами. Без токенизации алгоритмы могли бы столкнуться с проблемами в понимании грамматики и смысла текста. Алгоритмам, также как и людям, легче обрабатывать текст в виде токенов, чемввиденепрерывногопотокасимволов.

Путем токенизации можно сократить количество уникальных элементов в тексте, что облегчает обработку данных и уменьшает размерность пространства признаков.

После токенизации проще провести векторизацию — преобразование текста в набор векторов — то есть последовательности чисел, где каждое слово заменяется его номером в словаре. Векторизация нужна для многих, но не для всех алгоритмов.

Во многих языках (например немецком и китайском) разделение на слова не является однозначным. Токенизация, оптимизированная для конкретного языка значительно упрощает задачу дальнейшей обработки.

Для токенизации можно использовать несколько приёмов.

Токенизация по пробелам — это способ пословной токенизации, при котором границей токена считается пробел. Это самый простой способ токенизации. Этот метод может быть достаточным для многих языков и задач. Однако, с его помощью нельзя отделить от слов знаки пунктуации. В этом его большой недостаток, так как одинаковые слова с разными знаками пунктуации после них будут восприняты как разные, что может отрицательно сказаться на дальнейшей обработке.

Задание на лабораторную работу 4:

Провести пословную токенизацию текста на русском языке.

Текст найти или написать самостоятельно.

Для токенизации использовать одну из многих существующих библиотек. Не надо писать свой токенизатор!

Для получения дополнительного балла можно самостоятельно исследовать проблемы токенизации, характерные для русского языка, и написать (или адаптировать) код, которые решает одну такую проблему.

Программные средства и инструменты

Токенизация текста является важным этапом в обработке естественного языка (NLP). Для этой задачи существует несколько популярных библиотек, каждая из которых обладает своими особенностями и преимуществами.

NLTK - одна из старейших и наиболее используемых библиотек для NLP в Python.

```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize

text = "Hello, world!"
tokens = word_tokenize(text)
print(tokens)
```

spaCy — более новая библиотека для python, поддерживает несколько языков.

```
import spacy
nlp = spacy.load('en_core_web_sm')
doc = nlp("Hello, world!")
tokens = [token.text for token in doc]
print(tokens)
```

TextBlob — Простая библиотека для обработки текста на Python, построенная на базе NLTK и Pattern. Подходит для простых задач, таких как эта работа.

```
from textblob import TextBlob

text = "Hello, world!"
blob = TextBlob(text)
tokens = blob.words
print(tokens)
```

Tensorflow — это свободная библиотека с открытым кодом для машинного обучения, в основном ориентированная на нейронные сети. Библиотека изначально была разработана Google для внутреннего использования.

В библиотеке tensorflow есть несколько токенайзеров, в том числе простой WhitespaceTokenizer и более функциональный UnicodeScriptTokenizer.

Пример использования WhitespaceTokenizer:

```
import tensorflow as tf
import tensorflow_text as tf_text
text = tf.constant(["Привет, как дела?"])
tokenizer = tf_text.WhitespaceTokenizer()
tokens = tokenizer.tokenize(text)
tokens = tokens.to_list()
print(tokens)
```

Пример использования UnicodeScriptTokenizer:

```
import tensorflow as tf
import tensorflow_text as tf_text
text = tf.constant(["Привет, как дела?"])
unicode_tokenizer = tf_text.UnicodeScriptTokenizer()
unicode_tokens = unicode_tokenizer.tokenize(text)
unicode_tokens = unicode_tokens.to_list()
print(unicode_tokens)
```

Лабораторная работа №5. Лемматизация, стемминг и векторизация

Цель работы:

Научиться производить выделение основы — стемминг, выделение словарной формы — лемматизацию, и преобразование текста в вектор или набор векторов — векторизацию. Понять, в каких случаях эти виды обработки текста полезны, а в каких — избыточны.

Теоретические основы

Стемминг - это процесс обрезания слов до их основы или корня. Цель состоит в том, чтобы свести различные формы одного слова к единой базовой форме.

Стемминг может быть быстрым и простым, но он не всегда создает правильные или лексически корректные основы, так как результат может быть несуществующим или неправильным словом.

Лемматизация - это процесс приведения слов к их леммам или базовым формам, учитывая их морфологические характеристики и грамматический контекст. Лемма - это слово, которое представляет собой словарную форму слова.

В отличие от стемминга, лемматизация учитывает контекст и обеспечивает корректные словоформы.

Векторизация - это процесс преобразования текстовых данных в числовые векторы, которые могут быть использованы компьютерными алгоритмами для анализа и обработки. Текстовые данные, такие как слова или фразы, представляются в виде числовых признаков.

Вот пример стемминга и лемматизации предложения «Понять, в каких случаях эти виды обработки текста полезны, а в каких — избыточны.»:

Стемминг:

Понят, в как случа эти вид обработк текст полезн, а в как — избыточн.

Лемматизация:

Понять, в каких случай эти вид обработка текст полезный, а в какой — избыточный.

Методы лемматизации, стемминга и векторизации основаны на различных подходах к обработке текстовых данных. Лемматизация преобразует слова в их начальную форму (лемму), что часто делается с помощью словарей и грамматических правил. Словарные методы используют базы данных, такие как WordNet, для сопоставления слов с их базовыми формами. Методы, основанные на правилах грамматики, применяют морфологические правила для преобразования слов. Статистические модели, обученные на больших корпусах текстов, предсказывают леммы на основе контекста слова.

Стемминг направлен на удаление аффиксов для получения основы слова. Алгоритм Porter использует последовательность правил для удаления

суффиксов, проходя через несколько фаз. Улучшенная версия Porter, известная как Snowball или Porter2, применяет дополнительные правила для большей гибкости и точности. Lancaster Stemmer, более агрессивный алгоритм, значительно сокращает слова, что может быть как плюсом, так и минусом. Lovins Stemmer, один из самых ранних стеммеров, использует сложные правила и словари для однофазного удаления суффиксов. Krovetz Stemmer менее агрессивен и стремится сохранять лексическое значение слов после стемминга.

Векторизация текста преобразует текстовые данные в числовые векторы для машинного обучения. Метод мешка слов (Bag of Words) преобразует текст в вектор фиксированной длины на основе частоты слов в документе, где каждое слово представлено отдельной позицией в векторе. TF-IDF (Term Frequency-Inverse Document Frequency) модифицирует мешок слов, учитывая частоту слова в документе и его редкость в коллекции документов, что помогает выделить более информативные слова.

Word2Vec создает плотные векторы (эмбеддинги) фиксированной длины для каждого слова, используя два подхода: CBOW (Continuous Bag of Words), который прогнозирует текущее слово по его контексту, и Skip-gram, который прогнозирует окружающие слова по текущему слову. GloVe (Global Vectors for Word Representation) обучает модели на основе матрицы совместной встречаемости слов в корпусе текстов, создавая вектора слов, которые учитывают как локальную, так и глобальную статистику слов. FastText расширяет Word2Vec, учитывая подсловные структуры (символьные n-граммы), что позволяет моделям лучше обрабатывать редкие слова и новые формы слов.

ELMo (Embeddings from Language Models) использует двунаправленные LSTM (Long Short-Term Memory) для создания контекстуальных эмбеддингов, которые зависят от всего предложения и учитывают различия в значении слова в разных контекстах. BERT (Bidirectional Encoder Representations from Transformers) основан на архитектуре трансформеров и создает контекстуальные эмбеддинги, учитывающие как левый, так и правый контекст слова. Модель обучается на задачах маскированного языкового моделирования и предсказания следующего предложения.

Doc2Vec является расширением Word2Vec для представления целых документов в виде векторов и имеет два основных подхода: Distributed Memory (DM), который учитывает порядок слов в документе, и Distributed Bag of Words (DBOW), который игнорирует порядок слов. Sentence Transformers используют архитектуру трансформеров для создания эмбеддингов предложений, что позволяет моделям улавливать семантические связи между словами в предложении и учитывать контекст целого предложения.

Задание на лабораторную работу 5:

В тексте посчитать количество различных слов, для чего потребуется выполнить токенизацию по словам любым способом. Далее использовать готовый стеммер и лемматизатор из любой библиотеки.

Произвести удаление стоп-слов из текста.

Посчитать количество различных слов в тесте до и после каждого процесса, а также всех их комбинаций.

Субъективно оценить понятность текста после стемминга, лемматизации и удаления стоп-слов.

Написать код для векторизации текста, используя различный размер словаря, который выводит результат векторизации, а также производит обратный процесс — заменяет все численные значения, которые есть в словаре, на слова.

Оценить понятность текста, прошедшего векторизацию и обратное преобразование, если перед векторизацией выполнялись или не выполнялись стемминг, лемматизация и удаление стоп-слов. Определить оптимальную комбинацию для вашего текста.

Программные средства и инструменты

На языке python для лемматизации можно использовать библиотеки nltk, spacy или pymorphy2.

Пример с использованием библиотеки nltk:

```
import nltk
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
nltk.download('omw-1.4')
lemmatizer = WordNetLemmatizer()
words = ['running', 'ran', 'runs', 'easily', 'fairly']
lemmatized_words = [lemmatizer.lemmatize(word) for word in words]
print(lemmatized_words)
```

Пример с использованием библиотеки spacy:

```
import spacy
nlp = spacy.load('en_core_web_sm')
doc = nlp("running ran runs easily fairly")
lemmatized_words = [token.lemma_ for token in doc]
print(lemmatized_words)
```

Пример с использованием библиотеки pymorphy2:

```
import pymorphy2
morph = pymorphy2.MorphAnalyzer()
words = ['бегущий', 'бежал', 'бегает', 'легко', 'справедливо']
lemmatized_words = [morph.parse(word)[0].normal_form for word in words]
print(lemmatized_words)
```

Лабораторная работа №6. Подбор датасета для машинного обучения

Цель работы:

Ознакомиться с общедоступными наборами данных для машинного обучения для обработки текста на естественном языке, их структурой и содержанием. Подобрать набор данных для оставшихся работ.

Теоретические основы

Датасет для машинного обучения представляет собой набор данных, который используется для обучения моделей машинного обучения или для их оценки. Он обычно состоит из входных признаков и соответствующих целевых значений, если речь идет о задаче обучения с учителем.

Датасеты с текстом на естественном языке могут храниться в различных форматах. Наиболее типичными вариантами являются JSON и CSV.

Датасеты для машинного обучения могут быть построены и храниться в различных форматах в зависимости от их назначения и типа данных. Построение датасетов начинается со сбора данных, который может быть ручным (через анкеты, эксперименты или ручной ввод) или автоматическим (с помощью веб-скрейпинга, API или датчиков). Далее следует предобработка данных, включающая очистку (удаление пропущенных значений, устранение дубликатов, исправление ошибок) и преобразование (нормализация, масштабирование, кодирование категориальных переменных). Затем данные

разделяются на обучающую выборку (для обучения модели), валидационную выборку (для настройки гиперпараметров модели) и тестовую выборку (для оценки производительности модели).

Датасеты могут храниться в различных форматах. Текстовые форматы включают CSV (Comma-Separated Values) для табличных данных, TSV (Tab-Separated Values) с табуляцией в качестве разделителя, JSON (JavaScript Object Notation) для структурированных данных и XML (eXtensible Markup Language) для хранения и передачи данных. Бинарные форматы включают HDF5 (Hierarchical Data Format) для хранения больших объемов данных и Parquet, колонко-ориентированный формат, оптимизированный для больших данных в системах Hadoop. Специализированные форматы включают файлы изображений (JPEG, PNG, TIFF), аудиофайлы (WAV, MP3, FLAC) и видеофайлы (MP4, AVI, MKV).

Использование датасетов включает обучение моделей, где обучающая выборка применяется для построения модели, а валидационная и тестовая выборки - для оценки её производительности. Перед обучением часто выполняется исследовательский анализ данных (EDA) для понимания основных характеристик данных, выявления закономерностей и аномалий. Датасеты также используются для проверки гипотез и проведения статистических тестов.

Примеры использования форматов данных включают CSV, который часто используется в начальных стадиях проектов машинного обучения для

хранения небольших табличных данных, таких как набор данных о стоимости недвижимости. JSON применяется для работы с данными, полученными через веб-сервисы, например, данными о погоде. HDF5 используется в проектах, требующих хранения и обработки больших объемов данных, например, для анализа изображений в высоком разрешении. Parquet широко используется в экосистемах Big Data для эффективного хранения и обработки больших объемов данных, например, для хранения логов веб-сайтов. Правильное построение и хранение датасетов играет ключевую роль в успешности проектов машинного обучения, так как качество данных напрямую влияет на качество моделей.

При выборе датасета для практических задач стоит обратить внимание на лицензию. Для данной работы можно использовать и датасеты для некоммерческого использования, но надо обязательно посмотреть, где в выбранном ресурсе указывается лицензия.

При использовании данных важно учитывать лицензирование, чтобы убедиться в законности их использования:

Открытые лицензии: такие как Creative Commons (CC), Apache License, MIT License и другие. Эти лицензии позволяют использовать, изменять и распространять данные при соблюдении определенных условий. Например, Creative Commons предлагает несколько типов лицензий, от CC0 (публичное достояние) до CC BY (требует указания авторства) и CC BY-SA (разрешает изменения при условии, что новые работы будут распространяться на тех же условиях).

Коммерческие лицензии: данные могут быть доступны по платной подписке или за разовую плату. Условия использования таких данных могут варьироваться в зависимости от поставщика.

К особому случаю стоит отнести лицензии, допускающие некоммерческое использование, но не разрешающие коммерческое, например Creative Commons CC BY-NC.

Собственные данные: данные, собранные и аннотированные собственноручно, могут использоваться без ограничений, но при их публикации или распространении стоит рассмотреть возможность лицензирования для защиты авторских прав.

Задание на лабораторную работу 6:

Найти в интернете массив данных, подходящий для машинного обучения.

Требования к данным:

1. Данные должны содержать фразу из нескольких слов на любом языке, кроме английского. Например — название статьи, работы, цитата, анекдот и т. п. На английском нельзя, потому что слишком просто, и много готовых примеров.

2. Данные должны содержать признак, который можно использовать как предмет распознавания. Это может быть отрасль, тема, возраст автора и вообще всё, что угодно. Если в процессе исследования связи не будет найдено — это не страшно — доказать отсутствие связи или то, что это плохой пример — тоже результат.

Например, в этом видео https://youtu.be/Y_hzMnRXjhI автор использует базу данных заголовков статей, и признак — саркастический заголовок или серьёзный.

Но эта база на английском, так что надо искать другую.

3. Данные можно легко получить в машинно-читаемом виде. Либо они сразу уже в xml или json и т.п., либо их легко распарсить, и вы готовы это сделать для следующей лабораторной.

4. Массив должен содержать не менее 200 записей, но лучше — в разы больше.

5. У каждого студента должен быть свой массив данных. Краткое описание, признак и ссылку на ресурс застолбить в комментариях.

Онлайн-ресурсы для поиска датасета

Есть несколько популярных источников, где можно найти датасеты с текстами для задач машинного обучения:

1. **Kaggle**: Платформа с множеством датасетов для различных задач, включая обработку естественного языка (NLP).
    - https://www.kaggle.com/datasets
2. **UCI Machine Learning Repository**: Один из старейших и наиболее известных ресурсов для различных машинных задач, включая текстовые данные.
    - https://archive.ics.uci.edu/
3. **Google Dataset Search**: Поисковая система от Google, позволяющая находить датасеты, размещенные в интернете.
    - https://datasetsearch.research.google.com/

4. **Hugging Face Datasets**: Платформа, специализирующаяся на датасетах и моделях для NLP.
    - https://huggingface.co/docs/datasets/en/index
5. **The Stanford Natural Language Processing Group**: Коллекция различных NLP датасетов от Стэнфордского университета.
    - https://nlp.stanford.edu/projects/snli/
6. **Wikipedia Dumps**: Тексты из Википедии, которые можно использовать для различных NLP задач.
    - https://dumps.wikimedia.org/
7. **Reddit Datasets**: Датасеты с данными из Reddit, полезные для анализа социальных сетей и текстов.
    - https://paperswithcode.com/dataset/pushshift-reddit

Эти ресурсы предоставляют разнообразные текстовые датасеты, которые можно использовать для обучения моделей машинного обучения и NLP.

Лабораторная работа №7. Обучение классификатора

Цель работы:

Научиться обрабатывать датасет и использовать данные для обучения классификаторов и их применения.

Теоретические основы

Классификация текста — это процесс категоризации текстовых данных на основе их содержания. Существует множество методов и подходов к классификации текста, и их можно разделить на несколько основных типов:

Бинарная классификация:

Бинарная классификация предполагает разделение текста на две категории, например, спам и не спам.

Многоклассовая классификация:

Многоклассовая классификация предполагает распределение текста по одной из нескольких категорий. Например, классификация новостей по темам (спорт, политика, экономика и т.д.).

Многоуровневая (иерархическая) классификация:

Многоуровневая классификация включает распределение текста по иерархии категорий, где каждая категория может содержать подкатегории.

Многомаркерная классификация (multi-label classification):

Многомаркерная классификация предполагает, что один и тот же текст может принадлежать сразу нескольким категориям. Например, документ может одновременно относиться к категориям "наука" и "технологии".

Методы классификации текста

Базовые методы:

Bag of Words (BoW): Преобразование текста в вектор признаков на основе частоты слов.

TF-IDF (Term Frequency-Inverse Document Frequency): Взвешивание частоты слов с учетом их значимости в документе и в корпусе текстов.

Модели машинного обучения:

Наивный Байесовский классификатор: Статистический метод, основанный на теореме Байеса.

Логистическая регрессия: Метод классификации, использующий логистическую функцию.

Поддерживающие векторы (SVM): Метод, который пытается найти оптимальную гиперплоскость для разделения классов.

Модели глубокого обучения:

Рекуррентные нейронные сети (RNN): Используются для обработки последовательных данных, таких как текст.

Долгосрочная кратковременная память (LSTM): Улучшенная версия RNN, которая справляется с проблемой долгосрочных зависимостей.

Трансформеры: Современные архитектуры, такие как BERT и GPT, которые хорошо справляются с задачами обработки естественного языка.

Модели на основе глубокого обучения с предобучением:

BERT (Bidirectional Encoder Representations from Transformers): Модель, которая обучается на большом количестве данных и затем дообучается на конкретной задаче.

GPT (Generative Pre-trained Transformer): Модель, которая используется для генерации текста и других задач обработки естественного языка.

В данной работе надо использовать методы машинного обучения. При этом не обязательно использовать глубокое обучение — допустимо использовать любой метод машинного обучения.

Задание на лабораторную работу 7:

Используя датасет из предыдущей работы произвести обучение классификатора текста. Для классификации использовать один из признаков в датасете.

Обучить классификатор на подготовленных данных из пятой лабораторной работы, исключив часть данных для контроля. Для этого фразу токенизировать по словам и нормализовать.

Проверить работу классификатора на тестовых данных из того же массива, и на данных из другого места, если их возможно получить.

Сделать выводы об эффективности сети для данных из массива и для других, если они есть.

Низкая или нулевая эффективность не является основанием для снижения оценки за лабораторную работу, если всё остальное сделано.

Можно использовать tensorflow или любое другое средство.

Программные средства и инструменты

Классификатор может быть построен как с помощью глубокого обучения, так и с помощью классического машинного обучения.

Это пример на основе глубокого обучения с использованием библиотеки PyTorch:

```
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import LabelEncoder
# Данные
texts = ["I love programming", "Python is great", "I hate bugs", "Debugging is fun"]
labels = ["positive", "positive", "negative", "positive"]
# Подготовка данных
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts).toarray()
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(labels)
# Модель
```

```
model = nn.Sequential(
    nn.Linear(X.shape[1], 2)
)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
# Обучение
X_train = torch.tensor(X, dtype=torch.float32)
y_train = torch.tensor(y, dtype=torch.long)
for epoch in range(100):
    optimizer.zero_grad()
    outputs = model(X_train)
    loss = criterion(outputs, y_train)
    loss.backward()
    optimizer.step()
# Оценка
_, predicted = torch.max(outputs, 1)
accuracy = (predicted == y_train).float().mean().item()
print(f'Accuracy: {accuracy:.4f}')
```

Пояснение для примера:

Подготовка данных: Тексты преобразуются в мешок слов (bag-of-words) с использованием CountVectorizer. Метки преобразуются в числовой формат с помощью LabelEncoder.

Модель: Простая полносвязная сеть с одним линейным слоем.

Обучение: Градиентный спуск с оптимизатором Adam и функцией потерь кросс-энтропия.

Оценка: Предсказания на обучающем наборе данных и вычисление точности.

Это пример на основе классического машинного обучения с использованием библиотеки scikit-learn:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
# Данные
texts = ["I love programming", "Python is great", "I hate bugs", "Debugging is fun"]
labels = ["positive", "positive", "negative", "positive"]

# Подготовка данных и обучение
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts)
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(labels)

model = MultinomialNB()
model.fit(X, y)
```

 # Оценка

y_pred = model.predict(X)

accuracy = accuracy_score(y, y_pred)

print(f'Accuracy: {accuracy:.4f}')

Пояснение для примера:

Подготовка данных: Тексты преобразуются в мешок слов с использованием CountVectorizer. Метки преобразуются в числовой формат с помощью LabelEncoder.

Обучение: Наивный байесовский классификатор MultinomialNB.

Оценка: Предсказания на обучающем наборе данных и вычисление точности.

Существует множество других библиотек и подходов, вы можете использовать любой из них, если он позволяет классифицировать естественный текст.

Исходные данные в примерах заданы в коде. Если вы берете за основу эти примеры (что, в данном случае, не обязательно) — надо организовать загрузку данных из выбранного датасета.

Лабораторная работа №8. Генерация текста

Цель работы:

Понять принцип генерации текста.

Теоретические основы

Для генерации текста необходимо задать начальную последовательность. Это может быть произвольная фраза или случайно выбранный фрагмент из обучающего текста.

Генерация текста происходит итеративно. На каждом шаге модель предсказывает следующий символ или слово, добавляет его к текущей последовательности и использует обновленную последовательность для следующего предсказания.

Существует несколько методов для выбора следующего символа из предсказанного распределения вероятностей:

Argmax: Выбор символа с наибольшей вероятностью. Это может привести к однообразному тексту.

Sampling: Случайный выбор символа с учетом предсказанных вероятностей. Это делает текст более разнообразным и естественным.

Температура (Temperature): Параметр, контролирующий креативность модели. Низкие значения температуры делают распределение вероятностей более "острым" (модель становится более уверенной в своих предсказаниях), а высокие значения делают распределение более "плоским" (модель становится менее уверенной, и вероятность выбора различных символов становится более равномерной).

Задание на лабораторную работу 8:

Выберите текстовый датасет для обучения модели. Это может быть тот же датасет, что и в предыдущей работе. Но, возможно, лучше выбрать другой.

Преобразуйте текст в последовательности символов или слов, которые будут использоваться для обучения модели.

Обучите модель на основе рекуррентной нейронной сети.

Реализуйте функцию для генерации текста на основе обученной модели.

Сгенерируйте несколько текстовых фрагментов, начиная с различных начальных фраз.

Проведите анализ полученных результатов, оцените осмысленность и креативность сгенерированных текстов.

Дополнительное задание (не обязательно для получения зачета):

Попробуйте улучшить качество генерации текста, изменив архитектуру модели или параметры обучения.

Исследуйте возможность использования других архитектур нейронных сетей (например, трансформеров) для генерации текста.

Программные средства и инструменты

TensorFlow/Keras — инструмент для создания и обучения нейронных сетей с гибким высокоуровневым API. Он поддерживает как простые, так и сложные модели, может использовать CPU, GPU и TPU, и включает инструменты для визуализации (TensorBoard), мобильных приложений (TensorFlow Lite) и развертывания моделей (TensorFlow Serving).

Генерация текста с использованием TensorFlow/Keras и LSTM

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

text = "Пример текста."
chars = sorted(set(text))
char_to_idx = {c: i for i, c in enumerate(chars)}
idx_to_char = {i: c for i, c in enumerate(chars)}
seq_length = 5
x = np.zeros((len(text) - seq_length, seq_length, len(chars)), dtype=np.bool)
y = np.zeros((len(text) - seq_length, len(chars)), dtype=np.bool)
for i in range(len(text) - seq_length):
    for t, char in enumerate(text[i:i + seq_length]):
        x[i, t, char_to_idx[char]] = 1
```

```
    y[i, char_to_idx[text[i + seq_length]]] = 1
model = Sequential([
    LSTM(128, input_shape=(seq_length, len(chars))),
    Dense(len(chars), activation='softmax')
])
model.compile(loss='categorical_crossentropy', optimizer='adam')
model.fit(x, y, epochs=10, batch_size=2)

start_text = "Пример"
generated_text = start_text
for _ in range(20):
    x_pred = np.zeros((1, seq_length, len(chars)), dtype=np.bool)
    for t, char in enumerate(generated_text[-seq_length:]):
        x_pred[0, t, char_to_idx[char]] = 1
    preds = model.predict(x_pred, verbose=0)[0]
    next_index = np.argmax(preds)
    next_char = idx_to_char[next_index]
    generated_text += next_char
print(generated_text)
```

PyTorch использует динамическое построение вычислительных графов, что делает его код интуитивно понятным и удобным для отладки. Гибкость позволяет создавать сложные архитектуры, что делает его популярным в научных исследованиях. Он хорошо интегрируется с другими Python-

библиотеками и имеет встроенную поддержку автоматического дифференцирования.

Генерация текста с использованием PyTorch и LSTM

```
import torch
import torch.nn as nn
import numpy as np
text = "Пример текста."
chars = sorted(set(text))
char_to_idx = {c: i for i, c in enumerate(chars)}
idx_to_char = {i: c for i, c in enumerate(chars)}
seq_length = 5
x = np.zeros((len(text) - seq_length, seq_length, len(chars)), dtype=np.float32)
y = np.zeros((len(text) - seq_length, len(chars)), dtype=np.float32)
for i in range(len(text) - seq_length):
    for t, char in enumerate(text[i:i + seq_length]):
        x[i, t, char_to_idx[char]] = 1
    y[i, char_to_idx[text[i + seq_length]]] = 1

x = torch.tensor(x)
y = torch.tensor(y)
class TextGeneratorModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
```

```
        super(TextGeneratorModel, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)
    def forward(self, x):
        h, _ = self.lstm(x)
        out = self.fc(h[:, -1, :])
        return out
model = TextGeneratorModel(len(chars), 128, len(chars))
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
for epoch in range(10):
    outputs = model(x)
    loss = criterion(outputs, y.argmax(dim=1))
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
start_text = "Пример"
generated_text = start_text

for _ in range(20):
    x_pred = np.zeros((1, seq_length, len(chars)), dtype=np.float32)
    for t, char in enumerate(generated_text[-seq_length:]):
        x_pred[0, t, char_to_idx[char]] = 1
    x_pred = torch.tensor(x_pred)
    preds = model(x_pred)
    next_index = torch.argmax(preds).item()
```

```
    next_char = idx_to_char[next_index]
    generated_text += next_char
print(generated_text)
```

Transformers от Hugging Face предоставляет доступ к современным моделям NLP, таким как GPT, BERT и T5. Высокоуровневый API упрощает загрузку и использование предобученных моделей для задач генерации текста, перевода и классификации. Поддерживает как TensorFlow, так и PyTorch, и включает инструменты для работы с данными (datasets) и токенизацией (tokenizers).

Генерация текста с использованием GPT-2 и библиотеки transformers (Hugging Face)

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer
model = GPT2LMHeadModel.from_pretrained('gpt2')
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
start_text = "Пример"
input_ids = tokenizer.encode(start_text, return_tensors='pt')
output = model.generate(input_ids, max_length=len(input_ids[0]) + 20, num_return_sequences=1)
generated_text = tokenizer.decode(output[0], skip_special_tokens=True)
```

```
print(generated_text)
```

Задание для самостоятельной работы 1: Принцип работы трасформеров

Задание на работу:

самостоятельно реализовать и обучить нейронную сеть, построенную по принципу трансформера. Понять отличия этой архитектуры от других архитектур нейронных сетей, которые вы изучали ранее.

Теоретические сведения

Трансформеры — это архитектура нейронных сетей, представленная в статье "Attention is All You Need" Ашиша Васвани и других авторов в 2017 году. Основное нововведение трансформеров — механизм внимания (attention), который позволяет модели учитывать все части входной последовательности одновременно, а не поочередно, как в рекуррентных нейронных сетях (RNN).

Принцип работы трансформеров:

Введение входных данных: Входная последовательность (например, текст) преобразуется в эмбеддинги — вектора фиксированной длины, представляющие смысл слов. Также добавляются позиционные эмбеддинги, чтобы модель знала положение каждого слова в последовательности.

Механизм внимания (Self-Attention): На этом этапе каждый элемент входной последовательности преобразуется с учетом всех других элементов. Механизм внимания вычисляет три матрицы: Query (запросы), Key (ключи) и Value (значения). Для каждого слова в последовательности создается вектор запроса, который сопоставляется с векторами ключей всех других слов, чтобы определить их важность. Результирующие веса используются для взвешивания векторов значений, что позволяет модели учитывать контекст всей последовательности.

Многоголовочный механизм внимания (Multi-Head Attention): Этот механизм включает несколько механизмов внимания, работающих параллельно. Результаты их работы объединяются и линейно преобразуются. Это позволяет модели учитывать разные аспекты взаимосвязей между словами.

Feed-Forward слой: После внимания результаты проходят через два линейных слоя с функцией активации (обычно ReLU) между ними. Эти слои применяются к каждому элементу последовательности независимо.

Добавление и нормализация (Add & Norm): Результаты каждого внимания и feed-forward слоев добавляются к исходным входным данным (residual connection), а затем нормализуются (layer normalization). Это помогает улучшить стабильность и эффективность обучения.

Стек слоев: Описанные выше шаги (многоголовочный механизм внимания и feed-forward слой) повторяются несколько раз, образуя стек энкодеров.

Декодер (Decoder): В трансформерах для задач, таких как машинный перевод, используется декодер. Он похож на энкодер, но дополнительно имеет механизм внимания к выходу энкодера (encoder-decoder attention), что

позволяет учитывать закодированную входную последовательность при генерации выходной.

Ход работы

Возьмите за основу приведенный пример кода. Он реализует простейший трасформер без использования библиотек глубокого обучения, чтобы все принципы работы были понятны.

```
import numpy as np

# Параметры
vocab_size = 10
seq_length = 6
d_model = 8

# Входная последовательность (2 последовательности длиной 6)
input_seq = np.random.randint(0, vocab_size, (2, seq_length))

# Эмбеддинги (здесь просто случайные для примера)
np.random.seed(0)
embedding_matrix = np.random.rand(vocab_size, d_model)
```

```
    embedded_seq = np.array([[embedding_matrix[token] for token in seq]
for seq in input_seq])

    # Позиционные эмбеддинги
    positional_encodings = np.array([[pos / np.power(10000, 2 * (i // 2) /
d_model) for i in range(d_model)] for pos in range(seq_length)])
    positional_encodings[:, 0::2] = np.sin(positional_encodings[:, 0::2])
    positional_encodings[:, 1::2] = np.cos(positional_encodings[:, 1::2])

    embedded_seq = embedded_seq + positional_encodings

    # Self-Attention
    def self_attention(query, key, value):
      scores = np.matmul(query, key.transpose(0, 2, 1)) / np.sqrt(d_model)
            weights = np.exp(scores) / np.sum(np.exp(scores), axis=-1,
keepdims=True)
      output = np.matmul(weights, value)
      return output

    query = np.matmul(embedded_seq, np.random.rand(d_model, d_model))
    key = np.matmul(embedded_seq, np.random.rand(d_model, d_model))
    value = np.matmul(embedded_seq, np.random.rand(d_model, d_model))

    attention_output = self_attention(query, key, value)

    # Feed-Forward Layer
```

```
    def feed_forward(x):
        x = np.maximum(0, np.matmul(x, np.random.rand(d_model, 16))) # ReLU
        x = np.matmul(x, np.random.rand(16, d_model))
        return x

    ffn_output = feed_forward(attention_output)

    # Add & Norm
    output = (embedded_seq + ffn_output - np.mean(embedded_seq + ffn_output, axis=-1, keepdims=True)) / np.std(embedded_seq + ffn_output, axis=-1, keepdims=True)

    print("Входная последовательность:")
    print(input_seq)
    print("\nЭмбеддинги с позиционными кодировками:")
    print(embedded_seq)
    print("\nAttention выход:")
    print(attention_output)
    print("\nИтоговый выход:")
    print(output)
```

Выполните этот код и разберитесь, как он работает. Замените случайные входные данные на какие-либо неслучайные данные, чтобы убедиться, что пример работает.

Так как пример минималистичный — его можно улучшить самыми разными способами, чтобы достичь большего быстродействия, качества обучения или адаптировать к определенной задаче. Найдите способ улучшить данный пример, и добавьте одно усовершенствование. Для защиты подготовьте объяснение того, что вы сделали. Обязательно укажите, какие строки вы добавили или заменили и почему.

Задание для самостоятельной работы 2: Распознавание речи

Теоретические сведения

Распознавание речи, или автоматическое распознавание речи (ASR), представляет собой технологию, позволяющую компьютерам и другим устройствам интерпретировать и обрабатывать устную речь человека. Эта технология базируется на преобразовании звуковых волн в текстовую форму, что позволяет компьютерам понимать и реагировать на голосовые команды. Основной принцип работы системы распознавания речи заключается в анализе акустических сигналов и их преобразовании в последовательность слов с использованием различных алгоритмов и моделей.

Использование распознавания речи охватывает широкий спектр приложений и задач. Одной из самых распространенных областей применения является управление устройствами и программами с помощью голосовых команд. Это включает в себя голосовых ассистентов, таких как Siri от Apple, Google Assistant и Amazon Alexa, которые позволяют пользователям выполнять различные действия, начиная от поиска информации в интернете и заканчивая управлением умным домом. Такие ассистенты делают взаимодействие с

техникой более естественным и удобным, снижая необходимость ввода данных вручную.

В бизнес-среде распознавание речи активно применяется для повышения эффективности работы. Например, системы автоматического ответа и обработки звонков в колл-центрах позволяют значительно сократить время обслуживания клиентов, автоматически распознавая их запросы и направляя к соответствующим специалистам.

В медицинской области технологии распознавания речи помогают врачам и медицинскому персоналу создавать и управлять электронными медицинскими записями. Врачи могут диктовать заметки и диагнозы, которые автоматически преобразуются в текст, что экономит время и снижает вероятность ошибок, связанных с ручным вводом данных.

В образовательной сфере распознавание речи также находит широкое применение. Оно используется для создания субтитров и транскрипций лекций и уроков, что делает образовательные материалы более доступными для людей с ограниченными возможностями слуха. Кроме того, системы распознавания речи могут быть интегрированы в приложения для изучения языков, помогая учащимся улучшать произношение и понимать устную речь на изучаемом языке.

В повседневной жизни распознавание речи используется в мобильных приложениях и устройствах, таких как смартфоны, планшеты и автомобили. Встроенные системы распознавания речи позволяют водителям управлять навигацией, отправлять сообщения и совершать звонки без необходимости отвлекаться от дороги. В сфере развлечений такие технологии применяются для управления мультимедийными системами и поиска контента по голосовым командам.

Распознавание речи включает различные подходы, которые можно разделить на традиционные методы и методы, основанные на современных технологиях машинного обучения и глубокого обучения. Вот основные из них:

Традиционные методы

Метод динамического программирования (Dynamic Time Warping, DTW): Позволяет сравнивать временные последовательности с различной длительностью. Этот метод использовался в ранних системах распознавания речи.

Метод скрытых марковских моделей (Hidden Markov Models, HMM) основан на вероятностных моделях, описывающих последовательности наблюдений. Широко использовался для распознавания речи в 1990-х и 2000-х годах.

Метод гауссовских смесевых моделей (Gaussian Mixture Models, GMM) используется совместно с HMM для моделирования непрерывных признаков речи. Позволяет улучшить точность распознавания.

Современные методы

Глубокие нейронные сети (Deep Neural Networks, DNN):

Используются для автоматического извлечения признаков и улучшения качества распознавания.Подходы включают многослойные перцептроны (MLP), свёрточные нейронные сети (CNN) и рекуррентные нейронные сети (RNN).

Рекуррентные нейронные сети (Recurrent Neural Networks, RNN):

Специально предназначены для работы с последовательностями данных. Подтипы включают долговременную краткосрочную память (LSTM) и сети на основе гейтовых рекуррентных единиц (GRU).

Сверточные нейронные сети (Convolutional Neural Networks, CNN):

Применяются для обработки временных и спектральных признаков речи. Часто используются в комбинации с другими типами нейронных сетей.

Трансформеры (Transformers):

Современные архитектуры, такие как BERT и GPT, которые могут быть адаптированы для распознавания речи. Используются для моделирования долгосрочных зависимостей в данных.

End-to-End модели:

Архитектуры, которые объединяют все этапы распознавания речи в одну нейронную сеть. Примеры включают модели Connectionist Temporal Classification (CTC) и Sequence-to-Sequence (Seq2Seq) с механизмом внимания (attention).

Гибридные модели HMM-DNN комбинируют преимущества HMM и DNN для улучшения качества распознавания. Используются в современных системах ASR.

Использование предобученных моделей, таких как Wav2Vec, для улучшения качества распознавания на специфических задачах. Аугментация данных, включающая добавление шума, изменение скорости речи и другие техники для улучшения устойчивости моделей.

Каждый из этих подходов имеет свои сильные и слабые стороны, и выбор конкретного метода зависит от задач, данных и требований к системе распознавания речи.

Задание на работу

Соберите и проверьте работу следующего примера:

```
import speech_recognition as sr
recognizer = sr.Recognizer()
with sr.AudioFile('voice.flac') as source:
    audio = recognizer.record(source)
text = recognizer.recognize_sphinx(audio)
print(text)
```

Для данного примера необходимо установить библиотеки SpechRecognition и pocketsphinx, а также на английском языке сказать тестовую фразу и сохранить её в формате wav. Обязательно запишите свой голос, не берите файл из интернета.

Скорее всего, качество распознавания будет не высоким. Библиотека pocketsphinx плохо работает (на момент написания) с русским акцентом, а также не поддерживает русский язык, а только английский, китайский французский и итальянский.

Вы должны найти библиотеку, которая работает лучше с английским языком, несмотря на акцент, и написать с ней работающий пример, или найти библиотеку, поддерживающую русский язык, и написать пример с ней. Обязательно использовать в тестовых примерах свой голос.

Программные средства и инструменты

В Python для распознавания речи можно использовать одну из нескольких библиотек, каждая из которых предлагает уникальные возможности и преимущества, а также разные варианты лицензирования, на что следует обратить внимание при разработке коммерческих проектов. Одной из наиболее популярных библиотек является SpeechRecognition, которая предоставляет простой и интуитивно понятный интерфейс для взаимодействия с несколькими движками распознавания речи, такими как Google Web Speech API, Microsoft Bing Voice Recognition и IBM Speech to Text. Эта библиотека особенно удобна для быстрых прототипов и простых приложений, поскольку она абстрагирует сложность работы с API и позволяет разработчикам легко интегрировать функциональность распознавания речи в свои проекты.

Pocketsphinx, являясь частью проекта CMU Sphinx, представляет собой офлайн-движок распознавания речи. Она поддерживает работу на различных платформах и устройствах с ограниченными ресурсами, что делает её подходящей для встроенных систем и приложений, требующих локальной обработки данных. Pocketsphinx обеспечивает достаточную точность для множества задач и не требует подключения к интернету, что является важным преимуществом для приложений, работающих в автономном режиме.

DeepSpeech, разработанная Mozilla, основана на нейронных сетях и предназначена для обеспечения высокой точности распознавания речи. Она использует методы глубокого обучения и доступна для использования в режиме офлайн. DeepSpeech активно развивается и поддерживается сообществом, что позволяет разработчикам использовать современные

технологии машинного обучения для создания решений в области распознавания речи.

Wav2Vec 2.0 от Facebook AI представляет собой ещё одну современную библиотеку, основанную на трансформерах и обучении с самоконтролем. Она демонстрирует высокую точность распознавания даже на малых объемах данных и может быть дообучена на специализированных наборах данных для улучшения производительности в конкретных областях применения. Wav2Vec 2.0 открывает новые возможности для использования методов трансформеров в задачах обработки речи и является полезным инструментом для разработчиков и исследователей.

Для комплексных решений и интеграции с облачными сервисами разработчики могут воспользоваться API от крупных технологических компаний. Google Cloud Speech-to-Text API, Microsoft Azure Cognitive Services Speech SDK и IBM Watson Speech to Text предлагают высокую точность распознавания речи и широкий набор дополнительных функций, таких как поддержка множества языков и возможность кастомизации моделей. Эти сервисы предоставляют масштабируемость и надежность, что делает их подходящими для использования в коммерческих и производственных приложениях.

Таким образом, разнообразие библиотек для распознавания речи в Python позволяет разработчикам выбрать оптимальное решение в зависимости от конкретных требований и задач. Независимо от того, требуется ли локальная

обработка данных или интеграция с облачными сервисами, существует множество инструментов, которые могут удовлетворить различные потребности в области распознавания речи.

Рекомендации по выполнению и оценке работ

Этот раздел поможет студентам понять, как именно подойти к выполнению работы, чтобы получить максимальный объём и качество знаний, а вместе с тем и хорошие оценки.

С этим разделом следует ознакомиться и преподавателю — чтобы процесс оценки и защиты работ способствовал, а не препятствовал получению студентами знаний и навыков в области обработки естественных языков.

Выполнение лабораторной работы

Ознакомьтесь с разделом «теоретические сведения» работы. Это позволит понять терминологию, что необходимо для правильного понимания задания на лабораторную работу. Следует заметить, что терминология обработки естественных языков зачастую заимствована из других информационных технологий и разделов лингвистики, но при этом может иметь иной смысл. Неправильное понимание задания может усложнить сдачу лабораторной работы, привести к излишним трудозатратам.

Прочитав задание, постарайтесь его выполнить. Если вы не знаете средств или технологий для выполнения — ознакомьтесь с разделом «Программные средства и инструменты». Не обязательно разбираться со всеми приведенными технологиями — выберете ту, которая позволит максимально эффективно использовать уже имеющиеся у вас знания.

Результат выполнения работы может не быть идеальным. Важно понимание полученного результата, и способность его объяснить, а главное — модифицировать.

Оценивание выполнения работ

Во время защиты работы преподаватель может попросить внести небольшие изменения в код, чтобы проверить степень владения кодом и его авторство, а также задавать любые вопросы в рамках выполненной работы. Рекомендуется оценивать работу, сделанную на минимальные требования в 70% от максимальной оценки, при условии что нет сомнения в авторстве работы или её частей.

Более высокую оценку можно поставить за **выполнение сложной версии задания**, а при отсутствии градаций — **за оригинальность подхода, элегантность или изобретательность при выполнении работы.**

В качестве критериев оценки не рекомендуется учитывать следующие свойства работы:

1. **Качество обработки данных.** Достижение низкой ошибки в работе классификатора или генерация качественного текста зависит от удачи и от того, насколько много времени потрачено на оптимизацию макропараметров. При этом объём знаний, получаемых в единицу времени в процессе такой оптимизации падает тем больше, чем дольше она проводится, а трудозатраты — наоборот, быстро растут. Также качество обработки будет заведомо высоким у списанной работы, что может вынудить большинство студентов списывать, а тех, кто отказывается от этого — получать более низкие оценки, чем списывающие. Это крайне нежелательное явление. Рекомендуется

пожертвовать возможными преимуществами этого критерия — простотой и объективностью — для предотвращения этих нежелательных последствий.

2. **Небольшая просрочка сдачи работы** — не все работы в этом курсе одинаковы по объёму. Поэтому студенту может потребоваться больше времени на некоторые задания. Допустимо сдавать работу на 2-4 недели позже срока. За это не рекомендуется снижать оценку. Повышенную оценку за раннюю сдачу также не рекомендуется ставить, исключение — первая сданная работа группе. Однако работы, сданные после последнего занятия в семестре по расписанию в часы ликвидации академической задолженности следует оценивать уже со значительным штрафом, тем не менее позволяющим получить удовлетворительную оценку. Студент должен сдать работу в часы, выделенные для этого. Если работу не получается сделать идеально — лучше сдать несовершенную работу, чем не успеть до наступления часов ликвидации задолженности.
3. **Выбор технологии.** Студент должен выбрать одну из имеющихся технологий или библиотек. Его задача — не понять конкретную библиотеку, которая может устареть до окончания обучения, а общие принципы, которые останутся надолго. Поэтому даже выбор плохой, по мнению преподавателя, технологии не должен являться основанием для снижения оценки, если результат был хоть как-то достигнут.

Студенты могут ссылаться на данный раздел во время защиты лабораторной работы, если они с ним ознакомились заранее. Однако, они не могут ссылаться на него после защиты — чтобы оспорить оценку, если не вспомнили о критериях на защите. Такие апелляции не полезны для учебного процесса, так как загружают преподавателей и студентов непроизводительной работой.

Применение стобальной шкалы

Для получения оценки «отлично» необходимо набрать 86 баллов. Имеется всего 8 обязательных лабораторных работ, их рекомендуется оценивать на 11 баллов каждую. Это позволит студенту при выполнении только обязательных работ получить оценку «отлично», если он нигде не потеряет баллы, или потеряет не более двух баллов в сумме.

Задания для самостоятельной работы рекомендуется оценивать на 12 баллов каждое. Это теоретически позволит получить студенту 100 баллов при выполнении всех лабораторных и одного самостоятельного задания.

При накоплении достаточного количества баллов для достижения нужной оценки рекомендуется её выставить при желании студента, даже если не все работы выполнены. Ввиду структуры данного курса пропуск любой из обязательных работ всё равно приведет к формированию нужных компетенций при выполнении остальных работ. При этом трудозатраты и преподавателя и студента при выполнении последней работы «для галочки» не будут производительными.

Ставить баллы за посещение работ в курсе магистратуры может быть нецелесообразным. Магистранты уже обладают достаточными навыками для освоения работ самостоятельно, если они выберут такой путь. Однако, это

зависит от группы и набора, а также от стиля преподавания команды преподавателей. Поэтому такие баллы являются допустимыми, при этом можно сократить количество баллов за обязательную работу до 10, а 8 баллов оставить на посещаемость.

Использование искусственного интеллекта для выполнения заданий

Использование систем искусственного интеллекта, таких как ChatGPT, для выполнения лабораторных работ может быть полезным по нескольким причинам, если подходить к этому процессу разумно и осознанно. Вот несколько причин и советов о том, как использовать такие системы для получения знаний:

Быстрый доступ к информации — ИИ может предоставить быстрые и точные ответы на теоретические вопросы, что помогает сократить время на поиск информации в учебниках или в интернете.

Объяснение сложных концепций — системы ИИ могут объяснить сложные понятия более простым и понятным языком, адаптированным к вашему уровню знаний.

Помощь в программировании и анализе данных — ИИ может помочь с написанием кода, разбором ошибок, предоставлением примеров кода и алгоритмов, что особенно полезно для лабораторных работ по программированию.

Генерация идей и подходов — ИИ может предложить различные подходы к решению задач, что помогает развивать творческое мышление и находить более эффективные решения.

Поддержка в структурировании и оформлении работы — ChatGPT может помочь с составлением плана работы, структурированием отчета и даже с оформлением документации в соответствии с требованиями.

Используйте ИИ для объяснений, а не для выполнения — спрашивайте у ChatGPT объяснения понятий и методов, а не готовые ответы. Например, попросите объяснить, как работает тот или иной алгоритм, или почему используется определенный метод.

Проверяйте и анализируйте ответы — всегда проверяйте информацию, полученную от ИИ. Используйте её как отправную точку для более глубокого изучения темы. Сравнивайте ответы с учебниками и другими авторитетными источниками.

Используйте для самопроверки — после самостоятельного выполнения задания, вы можете использовать ChatGPT для проверки ваших решений и поиска ошибок. Это поможет вам увидеть, где вы допустили ошибки и как их исправить.

Развивайте навыки самостоятельного поиска информации — используйте ChatGPT для улучшения навыков поиска и анализа информации. Например, попросите ИИ дать список литературы или источников по теме и самостоятельно изучайте их.

Интерактивное обучение — задавайте уточняющие вопросы и проводите диалоги с ИИ. Это поможет лучше понять материал и закрепить знания.

Использование ChatGPT и других подобных систем может значительно облегчить процесс выполнения лабораторных работ и углубить понимание учебного материала, если использовать их правильно. Важно помнить, что

такие системы должны служить дополнением к обучению, а не заменой самостоятельного изучения и практики. Недобросовестное использование систем искусственного интеллекта приводит к проблемам при защите лабораторных работ, а правильное использование — к значительному улучшению всех показателей обучения.

Заключение

В этом сборнике лабораторных работ мы рассмотрели основные методы и подходы как математической лингвистики, так и обработки естественных языков (МЛиОЕЯ). Каждая лабораторная работа была направлена на изучение конкретного аспекта данных областей, начиная от работы с регулярными выражениями и заканчивая генерацией текста. В рамках лабораторных работ мы изучили использование регулярных выражений для поиска и замены, что позволило нам эффективно работать с текстовыми данными и выполнять сложные текстовые манипуляции.

Затем мы перешли к изучению контекстно-свободных грамматик, что дало нам понимание более сложных структур текста и позволило разбирать предложения на составляющие. Мы рассмотрели методы токенизации текста, что является важным этапом при подготовке данных для дальнейшего анализа. В процессе изучения лемматизации, стемминга и векторизации мы узнали, как преобразовывать текст в числовые представления, что является необходимым шагом для применения алгоритмов машинного обучения.

Особое внимание было уделено подбору датасета для машинного обучения и обучению классификатора, что позволило нам понять, как на практике применять машинное обучение для решения задач МЛиОЕЯ. Мы также рассмотрели методы генерации текста, что позволило нам создавать новые текстовые данные на основе заданных условий.

Кроме обязательных лабораторных работ, были предложены необязательные задания для самостоятельной работы. Изучение принципа работы трансформеров и распознавания речи дало нам возможность расширить свои знания и навыки в области современных технологий ОЕЯ.

Таким образом, пройденные лабораторные работы заложили прочную основу для дальнейшего изучения и практического применения методов математической лингвистики и обработки естественных языков. Полученные навыки могут быть применены в различных областях, таких как автоматическая обработка отзывов пользователей, анализ новостных потоков и разработка чат-ботов. Обработка естественных языков и математическая лингвистика являются ключевыми компонентами в развитии технологий искусственного интеллекта, и их значение будет только расти в будущем. Надеемся, что полученные знания и навыки будут полезны в вашей дальнейшей профессиональной деятельности и станут стимулом для новых открытий в этих увлекательных областях.

Приложение 1: Компетенции согласно ФГОС

Для освоения данной дисциплины требуются следующие компетенции, сформированные при освоении программы бакалавриата:

ОПК-1. Способен применять естественнонаучные и общеинженерные знания, методы математического анализа и моделирования, теоретического и экспериментального исследования в профессиональной деятельности;

ОПК-5. Способен инсталлировать программное и аппаратное обеспечение для информационных и автоматизированных систем;

Также требуются компетенции, сформированные или формируемые в процессе освоения программы магистратуры:

УК-1. Способен осуществлять критический анализ проблемных ситуаций на основе системного подхода, вырабатывать стратегию действий;

ОПК-3. Способен анализировать профессиональную информацию, выделять в ней главное, структурировать, оформлять и представлять в виде аналитических обзоров с обоснованными выводами и рекомендациями;

ОПК-4. Способен применять на практике новые научные принципы и методы исследований;

Изучение курса способствует формированию следующих компетенций:

УК-4. Способен применять современные коммуникативные технологии, в том числе на иностранном(ых) языке(ах), для академического и профессионального взаимодействия;

ОПК-2. Способен разрабатывать оригинальные алгоритмы и программные средства, в том числе с использованием современных интеллектуальных технологий, для решения профессиональных задач;

ОПК-5. Способен разрабатывать и модернизировать программное и аппаратное обеспечение информационных и автоматизированных систем;

ОПК-6. Способен самостоятельно приобретать с помощью информационных технологий и использовать в практической деятельности новые знания и умения, в том числе в новых областях знаний, непосредственно не связанных со сферой деятельности;

ОПК-7. Способен применять при решении профессиональных задач методы и средства получения, хранения, переработки и трансляции информации посредством современных компьютерных технологий, в том числе, в глобальных компьютерных сетях;

Приложение 2: Связь с другими дисциплинами

Этот курс практических занятий предназначен для использования в программе подготовки магистров. Он опирается на дисциплины, изучаемые в бакалавриате, а также изучаемые параллельно с ним в магистратуре.

В программу подготовки бакалавриата входят курсы «теория алгоритмических языков и методов трансляции», «алгоритмы и структуры данных», «человеко-машинное взаимодействие».

Дисциплина "Теория алгоритмических языков и методов трансляции" предоставляет важные знания и навыки, которые могут быть полезны при изучении "Математической лингвистики и обработки естественных языков" (МЛиОЕЯ). Формальные языки и грамматики, например контекстно-свободные и регулярные, используются для моделирования синтаксической структуры естественных языков. Эти грамматики помогают в синтаксическом анализе и разборе предложений, а регулярные выражения и конечные автоматы позволяют эффективно обрабатывать и искать текстовую информацию. Методы синтаксического анализа, такие как нисходящий и восходящий анализ, применяются для синтаксического разбора предложений, а построение синтаксических деревьев помогает в представлении структуры предложений для дальнейшей семантической и прагматической обработки.

Конечные автоматы и автоматы с магазинной памятью используются для распознавания шаблонов в текстах и анализа структуры текста, что особенно

важно при работе с вложенными конструкциями. Методы лексического анализа применяются для токенизации текста, разделения на слова и другие элементы, что является важным этапом предобработки текстов перед синтаксическим анализом. Методы трансляции также полезны для преобразования текстов из одной формы в другую и оптимизации текстовых данных для повышения эффективности их обработки.

Семантические действия и атрибутированные грамматики позволяют присваивать значения синтаксическим структурам и создавать семантические модели текста. Промежуточные представления, такие как абстрактные синтаксические деревья (AST), используются для хранения промежуточных представлений текстов и упрощают дальнейшую обработку и анализ.

Практическое применение знаний о компиляторах и интерпретаторах позволяет разрабатывать программы для анализа и обработки естественных языков, включая системы машинного перевода и чат-боты. Таким образом, знания из дисциплины "Теория алгоритмических языков и методов трансляции" обеспечивают необходимую базу для эффективного использования алгоритмических и математических подходов при обработке естественных языков, что является основой для успешного изучения и практического применения математической лингвистики.

Дисциплина "Алгоритмы и структуры данных" предоставляет множество знаний и навыков, которые могут быть полезны при изучении дисциплины "Математическая лингвистика и обработка естественных языков" (МЛиОЕЯ). Основные структуры данных, такие как массивы и списки, используются для хранения и обработки последовательностей слов и символов, позволяя эффективно выполнять операции вставки, удаления и поиска элементов в текстовых данных. Хеш-таблицы обеспечивают быстрый доступ к данным и

могут использоваться для создания словарей, хранения частотных словарей и поиска слов. Деревья, например, префиксные или суффиксные деревья, используются для представления и обработки текстовых данных, таких как поиск подстрок и автодополнение. Графы могут применяться для моделирования связей между словами и понятиями в текстах, например, для анализа социальных сетей и построения семантических сетей.

Алгоритмы поиска играют важную роль. Линейный и бинарный поиск используются для нахождения слов и фраз в текстовых данных, а алгоритмы поиска подстрок применяются для быстрого поиска шаблонов в текстах. Динамическое программирование используется для решения задач, связанных с обработкой естественных языков, таких как выравнивание последовательностей и анализ редактируемых расстояний, например, расстояние Левенштейна. Графовые алгоритмы, такие как поиск в ширину, поиск в глубину и алгоритм Дейкстры, могут применяться для анализа семантических сетей, построения деревьев синтаксического анализа и других задач, связанных с обработкой текстов. Комбинаторные алгоритмы помогают в решении задач, связанных с генерацией всех возможных комбинаций и перестановок слов и символов, что полезно для анализа текстов и создания языковых моделей.

Строковые алгоритмы важны для задач, связанных с обработкой текстов, таких как поиск подстрок и индексирование текстов. Алгоритмы для обработки больших данных применяются для обработки больших корпусов

текстов и выполнения вычислений на распределённых системах. Знания и навыки из дисциплины "Алгоритмы и структуры данных" обеспечивают прочную основу для эффективной работы с текстовыми данными в рамках математической лингвистики и обработки естественных языков. Они позволяют разрабатывать и применять эффективные методы и инструменты для анализа, обработки и интерпретации текстовой информации.

Курс «человеко-машинное взаимодействие» знакомит студентов с проблемой построения интерфейсов, а курс "Математическая лингвистика и обработка естественных языков" предоставляет средства для одного из самых распространенных решений этой проблемы — интерфейса естественных языков.

В программу подготовки магистратуры входят курсы «системы искусственного интеллекта», «нейронные сети».

Курс "Системы искусственного интеллекта" тесно связан с курсом "Математическая лингвистика и обработка естественных языков" (МЛиОЕЯ) по нескольким ключевым аспектам. Эти два курса пересекаются в использовании методов машинного обучения, анализа данных и алгоритмов для решения задач обработки естественного языка. В курсе "Системы искусственного интеллекта" изучаются различные модели машинного обучения, такие как регрессия, классификация, кластеризация и их применение к анализу текстов в МЛиОЕЯ. Например, классификаторы могут использоваться для автоматического определения тематики текстов или для задач сентимент-анализа. Современные подходы к обработке естественного языка часто используют глубокие нейронные сети, такие как рекуррентные нейронные сети (RNN), длинные краткосрочные памяти (LSTM) и трансформеры (например, модель BERT). Эти модели изучаются в курсе ИИ и

находят прямое применение в МЛиОЕЯ для задач таких, как машинный перевод, генерация текста и распознавание речи.

В курсе ИИ рассматриваются различные алгоритмы и эвристики, такие как генетические алгоритмы и алгоритмы роя частиц, которые могут быть применены к оптимизации задач в МЛиОЕЯ. Методы обучения с учителем (supervised learning) и без учителя (unsupervised learning), такие как кластеризация текстов и обучение без учителя для задачи выделения тем, изучаются в курсе ИИ и используются в МЛиОЕЯ.

Курс «нейронные сети» даёт более глубокое представление об использовании нейронных сетей для решения различных задач, а также об их архитектуре. Данный курс позволяет студентам понять, как применить полученные знания из курса нейронных сетей.

Содержание