**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук Департамент программной инженерии

УТВЕРЖДАЮ

Академический руководитель образовательной программы «Программная инженерия»,

профессор департамента программной инженерии, канд. техн. наук

В.В. Шилов

« » 2018 г.

# Выпускная квалификационная работа

на тему

# МОБИЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ МОЛОДОЙ МАМЫ

по направлению подготовки 09.03.04 «Программная инженерия»

|  |  |
| --- | --- |
| Научный руководитель | Выполнила  студентка группы БПИ143 4 курса бакалавриата  образовательной программы  «Программная инженерия» |
| Доцент факультета компьютерных наук |
| Должность |
| кандидат технических наук |  |
| Ученая степень |  |
| Римма Закиевна Ахметсафина | Анастасия Андреевна Репина |
| И.О. Фамилия | И.О. Фамилия |
| Подпись, дата | Подпись, дата |

Москва 2018

Реферат

Natural Language Toolkit (NLTK) является ведущей библиотекой на языке программирования Python для обработки естественного языка. Библиотека предоставляет открытый доступ к широкому набору инструментов для обработки текстов, часто используется в научных исследованиях, а также служит как учебное пособие для студентов. Поскольку существующие в библиотеке NLTK синтаксические анализаторы (парсеры) либо устарели, либо написаны на других языках программирования, в этой работе предлагается реализация быстрого и точного парсера зависимостей для включения в библиотеку.

В данной работе на языке программирования Python реализуется и интегрируется в NLTK алгоритм синтаксического анализа, созданный Кристофером Д. Мэннингом и Данчи Чен. В алгоритме используются новаторские концепции, и, следовательно, он имеет большую научную ценность. На основе алгоритма лежит система переходов Arc-standard, и для предсказания переходов используется искусственная нейронная сеть. Кроме того, вместо унитарного кодирования парсер использует плотные представления для слов, меток частей речи и синтаксических зависимостей. Другим примечательным нововведением является использование в нейросети кубической функции активации вместо широко используемых функций tanh и sigmoid.

Работа содержит 37 страниц, 3 главы, 11 рисунков, 7 таблиц, 5 схем, 39 источников, 5 приложений.

***Ключевые слова*** *— мобильное приложение, Android, носимые устройства, здоровье.*

Abstract

The Natural Language Toolkit (NLTK) is a leading suite of libraries for natural language processing, and with the rise of artificial intelligence, its popularity has only increased in the recent years. Since existing parsers in NLTK are either outdated or not implemented in Python, its primary programming language, this project proposes an implementation of a fast and accurate dependency parser to include in the toolkit.

In this work, the transition-based parsing algorithm created by Danqi Chen and Christopher D. Manning is implemented and incorporated into NLTK. The algorithm uses artificial neural networks for predicting transitions and introduces several groundbreaking concepts. The parser employs an efficient new way of feature representation, particularly for part-of-speech tags and arc labels. Instead of one-hot encoding, dense representations are used for words, tags, and labels. Another innovation is the use of cube activation function in the neural network classifier instead of the commonly used tanh and sigmoid functions.

***Index Terms*** *— natural language processing, dependency parsing, artificial neural networks, NLTK.*

Термины и определения

Грамматика зависимостей – теория синтаксической структуры предложения. Согласно грамматике зависимостей, связи между членами предложении рассматриваются как подчинительные, а сказуемое признаётся вершиной предложения. В результате синтаксического анализа на основе грамматики зависимостей строится дерево синтаксических зависимостей

Грамматика составляющих – теория синтаксической структуры предложения. Согласно грамматике составляющих, сложные грамматические единицы предложения складываются из более простых непересекающихся грамматических единиц.

Корпус текстов – собрание текстов, размеченных согласно определенным принципам и стандартам.

Семантическое отношение – смысловое отношение между словами.

Синтаксический анализ (парсинг) – процесс сопоставления последовательности слов и знаков пунктуации с грамматикой языка.

Синтаксический анализатор (парсер) – программа, выполняющая синтаксический анализ.

Синтаксическое дерево (дерево разбора) – результат синтаксического анализа предложения.

Система переходов – концепция, используемая при синтаксическом анализе, которая из предложения за конечное количество переходов строит его дерево разбора.

Составляющая – грамматическая единица предложения, которая может быть составлена из составляющих меньшего размера.

Токен – название членов предложения (слово или знак пунктуации), используемых в алгоритмах частеречной разметки или синтаксического анализа.

Тэггер – программа, выполняющая частеречную разметку текста.

Частеречная разметка – процесс сопоставления последовательности слов и знаков пунктуации с их частью речи.

UAS, unlabeled attachment score – процент слов в корпусе, для которых правильно определен родитель в дереве зависимостей.

LAS, labeled attachment score – процент слов в корпусе, для которых правильно определены родитель в дереве зависимостей и тип зависимости от родителя.

Оглавление

Реферат 2

Abstract 3

Термины и определения 4

Введение 6

Глава 1. Парсер зависимостей 9

* 1. Существующие подходы и решения 9
  2. Обзор аналогов NLTK 12
  3. Методы синтаксического анализа NLTK 14

Выводы по главе 15

Глава 2. Алгоритмы, модели и данные 16

* 1. Система переходов arc-standard 16
  2. Использование нейронной сети для классификации 18
  3. Генерация обучающей выборки 15
  4. Представление признаков и функция активации 20
  5. Частеречная разметка 21

Выводы по главе 21

Глава 3. Программная реализация 22

* 1. Структура проекта программы 22
     1. Парсер зависимостей 22
     2. Система переходов arc-standard 24
     3. Классификатор 25
  2. Интеграция парсера в NLTK 25
  3. Обучение парсера 26
  4. Определение качества парсера 27

Заключение 31

Список использованных источников 33

Приложение A. Техническое задание 38

Приложение B. Руководство оператора 53

Приложение C. Программа и методика испытаний 64

Приложение D. Текст программы 79

Приложение E. Диаграмма вариантов использования 85

Введение

В социальных сетях, поисковых и рекомендательных системах все чаще встречаются успешные применения алгоритмов искусственного интеллекта. Этот успех во многом можно отнести к достижениям в области обработки естественного языка (NLP), одной из основ искусственного интеллекта, а также использованию искусственных нейронных сетей. NLP позволяет компьютерам извлекать информацию из текстов, предоставляет методы машинного перевода, распознавания речи и оптического распознавания символов.

При обработке текстов или человеческой речи, важным компонентом являются синтаксические анализаторы (парсеры). Цель парсеров – определение синтаксической структуры предложения. Одной из наиболее популярных моделей, используемых в современных парсерах, является грамматика зависимостей. В грамматике зависимостей все отношения между членами предложения считаются подчинительными. Структура предложения представляется в виде дерева зависимостей – иерархии членов предложения, где на вершине сказуемое, а остальные слова его подчиненные.

Анализ синтаксических зависимостей между словами в предложениях играет важную роль при работе с данными на естественном языке. Машинные переводчики используют алгоритмы разбора для определения зависимостей в предложениях и извлечения информации из текстов. Учитывая значительную роль этих алгоритмов в извлечении информации, разработчики и лингвисты активно работают над созданием эффективных парсеров. C каждым годом появляются более точные и быстрые алгоритмы.

NLTK - это ведущая платформа с открытым исходным кодом на языке программирования Python, предоставляющая инструменты для обработки естественного языка. Несмотря на многие аналогичные проекты, NLTK выделяется благодаря сочетанию высококачественных алгоритмов с учебными материалами и учебниками. Эта платформа широко используется по всему миру в исследованиях в области NLP и в разработке алгоритмов машинного обучения. Поскольку код библиотеки открыт и тщательно документирован, платформа имеет важное образовательное значение и является самым популярным руководством для начинающих ученых в этой области.

Аналогами NLTK считаются spaCy, SyntaxNet от Google и библиотека CoreNLP от Стэнфордского университета. Основной язык программирования библиотеки CoreNLP – Java, и реализованные в ней алгоритмы слишком сложны для начинающих разработчиков в области NLP. Пакет инструментов spaCy реализован на более популярном среди начинающих языке Python, но ориентируется на другую аудиторию, нежели NLTK. SyntaxNet имеет те же

недостатки, что и CoreNLP. Кроме того, NLTK имеет более широкий набор инструментов и опережает перечисленные аналоги по качеству документации.

Так как область искусственного интеллекта развивается быстро, современные алгоритмы значительно быстрее и точнее, чем реализованные в NLTK парсеры. На данный момент, NLTK предоставляет только интерфейсы к внешним реализациям MaltParser и стэнфордского парсера, которые могут быть использованы только после установки дополнительных языков и программ, а также не имеют образовательное значение. В библиотеке также реализованы алгоритмы анализа составляющих и зависимостей, основанные на правилах и выполняющие строго образовательную функцию. С точки зрения научной ценности, последние алгоритмы не представляют особого интереса, так как сильно отстают от современных подходов к синтаксическому анализу. Такие парсеры могут претендовать на успех только при наличии большой группы высококвалифицированных специалистов, которые придумали бы необходимые лингвистические правила.

Чтобы устранить отсутствие современного парсера в библиотеке NLTK, этот проект реализует парсер зависимостей, предложенный Кристофером Мэннингом и Данчи Чен [1], и внедряет его в библиотеку. На основе реализуемого парсера – система переходов arc-standard. Принимая во внимание успехи искусственных нейронных сетей в решениях разных задач NLP, для предсказывания переходов парсер использует нейронную сеть. Нейронная сеть обучается на размеченном корпусе деревьев зависимостей. Во время классификации, для лексических признаков используются плотные представления вместо унитарного кодирования, что значительно ускоряет работу парсера. Еще одним нововведением является использование кубической функции активации в нейронной сети вместо более популярных гиперболического тангенса и сигмоида. Это позволяет лучше моделировать связи между входными признаками.

Во время экспериментов, проведенных на английском корпусе The Penn Treebank, парсер продемонстрировал более высокую точность, чем парсеры MST и Malt, а именно 92% UAS и 90% LAS. Этот алгоритм демонстрирует также значительное улучшение в скорости извлечения признаков и обработки предложений. Этот парсер обрабатывает тексты почти в 100 раз быстрее парсера MST.

Целью этой работы является разработка парсера зависимостей на языке программирования Python и интеграция этого парсера в библиотеку NLTK. Для достижения цели необходимо решить следующие задачи:

1. Изучение предметной области, анализ существующих решений.
2. Выбор средств реализации и определение форматов входных и выходных данных.
3. Реализация алгоритма синтаксического анализа.
4. Определение точности парсера для русскоязычных текстов.
5. Обучение парсера на корпусе русскоязычных текстов.
6. Интеграция парсера в NLTK.
7. Добавление модели парсера для русского языка в репозиторий NLTK Data.
8. Разработка пояснительной записки и технической документации.

Далее в этой работе рассмотрены существующие подходы к синтаксическому анализу зависимостей, обзор и анализ аналогов (Глава 1), описание выбранного алгоритма синтаксического анализа (Глава 2) и описание программной реализации (Глава 3). В заключении подводится итог и обсуждаются пути дальнейшей работы.

В Приложениях к работе приведена техническая документация и диаграмма вариантов использования.

Глава 1. Парсер зависимостей

В данной главе рассматривается предметная область синтаксического анализа текстов на основе деревьев зависимостей. Сначала описаны существующие подходы к синтаксическому анализу, их недостатки и преимущества. Далее рассмотрены основные аналоги NLTK, предоставляющие методы синтаксического анализа. В заключительной части главы приведены описания существующих в NLTK парсеров и их сравнение с реализуемым алгоритмом.

* 1. Существующие подходы и решения

Знание синтаксической структуры предложения необходимо при решении таких задач, как генерация текста, извлечение информации, определение тональности текста и машинный перевод. Исследования показывают, что знание зависимостей между членами предложения значительно улучшает точность перевода [2]. Парсеры также применяются для извлечения семантических отношений [3].

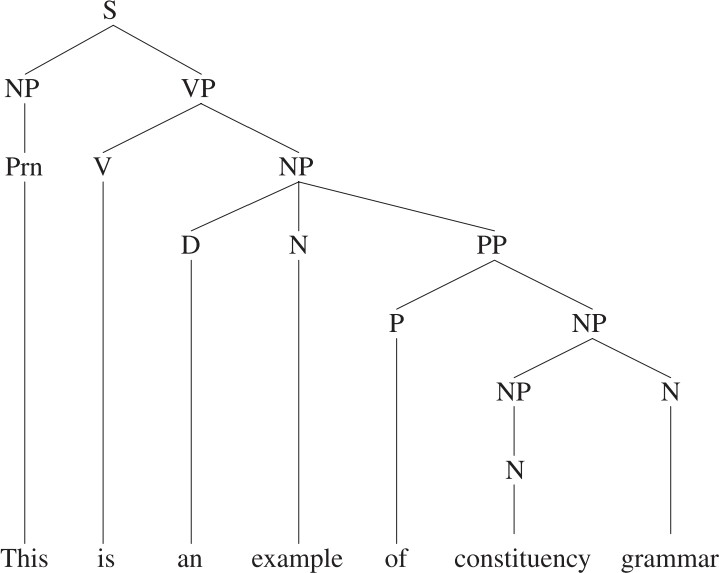
Для описания синтаксической структуры предложения используются грамматика составляющих или грамматика зависимостей. В грамматике составляющих строится иерархия составляющих, где слова в предложении группируются на основе их части речи в именные, глагольные, прилагательные, предложные и наречные группы. Каждая группа состоит из более простых, не пересекающихся единиц, а вершиной группы считается слово соответствующей части речи (например, именная группа возглавляется существительным).

Рисунок 1. Дерево составляющих предложения «This is an example of constituency

grammar»

На рис. 1 изображено дерево составляющих предложения «This is an example of constituency grammar» [4]. Всё предложение считается одной большой составляющей (вершина S), которая в свою очередь состоит из именной (вершина NP) и глагольной (вершина VP) групп.

В грамматике зависимостей все отношения между членами предложения считаются подчинительными. Структура предложения представляется в виде дерева зависимостей, иерархии членов предложения, где на вершине сказуемое, а остальные слова – его подчиненные [5].

На рис. 2 изображены зависимости в предложении «Bills on ports and immigration were submitted by Senator Brownback, Republican of Kansas». В этом представлении, члены предложения являются вершинами дерева зависимостей, а типы зависимостей между членами указывается на соответствующих рёбрах дерева.

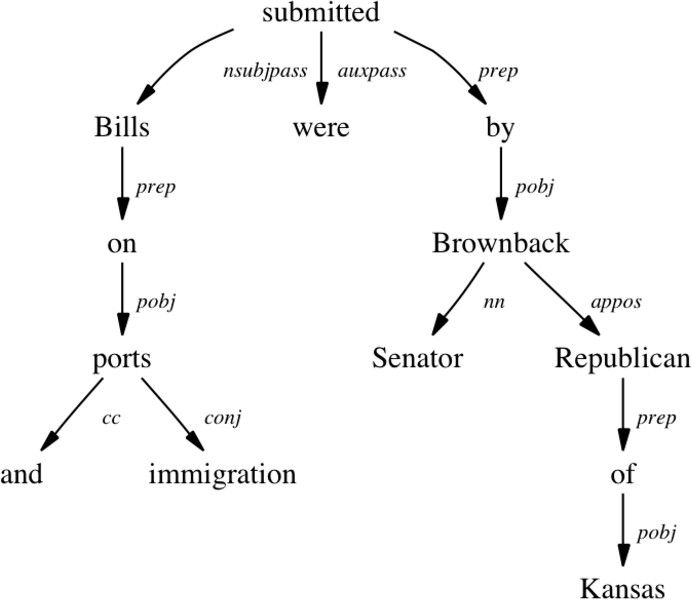


Рисунок 2. Дерево зависимостей предложения «Bills on ports and immigration were submitted by Senator Brownback, Republican of Kansas» [6]

На рисунках 3 и 4 изображены альтернативные способы представления дерева зависимостей на примере предложений «Economic news had little effect on financial markets.» и «This is an example of dependency grammar». В представлении из рис. 3, зависимость между членами изображена в виде дуги, соединяющей эти члены. Тип зависимости указывается на дуге.

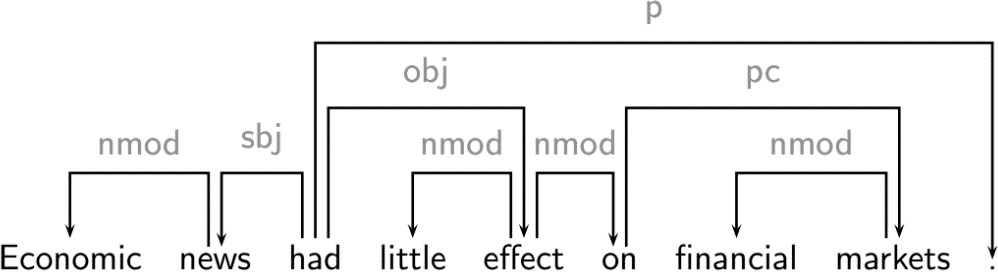


Рисунок 3. Дерево зависимостей предложения «Economic news had little effect on

financial markets.» [7]

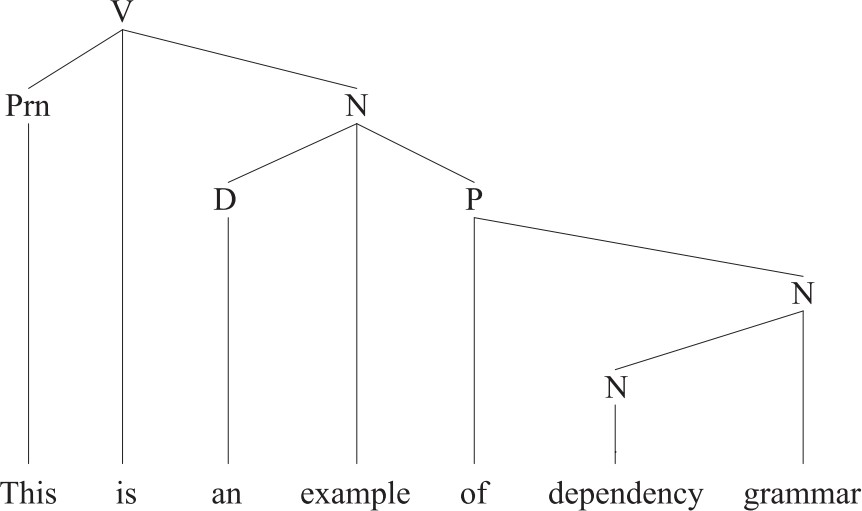
На рис. 4 представлено дерево зависимостей предложения «This is an example of dependency grammar». В этом представлении, вместо членов предложения на вершинах указывается их часть речи. Например, на вершине дерева вместо сказуемого используется метка V, указывающая на часть речи слова. В данном примере типы зависимостей не указаны.

Рисунок 4. Дерево разбора предложения «This is an example of dependency grammar» [4]

Синтаксический анализ на основе деревьев зависимостей в последние годы становится все более популярным по сравнению с анализом составляющих. Одной из возможных причин растущего интереса является то, что грамматика зависимостей лучше, чем грамматика составляющих, подходит для анализа языков, которые демонстрируют свободный или гибкий порядок слов и где лексические и морфологические признаки играют более важную роль при определении синтаксической структуры предложения, чем порядок слов [8].

По сравнению с грамматикой составляющих, парсеры зависимостей имеют следующие преимущества [4]:

* 1. Связи зависимостей между словами близки к их семантическим отношениям, и нет необходимости извлекать эти отношения из дерева, в котором они не указаны явно.
  2. В дереве зависимостей каждому слову из предложения соответствует только одна вершина.
  3. Парсер зависимостей обрабатывает члены предложения «по одному», в отличие от парсеров составляющих, где обработка члена предложения заканчивается только после нахождения всех групп, к которым он принадлежит. Исследования также показывают [9], что человеческий разум обрабатывает предложения примерно по такому же принципу, что и парсеры зависимостей.
  4. Существующие алгоритмы анализа зависимостей значительно быстрее, чем алгоритмы анализа составляющих [10].

Для извлечения синтаксической структуры предложения используются статистические методы, а также методы, основанные на правилах. Для точной работы парсеров, основанных на правилах, требуется большой набор лингвистических правил. Эти правила обычно создаются высококвалифицированными лингвистами, и для создания набора правил, необходима большая группа специалистов. Для создания точного статистического парсера такая группа не нужна. Статистические парсеры обучаются на размеченных корпусах и показывают конкурентоспособную точность (парсер SyntaxNet показывает точность 94.61% UAS на корпусе WSJ [11]).

Современные статистические парсеры работают на основе систем переходов arc- standard и arc-eager [1][11][12]. Парсер, созданный Кристофером Мэннингом и Данчи Чен, основан на системе arc-standard и успешно применяет нейронную сеть для предсказания переходов. В нейронной сети плотные представления слов, меток частей речи и зависимостей используются как признаки. Ранние попытки использовать нейронные сети в парсерах не были настолько удачными, так как использовали унитарное кодирование для представления признаков [13][14][15][16].

* 1. Обзор аналогов NLTK

NLTK - библиотека с открытым исходным кодом на языке программирования Python, предоставляющая инструменты для обработки естественного языка. Библиотека используется в исследованиях в области NLP. Поскольку код библиотеки открыт и тщательно документирован, платформа имеет важное образовательное значение и является самым популярным руководством для начинающих ученых в этой области [17].

Аналогами NLTK считаются spaCy, SyntaxNet от Google и библиотека CoreNLP от Стэнфордского университета. По сравнению с аналогами, NLTK имеет более широкий набор инструментов, тщательно документирован, а также сопровождается учебником c

примерами и описаниями алгоритмов. Еще одним существенным преимуществом NLTK является то, что кроме английского, она предлагает готовые решения для многих других языков.

CoreNLP – набор инструментов для обработки естественного языка [18]. Эта библиотека предоставляет алгоритмы токенизации, частеречной разметки, синтаксического анализа и извлечения именованных сущностей. Однако, основной язык программирования библиотеки CoreNLP – Java, и реализованные в ней алгоритмы слишком сложны для начинающих разработчиков в области NLP. Модуль parser, в котором хранятся реализации алгоритмов анализа, состоит из 187 файлов и более 50000 строк кода (в версии 3.6.0). Данная библиотека уступает NLTK также в разнообразии предоставляемых алгоритмов. В отличие от CoreNLP, для каждой задачи NLP в NLTK можно найти несколько решений с использованием как простых, так и более сложных подходов.

Пакет инструментов spaCy реализован на более популярном среди начинающих языке Python, но ориентируется на другую аудиторию, нежели NLTK [12]. Алгоритмы в этой библиотеке превосходят свои аналоги из NLTK в производительности и ориентированы на использование в промышленных проектах. Библиотека NLTK, в свою очередь, предназначена в основном для использования в исследовательских и образовательных целях.

SyntaxNet имеет те же недостатки, что и CoreNLP [19]. Примечательно, что в парсере SyntaxNet используются новаторские идеи, введенные в реализуемом в данной работе парсере [11].

В таблице 1 приведен список основных инструментов, предоставляемых упомянутыми библиотеками. Библиотеки сравниваются по наличию основных инструментов NLP – токенизатора слов (tokenization) и предложений (sentence segmentation), парсера зависимостей, алгоритмов частеречной разметки (part-of-speech tagging) и извлечения именованных сущностей (entity recognition). Также сравнивается поддержка нескольких естественных языков и простота установки. Как видно, все основные аналоги NLTK предоставляют парсер зависимостей, что еще раз подчеркивает актуальность данной работы.

Таблица 1.

Сравнение библиотек spaCy, SyntaxNet, NLTK и CoreNLP [20]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | SPACY | SYNTAXNET | NLTK | CORENLP |
| Легкость установки | + | - | + | + |
| Языка программирования Python | + | - | + | - |
| Поддержка разных языков | - | + | + | + |
| Токенизация | + | + | + | + |
| Частеречная разметка | + | + | + | + |
| Определение границ предложений | + | + | + | + |
| **Парсер зависимостей** | **+** | **+** | **-** | **+** |
| Извлечение именованных сущностей | + | - | + | + |

* 1. Методы синтаксического анализа NLTK

На данный момент, NLTK предоставляет только интерфейсы к внешним реализациям MaltParser и стэнфордского парсера [21], которые могут быть использованы только после установки дополнительных языков и программ, а также не имеют образовательное значение.

MaltParser работает на основе системы переходов, используя методы машинного обучения (LIBSVM и LIBLINEAR ) для определения переходов из одной конфигурации в другую [22]. Для использования MaltParser через NLTK, необходимо установить Java, скачать и установить парсер, и задать переменную среды MALT\_PARSER [23].

Стэнфордский пакет содержит несколько парсеров, главный из который работает на основе стохастической контекстно-свободной грамматики. Для использования этого парсера через NLTK, необходимо установить Java, скачать и распаковать пакет, добавить в переменную CLASSPATH пути к файлам .jar и задать переменную среды STANFORD\_MODELS. NLTK предназначен для Python-разработчиков, а MaltParser и стэнфордский парсер написаны на языке программирования Java, поэтому их реализация будет непонятна для тех, кто знает только Python.

В библиотеке также реализованы алгоритмы анализа составляющих и зависимостей, основанные на правилах и выполняющие образовательную функцию. В модуле parse содержатся следующие парсеры:

1. ShiftReduceParser - парсер составляющих, работающий на основе контекстно- свободной грамматики. Парсер на вход принимает описание грамматики и используя правила из этой грамматики, строит разбор предложения.
2. ViterbiParser – парсер составляющих, выполняющий восходящий анализ на основе стохастической контекстно-свободной грамматики.
3. ProjectiveDependencyParser – парсер зависимостей на основе правил. В правилах указывается между какими словами могут быть зависимости.
4. ProbabilisticProjectiveDependencyParser – парсер зависимостей на основе правил. В отличие от парсера ProjectiveDependencyParser, здесь у правил есть вероятности. Парсер выводит правила и их вероятность из размеченного корпуса текстов.

С точки зрения научной ценности, перечисленные алгоритмы не представляют особого интереса, так как сильно отстают от современных подходов к синтаксическому анализу. Такие парсеры могут претендовать на успех только при наличии большой группы высококвалифицированных специалистов, которые придумали бы необходимые лингвистические правила.

Выводы по главе

В первой главе была рассмотрена предметная область синтаксического анализа текстов. Были описаны существующие подходы к синтаксическому анализу и преимущества парсеров на основе грамматик зависимостей по сравнению с грамматикой составляющих. С целью подтверждения актуальности работы также были проанализированы основные аналоги NLTK и недостатки существующих в библиотеке методов анализа.

Глава 2. Алгоритмы, модели и данные

В данной главе описан алгоритм парсера зависимостей. Сначала приводится описание системы переходов arc-standard, на основе которой работает парсер. Далее описываются нейронная сеть, используемая для классификации переходов, алгоритм генерации обучающей выборки, и представление признаков, используемых в нейронной сети.

* 1. Система переходов arc-standard

Система переходов – концепция, используемая при синтаксическом анализе, которая из предложения за конечное количество переходов строит его дерево разбора. Система состоит из множества конфигураций и множества переходов (каждый переход – это функция из одной конфигурации в другую). Конфигурация предложения состоит из стека, буфера и дерева разбора этого предложения. В начальной конфигурации предложения дерево разбора пустое, в стеке хранится корневой элемент, в буфере хранятся члены предложения. В конечной конфигурации дерево разбора полностью построено, а буфер пуст. При анализе предложения, к его начальной конфигурации применяется последовательность переходов и получается конечная конфигурация предложения. При каждом переходе либо в дерево разбора добавляется новая зависимость, либо один член предложения сдвигается из буфера в стек [24].

Реализуемый парсер основан на системе переходов arc-standard, введенной Йоакимом Нивре [22]. Первоначальная конфигурация включает стек с одним элементом ROOT (корень дерева разбора), буфер, содержащий члены предложения, и пустое дерево зависимостей (дерево разбора). Для перехода к следующей конфигурации есть три возможных действия:

* *RIGHT-ARC*: при условии, что стек содержит по крайней мере два элемента, добавляет зависимость s2 → s1 к дереву разбора и удаляет s2 из стека.
* *LEFT-ARC*: при условии, что стек содержит по меньшей мере два элемента, добавляет зависимость s1 → s2 к дереву разбора и удаляет s1 из стека.
* *SHIFT*: при условии, что буфер не пуст, сдвигает верхнее слово буфера в стек.

Выше s1 и s2 являются соответственно первым и вторым элементами на вершине стека. На каждом шаге используется классификатор для определения следующего перехода, который затем применяется, и конфигурация обновляется. Разбор предложения

завершается, когда буфер и стек пусты. Полученное дерево разбора, в котором хранятся зависимости между членами предложения, возвращается как результат парсера.

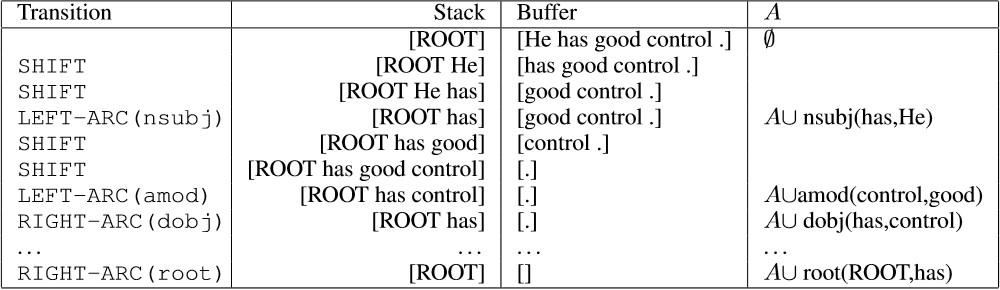
На рис. 5 отображена последовательность переходов, выполняемая во время анализа предложения «He has good control.».

Рисунок 5. Использование системы переходов для анализа предложения «He has

good control.» [1]

В деревьях разбора, получаемых через систему arc-standard, подчиненные каждого слова предложения по отношению к родительской вершине этого слова находятся на той же стороне в предложении, что и это слово. Иными словами, если использовать представление зависимостей, иллюстрированное на рис. 3, в получаемых разборах связи между членами предложения не будут пересекаться. Такие деревья называются проективными.

Парсеры, использующие систему переходов arc-standard, строят только проективные разборы. Однако в естественных языках деревья разбора не всех предложений являются проективными, и зависимости, нарушающие проективность дерева, определяются неправильно такими парсерами. На рис. 6 приведен пример предложения с непроективным разбором, а на рис. 7 – неправильный проективный разбор этого же предложения, построенный парсером.

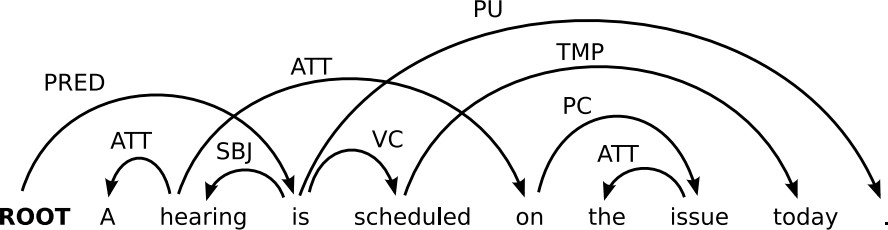


Рисунок 6. Правильный непроективный разбор предложения «A hearing is scheduled

on the issue today.» [24]

Но так как в естественных языках в большинстве предложений и зависимостей проективность не нарушается (в корпусе русских текстов СинТагРус [25] деревья разбора

почти 93% всех предложений являются проективными), парсеры, основанные на системе переходов arc-standard все равно показывают высокую точность.

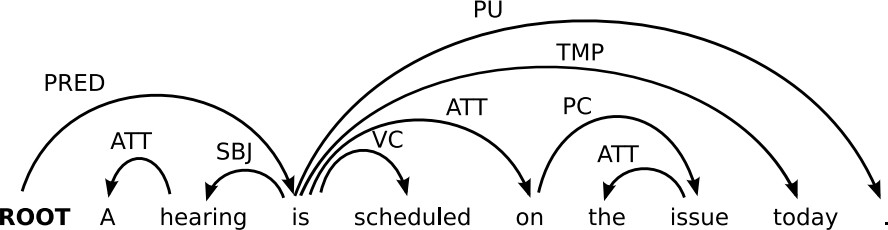


Рисунок 7. Неправильный проективный разбор предложения «A hearing is scheduled

on the issue today.» [24]

Для определения оптимальной последовательности переходов используется жадный алгоритм. Чтобы найти оптимальную последовательность переходов, на каждом шаге для текущей конфигурации из всех возможных переходов выбирается оптимальный. Чтобы определить оптимальный переход, используется классификатор. В общих чертах, алгоритм парсера приведен в схеме 1.

1. с = initial\_configuration(s) # получение начальной конфигурации
2. while not c.is\_terminal(): # пока конфигурация не конечная
3. t = classifier.get\_transition(c) # определение оптимального перехода
4. c = apply\_transition(c, t) # применение перехода
5. return c.tree. # дерево разбора построено

Схема 1. Алгоритм анализа предложения

* 1. Использование нейронной сети для классификации

Для классификации переходов с целью нахождения оптимального этот парсер использует искусственную нейронную сеть. Используемая сеть полносвязная и состоит из трех слоев: входной, скрытый и выходной. Входной слой принимает вектор признаков. Затем вычисляется функции активации нейронов скрытого слоя. В последнем слое к значениям скрытого слоя применяется функция softmax.

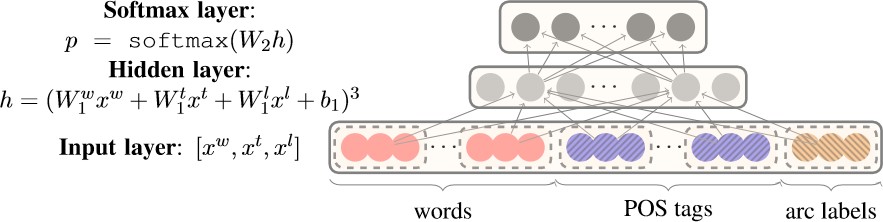


Рисунок 8. Структура нейронной сети [1]

Сеть классифицирует переходы конфигурации на основе признаков, извлеченных из текущей конфигурации. Из текущей конфигурации получаются следующие элементы:

* верхние 3 слова стека и буфера;
* первый и второй крайние левые и правые потомки верхних двух слов стека;
* крайний левый крайнего левого и крайний правый крайнего правого потомки верхних двух слов стека.

Выделенное множество из 18 слов (обозначим через Sw), части речи этих слов

(обозначим через St), а также метки зависимостей от их вершин (обозначим через Sl), за исключением меток 6-и слов из стека и буфера, составляют набор признаков, используемый при классификации.

Во время классификации вектор признаков, составленный из Sw, St, Sl, подается на

вход нейронной сети. Обозначив плотные представления признаков из выбранных элементов каждого типа (слова, части речи, метки зависимостей) через xw, xt, xl соответственно, можно выписать формулу вычисления скрытого слоя h:

ℎ = (𝑊𝑤𝑥𝑤 + 𝑊𝑡𝑥𝑡 + 𝑊𝑙𝑥𝑙 + 𝑏1)3

1 1 1

Где 𝑊𝑤, 𝑊𝑡, 𝑊𝑙 – веса связей между входным и скрытым слоем, а 𝑏1 – смещение.

1 1 1

На выходном слое сети применяется функция softmax, которая вычисляет вероятности классов:

𝑝 = softmax (𝑊2ℎ)

Где 𝑊2 – веса связей между скрытым и выходным слоем. Классификация выполняется среди всех возможных переходов, но во время анализа выбор оптимального перехода выполняется только из тех переходов, которые могут быть применены к конфигурации. Из применимых выбирается переход с самым большим значением в 𝑝.

* 1. Генерация обучающей выборки

Для обучения нейронной сети необходим синтаксически размеченный корпус. Обучающая выборка генерируется из проективных деревьев разбора этого корпуса, а непроективные деревья игнорируются.

Для генерации обучающих примеров для каждого дерева разбора восстанавливается последовательность переходов, через которою из начальной конфигурации предложения получается конечная. Чтобы восстановить эту последовательность, сначала получается начальная конфигурация предложения, затем с помощью дерева разбора определяется следующий переход. Из конфигурации извлекаются признаки, используемые классификатором, а также создается массив меток для всех возможных переходов. В этом

массиве для следующего перехода ставится метка 1, для других применимых к конфигурации переходов – метка 0, а для всех остальных – -1. После этого, извлеченные признаки и массив меток добавляются в обучающую выборку, и осуществляется переход к следующей конфигурации. Вышеописанные действия повторяются до достижения конечной конфигурации. В общих чертах, алгоритм генерации обучающих примеров из дерева разбора tree предложения s приведен в схеме 2.

1. if tree.is\_projective(): # если дерево разбора проективное
2. c = initial\_configuration(s) # получение начальной конфигурации
3. while not c.is\_terminal(): # пока конфигурация не конечная
4. t = get\_correct\_transition(c, tree)# получение правильного перехода
5. f = get\_features(c) # извлечение признаков
6. y = [] # метки переходов
7. for i in range(len(transitions)): # transitions хранит все переходы
8. if transitions[i] = t:
9. y[i] = 1 # 1 для правильного перехода
10. elif can\_apply(c, transitions[i]):
11. y[i] = 0 # 0 для применимых переходов
12. else:
13. y[i] = -1 # -1 для неприменимых переходов
14. train\_dataset.add(f, y) # добавление примера в обучающую выборку
15. c = apply\_transition(c, t) # обновление конфигурации

Схема 2. Алгоритм генерации обучающих примеров

* 1. Представление признаков и функция активации

В парсере, каждое слово представляется как d-мерный вектор. Аналогично, части речи и метки зависимостей тоже отображаются в d-мерное векторное пространство. У параметра d есть значение по умолчанию, но пользователь тоже может задавать значение этого параметра, а также предоставить файл с d-мерными представлениями слов, частей речи и меток зависимостей. Если такой файл отсутствует, то эти представления получаются во время обучения нейронной сети с помощью обратного распространения ошибки. Как показывают эксперименты [1], такой подход к представлению признаков быстрее унитарного кодирования, а также повышает точность парсера, так как лучше отражает семантическую близость признаков. Плотные представления для меток частей речи и зависимостей впервые были использованы в этом алгоритме анализа.

Нововведением также является использование кубической функции активации в нейронах скрытого слоя нейронной сети. В ранних работах как правило применялись гиперболический тангенс или сигмоид. Использование кубической активации помогает повысить точность парсера [1]. Это улучшение объясняется тем, что с этой функцией

взаимодействия входных признаков отражаются лучше. Для произвольного нейрона скрытого слоя, считается активация g(x) = x3, где x – взвешенная сумма элементов входного слоя, g – кубическая функция активации. Распишем вычисления подробно:

𝑔(𝑥) = 𝑔(𝑤1𝑥1 … + 𝑤𝑖𝑥𝑖 + ⋯ +𝑤𝑗𝑥𝑗 … +𝑤𝑘𝑥𝑘 + ⋯ + 𝑏) =

3

= (𝑤1𝑥1 … + 𝑤𝑖𝑥𝑖 + ⋯ +𝑤𝑗𝑥𝑗 … +𝑤𝑘𝑥𝑘 + ⋯ + 𝑏) = ⋯ + (𝑤𝑖𝑤𝑗𝑤𝑘)𝑥𝑖𝑥𝑗𝑥𝑘 + ⋯

С кубической функцией активации отражаются произведения трех произвольных входных признаков, что возможно и вызывает улучшение в точности парсера.

* 1. Частеречная разметка

Для выполнения синтаксического анализа необходимо сначала выполнить частеречную разметку текста, так как использование частей речи членов предложения как признаков значительно улучшает точность анализа (табл. 7). В реализуемом парсере для частеречной разметки используется усредненный перцептрон, основной алгоритм разметки библиотеки NLTK. Этот алгоритм разметки показывает 96.8% accuracy на англоязычном корпусе WSJ [26] и 99% на Национальном корпусе русского языка.

Выводы по главе

В данной главе был описан алгоритм реализуемого парсера на основе деревьев зависимостей. Было приведено описание системы переходов arc-standard, используемой парсером. Также были рассмотрены особенности нейронной сети, используемой для классификации переходов, а именно: векторное представление признаков и кубическая функция активации.

Глава 3. Программная реализация

В данной главе рассматривается программная реализация алгоритма синтаксического анализа текстов на основе деревьев зависимостей. Алгоритм реализован на языке программирования Python 3. Для программирования была использована среда разработки PyCharm и методы библиотеки NumPy. Сначала описана структура программы. Далее представлены результаты проверки качества анализа на корпусе СинТагРус.

* 1. Структура проекта программы

Структуру программной реализации можно разделить на три основные части:

* 1. Парсер зависимостей
  2. Система переходов arc-standard
  3. Классификатор
     1. Парсер зависимостей

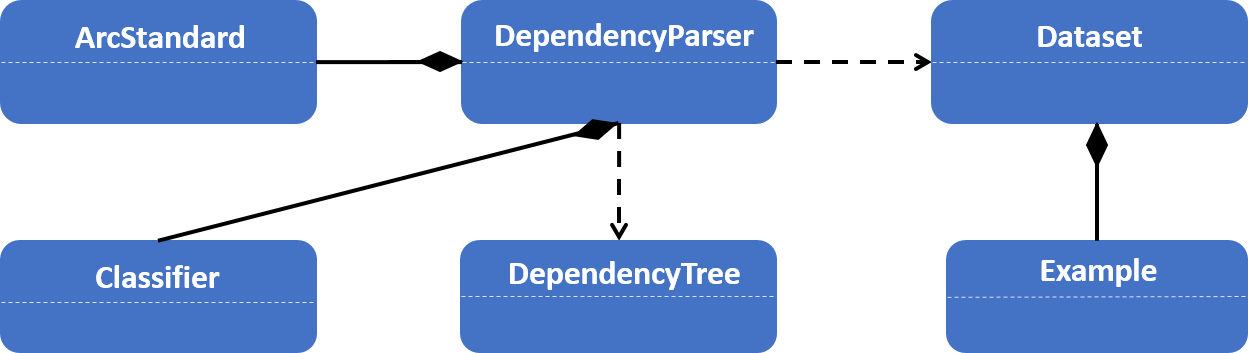


Рисунок 9. Диаграмма классов парсера

Выполняет синтаксический анализа текста, принимая на вход предложение с указанными частями речи своих членов и возвращая дерево зависимостей, хранящий зависимости между членами предложения. Позволяет пользователю обучать модели парсера для разных языков. Для обучения модели парсера нужен размеченный корпус текстов в формате CoNLL-U [27], содержащий деревья зависимостей. Позволяет пользователю сохранять обученную модель, а также использовать ее для выполнения синтаксического анализа. Основные классы приведены на рис. 9.

В таблицах 2, 3 и 4 приведены назначения классов и их основных методов.

Таблица 2.

Назначение классов парсера

|  |  |
| --- | --- |
| Класс | Назначение |
| DependencyParser | Содержит основные методы парсера – обучения, сохранения модели, анализа предложения, текстового файла, текстового  файла в формате CoNLL-U. |
| DependencyTree | Реализация дерева синтаксических зависимостей. |
| Dataset | Класс для хранения обучающей или тестовой выборки.  Примеры хранятся в списке экземпляров класса Example. |
| Example | Класс для хранения обучающего или тестового примера.  Хранит признаки и метки классов. |

Таблица 2. Назначение классов в парсере

Таблица 3.

Описание основных методов класса DependencyParser

|  |  |
| --- | --- |
| Метод | Назначение |
| parse | Принимает предложение с указанными частями речи и возвращает дерево разбор этого предложения – экземпляр  класса DependencyTree. |
| train | Принимает файл c размеченными предложениями в формате CoNLL-U и обучает модель парсера. Если в парсере предварительно не был указан тэггер, обучает усредненный перцептрон для выполнения частеречной разметки  предложений. |
| test\_conll | Принимает файл c размеченными предложениями в формате CoNLL-U и определяет точность парсера (UAS, LAS) на этих предложениях, сохраняет построенные для этих предложений деревья разбора в другом файле в формате CoNLL-U. Метод работает, только если парсер уже обучен или была загружена  модель парсера. |
| write\_model\_file | Записывает модель парсера в файл. |
| load\_model\_file | Загружает модель парсера и обновляет/задает параметры  парсера и классификатора. |

Таблица 4.

Описание основных методов класса DependencyTree

|  |  |
| --- | --- |
| Метод | Назначение |
| add | Добавляет новую зависимость в дерево разбора. |
| get\_head | Возвращает номер родителя переданного члена предложения. |
| get\_label | Возвращает тип зависимости от родителя для переданного члена  предложения. |
| get\_root | Возвращает номер корня дерева разбора. |
| is\_single\_root | Проверяет, что у предложения только один корень в дереве разбора. |
| is\_tree | Проверяет, является ли разбор предложения деревом. |
| is\_projective | Проверяет, является ли дерево разбора проективным. |

Поле head класса DependencyTree хранит список родителей членов предложения в дереве разбора, поле label – список типов соответствующих зависимостей.

* + 1. Система переходов arc-standard



Рисунок 10. Диаграмма классов реализации системы arc-standard

Таблица 5.

Назначение классов в реализации системы переходов arc-standard

|  |  |
| --- | --- |
| Класс | Назначение |
| ParsingSystem | Абстрактный класс, описывающий основные методы,  необходимые для реализации системы переходов. |
| ArcStandard | Класс, реализующий систему переходов arc-standard. Наследует  и реализует методы класса ParsingSystem. |
| Configuration | Класс, реализующий конфигурацию системы переходов. Хранит  стек, буфер и дерево разбора. |

Реализует систему переходов arc-standard, на основе которой работает парсер. Предоставляет методы для создания начальной конфигурации из предложения, нахождения возможных переходов, выполнения перехода из одной конфигурации в

другую. Также содержит метод, который принимает на вход конфигурацию и дерево разбора анализируемого предложения и определяет следующий переход. Последний метод используется во время обучения парсера для генерации тренировочных экземпляров. Реализация состоит из классов ParsingSystem, ArcStandard, Configuration (рис. 10). Назначения этих классов приведены в таблице 5.

* + 1. Классификатор

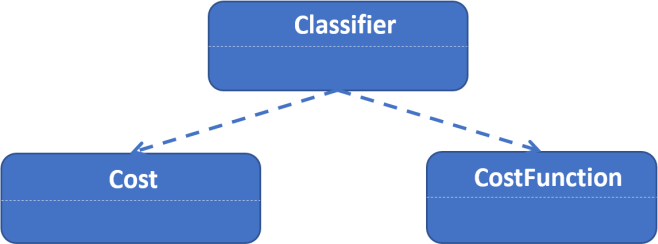


Рисунок 11. Диаграмма классов классификатора

В классификаторе реализована полносвязная нейронная сеть из трех слоев для классификации переходов. Предоставляет методы для обучения нейронной сети и вычисления вероятностей переходов. В качестве функции активации в нейронах скрытого слоя используется кубическая функция. На выходном слое применяется функция softmax. Реализация состоит из трех классов – Classifier, CostFunction, Cost (рис. 11).

В реализации нейронной сети матричные вычисления векторизованы, что позволяет значительно ускорить обучение классификатора и предсказание переходов. Для векторизации этих операций была использована библиотека NumPy [28].

* 1. Интеграция парсера в NLTK

Парсеры в библиотеке NLTK хранятся в модуле parse, при этом исходный код каждого парсера хранится в одном отдельном файле. Чтобы не нарушить эту практику, все классы и вспомогательные методы реализуемого парсера также были собраны в один файл под названием stanfordnndep (stanford – от имени университета, где был разработан алгоритм парсера; nndep – указывает что это парсер зависимостей на основе нейронной сети).

Так как в NLTK большинство инструментов на основе машинного обучения также предоставляют готовые модели, и чтобы сделать возможным использование парсера без

обучения, в репозиторий NLTK Data [29], где хранятся модели и корпусы библиотеки NLTK, была добавлена модель парсера для русского языка.

* 1. Обучение парсера

Для создания парсера нужно сначала обучить его на корпусе текстов и получить модель парсера, где будут сохранены необходимые для работы парсера параметры, полученные из корпуса. В модели хранятся веса связей нейронной сети, векторные представления слов, меток частей речи и зависимостей. Когда модель для языка получена, используя ее парсер может уже анализировать предложения на этом языке.

Чтобы обучить парсер для какого-то языка, нужен размеченный корпус текстов на этом языке. В корпусе для всех предложений должны присутствовать их деревья разбора, а также для всех членов предложения должны быть указаны части речи. Все тексты должны быть собраны в одном файле формата CoNLL-U (схема 3). В одной строке файла формата CoNLL-U хранятся атрибуты одного члена предложения:

1. порядковый номер в предложении;
2. токен (член предложения);
3. обработанный токен;
4. часть речи в формате Universal Dependencies [30];
5. часть речи из корпуса;
6. морфологические признаки;
7. порядковый номер родителя в дереве разбора;
8. тип синтаксической зависимости от родителя.

Атрибуты члена предложения разделяются одним символом табуляции. Предложения разделяются пустой строкой. В парсере 5-ый (метка части речи из корпуса) и 6-ой атрибуты (морфологические признаки) не используются.

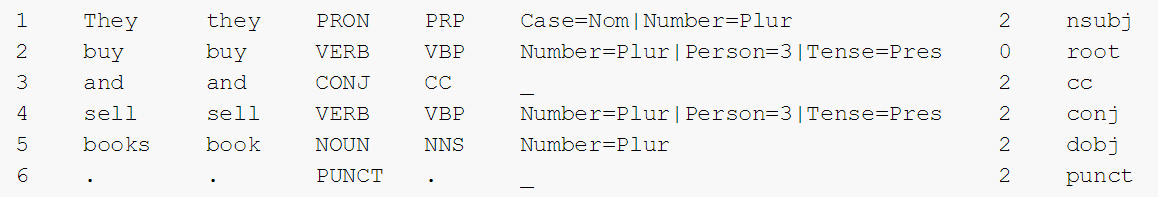


Схема 3. Размеченное предложение «They buy and sell books.» в формате CoNLL-U [27]

Для обучения может также понадобиться файл с векторными представлениями слов этого языка. В каждой строке этого файла должно находиться представление одного

слова. Слово должно быть в начале строки, а далее в этой же строке его векторное представление (между элементами вектора ставится пробел). Так как эти представления обучаются нейронной сетью парсера, то этот файл не необходим, но его наличие может значительно ускорить процесс обучения.

Модель парсера можно сохранить в файле для использования в будущем. В начале файла, где хранится модель парсера, записываются параметры парсера (количество слов, частей речи, синтаксических зависимостей из обучающей выборки, размер вектора представления слов, количество нейронов в скрытом слое нейронной сети и используемых признаков), далее обученные векторные представления слов, меток частей речи и синтаксических зависимостей, далее веса нейронной сети.

* 1. Определение качества парсера

В статье [1], где был представлен реализуемый парсер, авторы также опубликовали результаты своих экспериментов по определению точности и скорости парсера для анализа текстов на английском языке. В своих экспериментах авторы использовали корпус The Penn Treebank. Для определения точности использовались меры UAS и LAS, для определения скорости – количество обработанных слов и предложений в секунду. Полученные результаты были сопоставлены с другими популярными парсерами с открытым кодом (табл. 6): Malt:sp (arc-standard), Malt:eager (arc-eager), MSTParser.

Таблица 6.

Точность и скорость парсеров Malt, MST и реализуемого парсера на англоязычных текстах [1]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Парсер | Dev |  | Test |  | Скорость |
|  | UAS | LAS | UAS | LAS | (предложений/с) |
| Malt:sp | 90.0 | 88.8 | 89.9 | 88.5 | 560 |
| Malt:eager | 90.1 | 88.9 | 90.1 | 88.7 | 535 |
| MSTParser | 92.1 | 90.8 | 92.0 | 90.5 | 12 |
| **Реализуемый парсер** | **92.2** | **91.0** | **92.0** | **90.7** | **1013** |

MaltParser работает на основе системы переходов. В экспериментах использовался версия парсера с методом машинного обучения LIBLINEAR для определения переходов из одной конфигурации в другую.

Разбор предложения в парсере MST выполняется на основе взвешенных графов. Члены предложения рассматриваются как вершины взвешенного графа, где проведены все допустимые связи, а построение разбора предложения сводится к построению минимального остовного дерева в этом графе. Нетрудно заметить, что парсер на основе нейронной сети не уступает в точности и сильно превосходит в скорости парсеры Malt и MST.

Точность парсера была установлена также для русского языка. В экспериментах использовался корпус СинТагРус, составленный из научно-популярных, общественно- политических статей, а также русской художественной прозы конца 20-го века. Использованная версии корпуса содержит:

- 1032644 слов;

* 59130 предложений;
  + 54604 предложений с проективным разбором (~ 92.3%);
  + 4526 предложений с непроективным разбором (~ 7.7%);
* 15 меток частей речи [30]:
* ADJ: adjective (КОТОРЫЙ, ОДИН, ДРУГОЙ, НОВЫЙ, САМЫЙ, ПЕРВЫЙ)
* ADP: adposition (В, НА, С, ПО, К, ИЗ, О)
* ADV: adverb (ТАК, УЖЕ, ЕЩЕ, МОЖНО, БОЛЕЕ, КАК, ОЧЕНЬ, ОДНАКО)
* AUX: auxiliary verb (БЫТЬ)
* CCONJ: coordinating conjunction (И, А, НО, ИЛИ, ТО, ЧЕМ, НИ)
* DET: determiner (ЭТОТ, СВОЙ, ВЕСЬ, ТАКОЙ, НАШ)
* INTJ: interjection (АХ, О, ОХ, А, ТА-ДА-ДАМ, ЭХ)
* NOUN: noun (ЭТО, ГОД, ТО, ЧЕЛОВЕК, РОССИЯ, ВРЕМЯ)
* NUM: numeral (ДВА, НЕСКОЛЬКО, ТРИ, 1, 10, 20)
* PART: particle (НЕ, И, ЖЕ, ТОЛЬКО, БЫ, ДАЖЕ, ВОТ)
* PRON: pronoun (ОН, ОНИ, Я, МЫ, ОНА, ЧТО, СЕБЯ)
* PUNCT: punctuation (,, ., ", -, :, ?, ), (, !, …)
* SCONJ: subordinating conjunction (ЧТО, КАК, ЕСЛИ, ЧТОБЫ, КОГДА)
* SYM: symbol (%, $, #)
* VERB: verb (МОЧЬ, БЫТЬ, СТАНОВИТЬСЯ, ГОВОРИТЬ, ДЕЛАТЬ)
* 34 синтаксических зависимостей – acl, acl:relcl, advcl, advmod, amod, appos, aux, auxpass, case, cc, ccomp, compound, conj, cop, dep, det, dobj, expl, iobj, mark, mwe, name, neg, nmod, nmod:agent, nsubj, nsubjpass, nummod, nummod:entity, nummod:gov, parataxis, punct, root, xcomp [31].

В исходной версии корпуса, размеченные тексты хранятся в формате XML (Схема 4). Так как парсер обучается и анализирует тексты в формате CoNLL-U, в экспериментах была использована конвертированная версия корпуса (Схема 5).

<S ID="5">

<W DOM="2" FEAT="S ЕД ЖЕН ИМ НЕОД" ID="1" LEMMA="ПРИЕМНАЯ" LINK="предик">**Приемная**</W>

<W DOM="\_root" FEAT="V НЕСОВ ИЗЪЯВ ПРОШ ЕД ЖЕН" ID="2" LEMMA="БЫТЬ">**была**</W>

<W DOM="2" FEAT="V СОВ СТРАД ПРИЧ ПРОШ КР ЕД ЖЕН" ID="3" LEMMA="ОБСТАВЛЯТЬ"

LINK="пасс-анал">**обставлена**</W>

<W DOM="3" FEAT="ADV" ID="4" LEMMA="ПРОСТО" LINK="обст">**просто**</W>,

<W DOM="4" FEAT="CONJ" ID="5" LEMMA="НО" LINK="сочин">**но**</W>

<W DOM="5" FEAT="ADV" ID="6" LEMMA="ПО-ДЕЛОВОМУ" LINK="соч-союзн">**по-деловому**</W>.

</S>

Схема 4. Пример предложения из СинТагРус в формате XML («Приемная была

обставлена просто, но по-деловому.»).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | Приемная | приемная | NOUN | \_ | \_ | 3 nsubj | 3:nsubj \_ |
| 2 | была быть | AUX \_ | \_ | 3 | aux:pass | 3:aux:pass | \_ |
| 3 | обставлена | обставлять | VERB | \_ | \_ | 0 root | 0:root \_ |
| 4 | просто просто | ADV \_ | \_ | 3 | advmod | 3:advmod | SpaceAfter=No |
| 5 | , , | PUNCT \_ | \_ | 4 | punct | 4:punct \_ |  |
| 6 | но но | CCONJ \_ | \_ | 7 | cc | 7:cc \_ |  |
| 7 | по-деловому | по-деловому | ADV | \_ | \_ | 4 conj | 4:conj SpaceAfter=No |
| 8 | . . | PUNCT \_ | \_ | 7 | punct | 7:punct \_ |  |

Схема 5. Пример предложения в формате CoNLL-U из конвертированного СинТагРус («Приемная была обставлена просто, но по-деловому.»).

Точность парсера определяется с помощью метрик UAS и LAS. В метрике UAS считается процент правильно предсказанных зависимостей в тестовой выборке (тип зависимости не учитывается). Предсказанная зависимость ℎ𝑒𝑎𝑑𝑝𝑟𝑒𝑑𝑖𝑐𝑡𝑒𝑑 →

𝑑𝑒𝑝𝑒𝑛𝑑𝑒𝑛𝑡 считается правильной, если вершина ℎ𝑒𝑎𝑑𝑝𝑟𝑒𝑑𝑖𝑐𝑡𝑒𝑑 совпадает с вершиной

ℎ𝑒𝑎𝑑𝑡𝑒𝑠𝑡 в зависимости ℎ𝑒𝑎𝑑𝑡𝑒𝑠𝑡 → 𝑑𝑒𝑝𝑒𝑛𝑑𝑒𝑛𝑡 из тестовой выборки.

В метрике LAS также считается процент правильно построенных зависимостей, но с учетом типа зависимости (если тип определен неправильно, то зависимость тоже считается неправильной). Предсказанная зависимость ℎ𝑒𝑎𝑑𝑝𝑟𝑒𝑑𝑖𝑐𝑡𝑒𝑑

𝑡𝑦𝑝𝑒𝑝𝑟𝑒𝑑𝑖𝑐𝑡𝑒𝑑

→ 𝑑𝑒𝑝𝑒𝑛𝑑𝑒𝑛𝑡 считается правильной, если ℎ𝑒𝑎𝑑𝑝𝑟𝑒𝑑𝑖𝑐𝑡𝑒𝑑 и 𝑡𝑦𝑝𝑒𝑝𝑟𝑒𝑑𝑖𝑐𝑡𝑒𝑑 совпадают

с ℎ𝑒𝑎𝑑

𝑡𝑒𝑠𝑡

и 𝑡𝑦𝑝𝑒𝑡𝑒𝑠𝑡

в зависимости ℎ𝑒𝑎𝑑

𝑡𝑦𝑝𝑒𝑡𝑒𝑠𝑡

𝑡𝑒𝑠𝑡 → 𝑑𝑒𝑝𝑒𝑛𝑑𝑒𝑛𝑡

из тестовой выборки.

𝑈𝐴𝑆 =

число слов с правильно предсказанным ℎ𝑒𝑎𝑑𝑝𝑟𝑒𝑑𝑖𝑐𝑡𝑒𝑑 число слов

∗ 100%

𝐿𝐴𝑆 =

число слов с правильно предсказанными (ℎ𝑒𝑎𝑑𝑝𝑟𝑒𝑑𝑖𝑐𝑡𝑒𝑑, 𝑡𝑦𝑝𝑒𝑝𝑟𝑒𝑑𝑖𝑐𝑡𝑒𝑑) число слов

∗ 100%

Для определения точности парсера использовался метод кросс-валидации [32]. Корпус был разбит на 8 частей. Парсер сначала обучался на всех частях корпуса кроме одной, а потом вычислялась его точность (UAS и LAS) на неиспользованной во время обучения части корпуса. В предложениях, передаваемых в парсер, части речи были предварительно указаны.

Было проведено два эксперимента. В первом эксперименте части речи не использовались как признаки в классификаторе, во втором – использовались. Результаты эксперимента приведены в таблице 7.

Таблица 7.

Точность парсера на русскоязычных текстах

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Набор признаков | Dev |  | Test |  | Train | |
| UAS | LAS | UAS | LAS | UAS | LAS |
| С частями речи | 84.58 | 78.61 | 84.7 | 78.71 | 93.79 | 91.27 |
| Без частей речи | 75.69 | 68.36 | - | - | - | - |

Эксперименты проводились на компьютере с процессором Intel Core i7 2.67GHz, оперативной памятью 8 GB. Количество тренировочных итераций - 20 000. В качестве векторного представления слов была использована модель word2vec из 374 526 слов с размером вектора - 300 (Ruscorpora RUSSE model, [39]).

Заключение

Синтаксический анализ широко применяется при решении многих задач в области обработки естественного языка. Алгоритмы анализа зависимостей являются важным компонентом в системах машинного перевода, генерации текста и извлечения семантических отношений. Учитывая тот факт, что в последние годы разработчики стали более активно применять алгоритмы машинного обучения в своих проектах, очень важно иметь реализацию алгоритма синтаксического анализа с открытым доступом, которая будет выполнять образовательную функцию, а также использоваться в исследованиях.

NLTK - это широко используемая платформа, которая предоставляет инструменты для обработки текстов, включая синтаксический анализатор (парсер). Парсеры NLTK либо основаны на слишком простых моделях (парсеры, основанные на правилах) и не могут быть использованы в исследованиях, либо реализованы не на языке Python (интерфейсы к MaltParser и стэнфордскому парсеру) и не могут быть использованы в образовательных целях. К тому же, среди своих прямых аналогов, NLTK является единственной библиотекой, не предоставляющей современный парсер зависимостей. Стремясь устранить отсутствие современного парсера в NLTK, данная работа реализует алгоритм анализа зависимостей, разработанный исследователями Стэнфордского университета, и включает его в NLTK.

В реализуемом алгоритме представлены несколько инновационных идей, а именно: применение плотных представлений частей речи и меток зависимостей, а также кубическая функция активации в нейронной сети. По сравнению с ранними решениями, Парсер демонстрирует значительное улучшение в скорости обработки предложений.

Представленный проект является значительным шагом вперед для NLTK и способствует распространению современных подходов к анализу текстов. Учитывая быстрые темпы развития в области искусственного интеллекта, включение этого алгоритма в NLTK имеет большое значение, поскольку будет служить руководством для новичков и облегчит изучение обработки естественного языка.

Кроме того, была обучена модель парсера для анализа русскоязычных текстов, показывающая точность 84.7% UAS на корпусе СинТагРус. Чтобы предоставить открытый доступ к модели, она была помещена в репозиторий NLTK DATA. Используя эту модель, разработчики могут анализировать предложения на русском языке без необходимости обучения парсера.

Возможно несколько направлений дальнейшей работы. Текущий парсер обучается только на проективных деревьях, и получаемые деревья тоже всегда проективные. Замена

непроективных деревьев разбора псевдопроективными во время обучения парсера, потенциально может улучшить качество анализа непроективных предложений (синтаксические зависимости, нарушающие проективность, все равно будут определяться неправильно, но будет минимизирована ошибка на других зависимостях предложения).

Другой модификацией, которая может привести к улучшению качества парсера, является использование лучевого поиска для определения оптимальной последовательности переходов. Однако это замедлит работу парсера и потребует больше памяти. В этой работе, во время обучения в парсер передаются предложения с указанными частями речи. Учитывая высокую точность алгоритмов частеречной разметки для русского языка, можно изменить алгоритм обучения таким образом, чтобы части речи членов предложения определялись парсером в процессе обучения. Наличие этого парсера также делает возможным разработку других инструментов обработки естественного языка (например, алгоритма извлечения именованных сущностей).

Суммируя вышесказанное, цель работы была достигнута – на языке программирования Python был реализован парсер и интегрирован в библиотеку NLTK. Были выполнены следующие задачи:

1. Изучены алгоритмы синтаксического анализа на основе систем переходов.
2. Разработан парсер на основе деревьев зависимостей и нейронной сети.
3. Парсер интегрирован в библиотеку NLTK.
4. Определена точность парсера на корпусе СинТагРус.
5. Модель парсера для русского языка добавлена в репозиторий NLTK Data.

8. Разработаны пояснительная записка и техническая документация.

Список использованных источников

1. A Fast and Accurate Dependency Parser using Neural Networks / Danqi Chen, Christopher
   1. Manning // [Электронный ресурс]: Stanford University. – 2014 – Режим доступа: <http://cs.stanford.edu/people/danqi/papers/emnlp2014.pdf>, свободный. (дата обращения: 24.05.2017).
2. Quadratic-Time Dependency Parsing for Machine Translation / Michel Galley, Christopher
   1. Manning // [Электронный ресурс]: Stanford University. – 2009 – Режим доступа: <https://nlp.stanford.edu/pubs/acl09-depLM.pdf>свободный. (дата обращения: 24.05.2017).
3. Jing Jiang. INFORMATION EXTRACTION FROM TEXT // [Электронный ресурс] Режим доступа: <https://pdfs.semanticscholar.org/0f36/4400ce5eefc1298260ed1f01e201ef323d7e.pdf>, свободный. (дата обращения: 24.04.2017).
4. Michael A. Covington. 2001. A Fundamental Algorithm for Dependency Parsing // [Электронный ресурс] Режим доступа: [https://web.stanford.edu/~mjkay/covington.pdf](https://web.stanford.edu/%7Emjkay/covington.pdf), свободный. (дата обращения: 24.04.2017).
5. Грамматика зависимостей [Электронный ресурс] // Википедия. Режим доступа: [https://ru.wikipedia.org/wiki/Грамматика\_зависимостей](https://ru.wikipedia.org/wiki/%C3%90%E2%80%9C%C3%91%E2%82%AC%C3%90%C2%B0%C3%90%C2%BC%C3%90%C2%BC%C3%90%C2%B0%C3%91%E2%80%9A%C3%90%C2%B8%C3%90%C2%BA%C3%90%C2%B0_%C3%90%C2%B7%C3%90%C2%B0%C3%90%C2%B2%C3%90%C2%B8%C3%91%C2%81%C3%90%C2%B8%C3%90%C2%BC%C3%90%C2%BE%C3%91%C2%81%C3%91%E2%80%9A%C3%90%C2%B5%C3%90%C2%B9), свободный. (дата обращения: 24.04.2017).
6. Stanford Dependencies [Электронный ресурс] // The Stanford Natural Language Processing Group. Режим доступа: [https://nlp.stanford.edu/software/stanford- dependencies.shtml#Methods](https://nlp.stanford.edu/software/stanford-dependencies.shtml#Methods), свободный. (дата обращения: 24.05.2017).
7. Dependency Parsing / Joakim Nivre, Sandra Kubler // [Электронный ресурс] Режим доступа: [https://stp.lingfil.uu.se/~nivre/docs/ACLslides.pdf](https://stp.lingfil.uu.se/%7Enivre/docs/ACLslides.pdf), свободный. (дата обращения: 24.05.2017).
8. Parsing the SYNTAGRUS Treebank of Russian / Joakim Nivre, Igor M. Boguslavsky, Leonid L. Iomdin // [Электронный ресурс] Режим доступа: <http://www.aclweb.org/anthology/C08-1081.pdf>, свободный. (дата обращения: 24.05.2017).
9. A computational model of human parsing / Abney, Steven P. // Journal of Psycholinguistic Research 18 – 1989 – с.129–144.
10. Why is dependency parsing so much faster than constituency parsing? [Электронный ресурс] // Linguistics StackExchange. Режим доступа: [https://linguistics.stackexchange.com/questions/6875/why-is-dependency-parsing-so-much- faster-than-constituency-parsing/6890#6890](https://linguistics.stackexchange.com/questions/6875/why-is-dependency-parsing-so-much-faster-than-constituency-parsing/6890#6890), свободный. (дата обращения: 24.05.2017).
11. Globally Normalized Transition-Based Neural Networks / Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, Michael Collins // [Электронный ресурс]: Google Inc, New York, NY. – 2016 – Режим доступа: <https://arxiv.org/pdf/1603.06042.pdf>, свободный. (дата обращения: 24.05.2017).
12. SpaCy [Электронный ресурс] // Wikipedia. Режим доступа: <https://en.wikipedia.org/wiki/SpaCy>, свободный. (дата обращения: 24.04.2017).
13. Transition-based dependency parsing using recursive neural networks / Pontus Stenetorp // NIPS Workshop on Deep Learning – 2013.
14. Discriminative training of a neural network statistical parser / James Henderson // Proc. ACL, Main Volume – 2004 – с. 95–102.
15. Fast and robust multilingual dependency parsing with a generative latent variable model / Ivan Titov, James Henderson // Proc. EMNLP – 2007 – с. 947–951.
16. Deep learning for efficient discriminative parsing / Ronan Collobert // AISTATS – 2011.
17. 5 Heroic Python NLP Libraries [Электронный ресурс] // EliteDataScience. Режим доступа: <https://elitedatascience.com/python-nlp-libraries>, свободный. (дата обращения: 24.05.2017).
18. Stanford CoreNLP – Core natural language software [Электронный ресурс] // Stanford CoreNLP. Режим доступа: <https://stanfordnlp.github.io/CoreNLP/>, свободный. (дата обращения: 24.04.2017).
19. SyntaxNet [Электронный ресурс] // Режим доступа: <https://github.com/tensorflow/models/tree/master/syntaxnet>, свободный. (дата обращения: 24.04.2017).
20. spaCy API [Электронный ресурс] // Режим доступа: <https://spacy.io/docs/api/>, свободный. (дата обращения: 24.04.2017).
21. NLTK GitHub repository [Электронный ресурс] // Режим доступа: <https://github.com/nltk/nltk/tree/develop/nltk/parse>, свободный. (дата обращения: 24.04.2017).
22. Joakim Nivre. Memory-Based Dependency Parsing / J. Nivre, J. Hall, J. Nilsson // Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL), 2004, Бостон, Массачусетс, с. 49-56.
23. Installing Third Party Software [Электронный ресурс] // NLTK Wiki. Режим доступа: <https://github.com/nltk/nltk/wiki/Installing-Third-Party-Software>, свободный. (дата обращения: 24.05.2017).
24. Joakim Nivre. Transition-based parsing / Joakim Nivre // [Электронный ресурс]: Uppsala University. Режим доступа: [http://stp.lingfil.uu.se/~nivre/master/transition.pdf](http://stp.lingfil.uu.se/%7Enivre/master/transition.pdf), свободный. (дата обращения: 24.05.2017).
25. СинТагРус [Электронный ресурс] // НКРЯ. Режим доступа: [http://www.ruscorpora.ru/instruction-syntax.html,](http://www.ruscorpora.ru/instruction-syntax.html) свободный. (дата обращения: 24.04.2017).
26. A Good Part-of-Speech Tagger in about 200 Lines of Python [Электронный ресурс] // Режим доступа: <https://explosion.ai/blog/part-of-speech-pos-tagger-in-python>, свободный. (дата обращения: 24.05.2017).
27. CoNLL-U Format [Электронный ресурс] // Universal Dependencies. Режим доступа: <http://universaldependencies.org/docs/format.html>, свободный. (дата обращения: 24.05.2017).
28. NumPy [Электронный ресурс] // NumPy. Режим доступа: [http://www.numpy.org/,](http://www.numpy.org/) свободный. (дата обращения: 24.05.2017).
29. NLTK Data [Электронный ресурс] // NLTK. Режим доступа: <https://github.com/nltk/nltk_data>, свободный. (дата обращения: 24.05.2017).
30. POS tags [Электронный ресурс] // Universal Dependencies. Режим доступа: [http://universaldependencies.org/ru/pos/index.html,](http://universaldependencies.org/ru/pos/index.html) свободный. (дата обращения: 24.05.2017).
31. Dependencies [Электронный ресурс] // Universal Dependencies. Режим доступа: [http://universaldependencies.org/ru/dep/index.html,](http://universaldependencies.org/ru/dep/index.html) свободный. (дата обращения: 24.05.2017).
32. Перекрестная проверка [Электронный ресурс] // Википедия. Режим доступа: https://ru.wikipedia.org/wiki/Перекрестная проверка, свободный. (дата обращения:

24.04.2017)

1. Neural Network Dependency Parser [Электронный ресурс] // The Stanford NLP Group. Режим доступа: <https://nlp.stanford.edu/software/nndep.shtml>, свободный. (дата обращения: 24.04.2017).
2. Синтаксический уровень [Электронный ресурс] // Лингвистика. Режим доступа: <http://linguistics-konspect.org/?content=2471>, свободный. (дата обращения: 24.04.2017).
3. Speech and Language Processing. Chapter 14 / Daniel Jurafsky, James H. Martin // [Электронный ресурс]: Stanford University. Режим доступа: [https://web.stanford.edu/~jurafsky/slp3/14.pdf](https://web.stanford.edu/%7Ejurafsky/slp3/14.pdf), свободный. (дата обращения: 24.04.2017).
4. Natural Language Toolkit [Электронный ресурс] // Wikipedia. Режим доступа: <https://en.wikipedia.org/wiki/Natural_Language_Toolkit>, свободный. (дата обращения: 24.04.2017).
5. Синтаксический анализ [Электронный ресурс] // Википедия. Режим доступа: [https://ru.wikipedia.org/wiki/Синтаксический\_анализ](https://ru.wikipedia.org/wiki/%C3%90%C2%A1%C3%90%C2%B8%C3%90%C2%BD%C3%91%E2%80%9A%C3%90%C2%B0%C3%90%C2%BA%C3%91%C2%81%C3%90%C2%B8%C3%91%E2%80%A1%C3%90%C2%B5%C3%91%C2%81%C3%90%C2%BA%C3%90%C2%B8%C3%90%C2%B9_%C3%90%C2%B0%C3%90%C2%BD%C3%90%C2%B0%C3%90%C2%BB%C3%90%C2%B8%C3%90%C2%B7), свободный. (дата обращения: 24.04.2017).
6. What are good alternatives to NLTK? [Электронный ресурс] // Quora. Режим доступа: [https://www.quora.com/What-are-good-alternatives-to-NLTK,](https://www.quora.com/What-are-good-alternatives-to-NLTK) свободный. (дата обращения: 24.04.2017).
7. Influence of training settings on word2vec CBOW and skip-gram language models for Russian [Электронный ресурс] // Режим доступа: <http://ling.go.mail.ru/data/misc/dialogue_2015.html/>, свободный. (дата обращения: 24.04.2017).

Приложения

Приложение A. Техническое задание

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук Департамент программной инженерии

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Доцент факультета компьютерных наук базовой кафедры «Системное программирование» НИУ ВШЭ,  канд. физ.-мат. наук  Д.Ю. Турдаков  « » 2017г. | УТВЕРЖДАЮ  Академический руководитель образовательной программы  «Программная инженерия»  В.В. Шилов  « » 2017г. |

СИНТАКСИЧЕСКИЙ АНАЛИЗАТОР НА ОСНОВЕ ДЕРЕВЬЕВ ЗАВИСИМОСТЕЙ И НЕЙРОННЫХ СЕТЕЙ ДЛЯ БИБЛИОТЕКИ NLTK



|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

# ЛИСТ УТВЕРЖДЕНИЯ RU.17701729.501310-01 ТЗ 1-1-ЛУ



Исполнитель: студент группы БПИ133

/Гукасян Ц.Г./

«\_ » 2017г.



# УТВЕРЖДЕНО RU.17701729.501310-01 ТЗ 1-1

СИНТАКСИЧЕСКИЙ АНАЛИЗАТОР НА ОСНОВЕ ДЕРЕВЬЕВ ЗАВИСИМОСТЕЙ И НЕЙРОННЫХ СЕТЕЙ ДЛЯ БИБЛИОТЕКИ NLTK

ТЕХНИЧЕСКОЕ ЗАДАНИЕ RU.17701729.501310-01 ТЗ 1-1

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Листов 13



ОГЛАВЛЕНИЕ

1. [ВВЕДЕНИЕ 42](#_bookmark0)
   1. [Наименование программы 42](#_bookmark1)
   2. [Краткая характеристика области применения 42](#_bookmark2)
2. [ОСНОВАНИЯ ДЛЯ РАЗРАБОТКИ 43](#_bookmark3)
   1. [Документы, на основаниях которых ведется разработка 43](#_bookmark4)
   2. [Наименование темы разработки 43](#_bookmark5)
3. [НАЗНАЧЕНИЕ РАЗРАБОТКИ 44](#_bookmark6)
4. [ТРЕБОВАНИЯ К ПРОГРАММЕ 45](#_bookmark7)
   1. [Требования к функциональным характеристикам 45](#_bookmark8)
   2. [Требования к формату входных и выходных данных 46](#_bookmark9)
   3. [Условия эксплуатации 46](#_bookmark10)
   4. [Требования к составу и параметрам технических средств 46](#_bookmark11)
   5. [Требования к информационной и программной совместимости 47](#_bookmark12)
5. [ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ 48](#_bookmark13)
   1. [Предварительный состав программной документации 48](#_bookmark14)
6. [ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ 49](#_bookmark15)
   1. [Ориентировочная экономическая эффективность 49](#_bookmark16)
   2. [Предполагаемая потребность 49](#_bookmark17)
7. [СТАДИИ И ЭТАПЫ РАЗРАБОТКИ 50](#_bookmark18)
8. [ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ 51](#_bookmark19)
   1. [Виды испытаний 51](#_bookmark20)
   2. [Общие требования к приемке работы 51](#_bookmark21)
9. ВВЕДЕНИЕ
   1. Наименование программы

Наименование программы – «Синтаксический анализатор на основе деревьев зависимостей и нейронных сетей для библиотеки NLTK».

* 1. Краткая характеристика области применения

Синтаксические анализаторы используются для определения синтаксической структуры предложения. Современные синтаксические анализаторы основаны на грамматике зависимостей. В грамматике зависимостей все отношения между членами предложения считаются подчинительными. Структура предложения представляется в виде дерева зависимостей – иерархии членов предложения, где на вершине сказуемое, а остальные слова его подчиненные.

Синтаксические анализаторы применяются при решении таких задач, как генерация текста и речи, извлечение информации, определение тональности текста, машинный перевод. Исследования показывают, что знание зависимостей между членами предложения значительно улучшает точность перевода. Синтаксические анализаторы также используются для извлечения семантических отношений в предложениях. Текстовые процессоры используют синтаксические анализаторы для увеличения эффективности процесса корректирования написания слов.

1. ОСНОВАНИЯ ДЛЯ РАЗРАБОТКИ
   1. Документы, на основаниях которых ведется разработка

Приказ Национального исследовательского университета "Высшая школа экономики" №2.3-02/0812-04 от 08.12.2016.

* 1. Наименование темы разработки

Программа написана в рамках темы выпускной квалификационной работы

«Синтаксический анализатор на основе деревьев зависимостей и нейронных сетей для библиотеки NLTK» (факультет компьютерных наук, департамент программной инженерии), в соответствии с учебным планом подготовки бакалавров по направлению

09.03.04 «Программная инженерия».

1. НАЗНАЧЕНИЕ РАЗРАБОТКИ

Синтаксический анализатор может быть применен в системах машинного перевода, синтезаторах речи, а также в текстовых процессорах для увеличения эффективности процесса корректирования написания слов. Данная программа позволит определить синтаксические зависимости в предложениях, а также обучить и сохранять модели анализатора для естественных языков.

Natural Language Toolkit (NLTK) – открытая и бесплатная библиотека обработки текстов на естественном языке. Благодаря NLTK, язык программирования Python является удобным средством для анализа текстов. Библиотека используется в исследованиях в области NLP и в разработке алгоритмов машинного обучения. Поскольку код библиотеки открыт и тщательно документирован, платформа имеет важное образовательное значение и является самым популярным руководством для начинающих ученых в этой области. Но в NLTK отсутствует реализация быстрого и точного алгоритма синтаксического анализа, поэтому целью данной работы является реализация алгоритма, созданного Кристофером Д. Мэннингом и Данчи Чен, и включение его в NLTK. Алгоритм использует искусственную нейронную сеть для переходов и вводит несколько новаторских концепций. Вместо унитарного кодирования анализатор использует плотные представления для слов, тегов частей речи и меток зависимостей. В нейронной сети используется кубическая функция активации вместо широко используемых функций tanh и sigmoid. В экспериментах, проведенных Мэннингом и Чен, скорость анализатора превысила скорость анализатора MST-Parser в 100 раз, при этом показатели точности LAS и UAS получились лучше.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

Программа должна предоставлять метод обучения модели синтаксического анализатора на основе размеченного корпуса текстов, сохранения обученной модели синтаксического анализатора, метод синтаксического анализа предложения на основе заранее обученной модели. Для синтаксического анализа программа должна реализовать алгоритм, предложенный Кристофером Д. Мэннингом и Данчи Чен. Анализатор должен использовать систему переходов Arc-Standard. При анализе предложения, для определения перехода анализатор должен использовать полносвязную нейронную сеть со следующими параметрами:

1. 3 слоя – входной, скрытый, выходной;
2. принимает на вход плотные представления слов предложения, их часть речи и уже определенные на тот момент зависимости;
3. на скрытом слое применяется кубическая функция активации;
4. на выходном слое применяется функция softmax.

Программа должна быть интегрирована в модуль parse библиотеки NLTK. На корпусе русскоязычных текстов СинТагРус должна быть обучена модель анализатора для русского языка. Полученная модель должна быть добавлена в репозиторий NLTK Data.

* 1. Требования к функциональным характеристикам

Программа должна предоставлять возможность выполнения перечисленных ниже функций:

1. Метод, принимающий предложение и возвращающий список синтаксических зависимостей между всеми словами этого предложения и типы этих зависимостей. Входное предложение должно быть размечено (для всех слов должна быть указана часть речи).
2. Метод, обучающий нейронную сеть на основе размеченного корпуса текстов и сохраняющий полученные параметры в файле. В корпусе текстов для всех предложений должны быть указаны отношения между словами предложения, части речи слов, начало и конец предложения.
3. Метод, читающий файл с параметрами синтаксического анализатора и возвращающий синтаксический анализатор.
4. Метод, определяющий точность синтаксического анализатора. Метод должен принимать размеченный корпус текстов, и вычислять на нем показатели UAS и LAS.
5. Метод, выполняющий синтаксический анализ неразмеченного предложения (частеречная разметка предложения должна выполняться в методе).
6. Метод, выполняющий синтаксический анализ предложений из файла формата CoNLL-U.
   1. Требования к формату входных и выходных данных

В методе 1) из 4.1. язык входного предложения должен совпадать с языком модели, используемой синтаксическим анализатором. Также должны совпадать наборы меток частей речи, используемые во входном предложении и в корпусе текстов, на основе которого была получена модель анализатора.

В методах 2) и 4) из 4.1. входной корпус должен иметь формат CoNLL-U.

В методе 5) из 4.1. для входного предложения должны быть указаны границы его членов.

Формат входного файла в методе 3) из 4.1. должен совпадать с форматом файла, получаемого на выходе метода 2) из 4.1.

* 1. Условия эксплуатации

Минимальное количество персонала, требуемого для работы программы, должно составлять не менее 1 штатной единицы:

1) конечный пользователь – разработчик.

Для работы с данной библиотекой конечный пользователь должен:

1. обладать практическими навыками работы с библиотеками в Python;
2. иметь базовое понимание работы алгоритмов машинного обучения, синтаксического анализа.
   1. Требования к составу и параметрам технических средств

Для надежной и бесперебойной работы методов рекомендуется использовать компьютер со следующими характеристиками:

1. Рекомендуемый объем оперативной памяти для надежной и бесперебойной работы точного анализатора – 0.5 ГБ или больше. Объем памяти, необходимой для анализа предложения зависит от параметров анализатора.
2. Рекомендуемый объем оперативной памяти для обучения анализатора – 1 ГБ оперативной памяти или больше. Объем необходимой памяти зависит от размера обучающих данных. Для обучения анализатора на основе корпуса текстов с размером 80 МБ и файла векторных представлений слов с размером 0.5 ГБ необходимо примерно 3 ГБ оперативной памяти.
   1. Требования к информационной и программной совместимости

Для работы программы необходим следующий состав программных средств:

1. Python 2.7 или Python 3.4+.
2. Библиотека NumPy.
3. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ
   1. Предварительный состав программной документации
4. «Синтаксический анализатор на основе деревьев зависимостей и нейронных сетей для библиотеки NLTK». Техническое задание (ГОСТ 19.201 - 78);
5. «Синтаксический анализатор на основе деревьев зависимостей и нейронных сетей для библиотеки NLTK». Программа и методика испытаний (ГОСТ 19.301 - 78);
6. «Синтаксический анализатор на основе деревьев зависимостей и нейронных сетей для библиотеки NLTK». Текст программы (ГОСТ 19.401 - 78);
7. «Синтаксический анализатор на основе деревьев зависимостей и нейронных сетей для библиотеки NLTK». Руководство программиста (ГОСТ 19.504 - 79).
8. ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ
   1. Ориентировочная экономическая эффективность

В рамках данной работы расчет экономической эффективности не предусмотрен.

Программу можно использовать, распространять и модифицировать бесплатно.

* 1. Предполагаемая потребность

Программа будет представлять интерес для изучающих машинное обучение, обработку текстов, а также для разработчиков, занимающихся созданием алгоритмов машинного перевода, синтезаторов речи и т.д.

Как пример реализации современного алгоритма синтаксического анализа, программа будет служит образовательную роль. Программа предоставляет реализацию нейронной сети с использованием библиотеки NumPy для векторизации матричных операций.

Программу можно также использовать для создания других инструментов обработки естественного языка (например, программы извлечения именованных сущностей). Синтаксические структуры, получаемые в результате работа анализатора, также могут быть применены для генерации признаков в программах определения тональности текстов.

1. СТАДИИ И ЭТАПЫ РАЗРАБОТКИ
2. Техническое задание
   1. Обоснование необходимости разработки программы
      1. Постановка задачи;
      2. Сбор исходных материалов.
   2. Научно-исследовательские работы
      1. Определение структуры входных и выходных данных;
      2. Предварительный выбор методов решения задач;
      3. Определение требований к техническим средствам.
   3. Разработка и утверждение технического задания
      1. Определение требований к программе;
      2. Определение стадий, этапов и сроков разработки программы и документации на неё;
      3. Согласование и утверждение технического задания.
3. Технический проект
   1. Разработка технического проекта
      1. Уточнение структуры входных и выходных данных
      2. Разработка алгоритма решения задачи
      3. Определение формы представления входных и выходных данных
      4. Разработка структуры программы
      5. Окончательное определение конфигурации технических средств
   2. Утверждение технического проекта
      1. Разработка пояснительной записки.
      2. Согласование и утверждение технического проекта.
4. Рабочий проект
   1. Разработка программы
      1. Программирование и отладка.
   2. Разработка программной документации
      1. Разработка программных документов в соответствии с требованиями ГОСТ 19.101-77.
   3. Испытания библиотеки
      1. Разработка, согласование и утверждение порядка и методики испытаний.
      2. Корректировка программы и программной документации по результатам испытаний.

8. ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ

* 1. Виды испытаний

Осуществляется функциональное тестирование программы.

Функциональное тестирование осуществляется в соответствии с документом

«Синтаксический анализатор на основе деревьев зависимостей и нейронных сетей для библиотеки NLTK». Программа и методика испытаний.

* 1. Общие требования к приемке работы

Прием программного продукта происходит при его работоспособности при различных входных данных, при выполнении указанных в данном документе функций и требований и при наличии полной документации.

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в документе | № документа | Входящий № сопроводитель ного документа и  дата | Подпись | Дата |
| измененных | замененных | новых | аннулиро ванных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Приложение B. Руководство программиста

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук Департамент программной инженерии

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Доцент факультета компьютерных наук базовой кафедры «Системное программирование» НИУ ВШЭ,  канд. физ.-мат. наук  Д.Ю. Турдаков  « » 2017г. | УТВЕРЖДАЮ  Академический руководитель образовательной программы  «Программная инженерия»  В.В. Шилов  « » 2017г. |

СИНТАКСИЧЕСКИЙ АНАЛИЗАТОР НА ОСНОВЕ ДЕРЕВЬЕВ ЗАВИСИМОСТЕЙ И НЕЙРОННЫХ СЕТЕЙ ДЛЯ БИБЛИОТЕКИ NLTK



|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

РУКОВОДСТВО ПРОГРАММИСТА

# ЛИСТ УТВЕРЖДЕНИЯ RU.17701729.501310-01 33 1-1-ЛУ



Исполнитель: студент группы БПИ133

/Гукасян Ц.Г./

«\_ » 2017г.



# 2017

УТВЕРЖДЕНО RU.17701729.501310-01 33 1-1

СИНТАКСИЧЕСКИЙ АНАЛИЗАТОР НА ОСНОВЕ ДЕРЕВЬЕВ ЗАВИСИМОСТЕЙ И НЕЙРОННЫХ СЕТЕЙ ДЛЯ БИБЛИОТЕКИ NLTK

РУКОВОДСТВО ПРОГРАММИСТА RU.17701729.501310-01 33 1-1

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Листов 9



# 2017

Оглавление

1. НАЗНАЧЕНИЕ АНАЛИЗАТОРА 57
2. ТРЕБОВАНИЯ 58
   1. Требования к параметрам технических средств 58
   2. Требования к программной совместимости 58
3. ПРЕДОСТАВЛЯЕМЫЕ ФУНКЦИИ 59
4. ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ 60
5. ХАРАКТЕРИСТИКИ АНАЛИЗАТОРА 62
   1. Точность анализатора 62
   2. Временные характеристики 62
      1. НАЗНАЧЕНИЕ АНАЛИЗАТОРА

Синтаксический анализатор может быть применен в системах машинного перевода, синтезаторах речи, а также в текстовых процессорах для увеличения эффективности процесса корректирования написания слов. Данная программа позволит определить синтаксические зависимости в предложениях, а также обучить и сохранять модели анализатора для естественных языков.

Natural Language Toolkit (NLTK) – библиотека на языке программирования Python для обработки естественного языка. Благодаря NLTK, язык программирования Python является удобным средством для анализа текстов. Библиотека используется в исследованиях в области NLP и в разработке алгоритмов машинного обучения. Поскольку код библиотеки открыт и тщательно документирован, платформа имеет важное образовательное значение и является самым популярным руководством для начинающих ученых в этой области. В этой работе предлагается интегрированная в NLTK реализация быстрого и точного алгоритма синтаксического анализа, созданного Кристофером Д. Мэннингом и Данчи Чен. Алгоритм использует искусственную нейронную сеть для переходов и вводит несколько новаторских концепций. Вместо унитарного кодирования анализатор использует плотные представления для слов, тегов частей речи и меток зависимостей. В нейронной сети используется кубическая функция активации вместо широко используемых функций tanh и sigmoid.

Для использования данного анализатора в своей программе, разработчик должен подключить модуль parse.stanfordnndep библиотеки NLTK.

* + 1. ТРЕБОВАНИЯ
  1. Требования к параметрам технических средств

Для надежной и бесперебойной работы методов рекомендуется использовать компьютер со следующими характеристиками:

1. Рекомендуемый объем оперативной памяти для надежной и бесперебойной работы точного анализатора – 0.5 ГБ или больше. Объем памяти, необходимой для анализа предложения зависит от параметров анализатора.
2. Рекомендуемый объем оперативной памяти для обучения анализатора – 1 ГБ оперативной памяти или больше. Объем необходимой памяти зависит от размера обучающих данных. Для обучения анализатора на основе корпуса текстов с размером 80 МБ и файла векторных представлений слов с размером 0.5 ГБ необходимо примерно 3 ГБ оперативной памяти.
   1. Требования к программной совместимости

Для работы программы необходим следующий состав программных средств:

1. Python 2.7 или Python 3.4+.
2. Библиотека NumPy.
   * 1. ПРЕДОСТАВЛЯЕМЫЕ ФУНКЦИИ

Синтаксический анализатор написан на языке программирования Python 3.

Основной класс анализатора – DependencyParser, который содержит методы обучения, сохранения модели, анализа предложения, текстового файла, текстового файла в формате CoNLL-U. Класс DependencyTree – реализация дерева синтаксических зависимостей. Поле head класса DependencyTree хранит список родителей членов предложения в дереве разбора, поле label – список типов соответствующих зависимостей. В таблицах 3.1, 3.2 приведены описания основных методов.

Таблица 3.1

Описание основных методов класса DependencyParser

|  |  |
| --- | --- |
| Метод | Назначение |
| parse | Принимает предложение с указанными частями речи и возвращает  дерево разбор этого предложения – экземпляр класса DependencyTree. |
| train | Принимает файл c размеченными предложениями в формате CoNLL- U и обучает модель парсера. Если в парсере предварительно не был указан тэггер, обучает усредненный перцептрон для выполнения  частеречной разметки предложений. |
| test\_conll | Принимает файл c размеченными предложениями в формате CoNLL- U и определяет точность парсера (UAS, LAS) на этих предложениях, сохраняет построенные для этих предложений деревья разбора в другом файле в формате CoNLL-U. Метод работает, только если  парсер уже обучен или была загружена модель парсера. |
| write\_model\_file | Записывает модель парсера в файл. |
| load\_model\_file | Загружает модель парсера и обновляет/задает параметры парсера и  классификатора. |

Таблица 3.2

Описание основных методов класса DependencyTree

|  |  |
| --- | --- |
| Метод | Назначение |
| add | Добавляет новую зависимость в дерево разбора. |
| get\_head | Возвращает номер родителя переданного члена предложения. |
| get\_label | Возвращает тип зависимости от родителя для переданного члена  предложения. |
| get\_root | Возвращает номер корня дерева разбора. |

* + 1. ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

В таблицах 4.1 и 4.2 описаны параметры методов, значение по умолчанию и тип этих параметров, а также возвращаемое значение методов.

Таблица 4.1 Параметры и возвращаемое значение основных методов класса DependencyParser

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Параметр | Тип параметра | Значение по  умолчанию | Тип возвращаемого  значения |
| parse | sentence | list ({ “word”:слово, “pos”:метка  }) | нет | DependencyTree |
|  | tagged | boolean | True |  |
| train | train\_file | string | нет | нет |
|  | dev\_file | string | нет |  |
|  | model\_file | string | нет |  |
|  | embed\_file | string | нет |  |
|  | train\_tagger | boolean | False |  |
|  | tagger\_loc | string | None |  |
| test\_conll | test\_file | string | нет | float |
|  | out\_file | string | нет |  |
|  | tagged | boolean | True |  |
| write\_model\_file | model\_file | string | нет | нет |
|  | encoding | string | ‘utf8’ |  |
| load\_model\_file | model\_file | string | нет | нет |
|  | encoding | string | ‘utf8’ |  |
| load\_from\_model\_file | model\_file | string | нет | DependencyParser |
|  | encoding | encoding | ‘utf8’ |  |

Метод parse класса DependencyParser принимает список значений типа dict. Каждое значение в списке имеет ключ “word”, в котором хранится член предложения, и ключ “pos” для части речи, если tagged – True. Через параметр model\_file передается путь к файлу, где хранится/нужно сохранить модель анализатора.

В параметрах с названиями train\_file, dev\_file, test\_file передается путь к файлам формата CoNLL-U. В каждой строке файла формата CoNLL-U хранятся атрибуты одного члена предложения:

1. порядковый номер в предложении;
2. токен (член предложения);
3. обработанный токен;
4. часть речи в формате Universal Dependencies [30];
5. часть речи из корпуса;
6. морфологические признаки;
7. порядковый номер родителя в дереве разбора;
8. тип синтаксической зависимости от родителя.

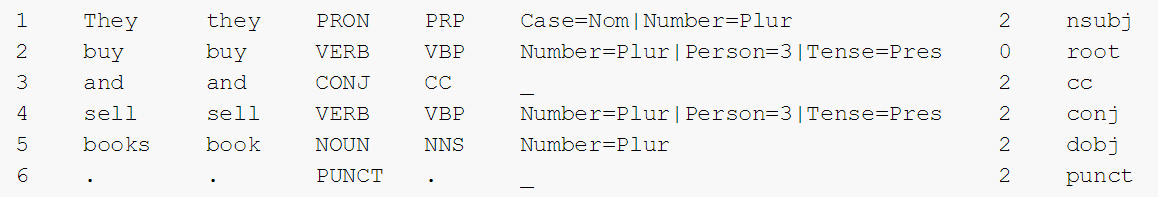
В файле атрибуты члена предложения разделяются одним символом табуляции, а предложения – пустой строкой. В анализаторе 5-ый (метка части речи из корпуса) и 6-ой атрибуты (морфологические признаки) не используются.

Схема 4.1. Размеченное предложение «They buy and sell books.» в формате CoNLL-U [27]. В таблице 4.2 параметры head и k указывают на порядковый номер члена предложения.

Через label передается метка синтаксической зависимости.

Таблица 4.2 Параметры и возвращаемое значение основных методов класса DependencyTree

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Параметр | Тип параметра | Значение по  умолчанию | Тип возвращаемого  значения |
| add | head | int | нет | нет |
|  | label | string | нет |  |
| get\_head | k | string | нет | int |
| get\_label | k | int | нет | string |
| get\_root | нет | нет | нет | int |

5. ХАРАКТЕРИСТИКИ АНАЛИЗАТОРА

* 1. Точность анализатора

Синтаксический анализатор показал точность 84.7% UAS и 78.71% LAS на корпусе русскоязычных текстов СинТагРус. Точность анализатора на корпусе английских текстов WSJ – 92% UAS и 90.7% LAS.

* 1. Временные характеристики

Синтаксический анализ предложения выполняется за линейное от длины входа время (длина входа – количество слов и знаков пунктуации в предложении).

# ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в документе | № документа | Входящий № сопроводитель ного  документа и дата | Подпись | Дата |
| измененных | замененных | новых | аннулиро ванных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Приложение C. Программа и методика испытаний

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук Департамент программной инженерии

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Доцент факультета компьютерных наук базовой кафедры «Системное программирование» НИУ ВШЭ,  канд. физ.-мат. наук  Д.Ю. Турдаков  « » 2017г. | УТВЕРЖДАЮ  Академический руководитель образовательной программы  «Программная инженерия»  В.В. Шилов  « » 2017г. |

СИНТАКСИЧЕСКИЙ АНАЛИЗАТОР НА ОСНОВЕ ДЕРЕВЬЕВ ЗАВИСИМОСТЕЙ И НЕЙРОННЫХ СЕТЕЙ ДЛЯ БИБЛИОТЕКИ NLTK



|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

# ЛИСТ УТВЕРЖДЕНИЯ RU.17701729.501310-01 51 1-1-ЛУ



Исполнитель: студент группы БПИ133

/Гукасян Ц.Г./

«\_ » 2017г.



# 2017

УТВЕРЖДЕНО RU.17701729.501310-01 51 1-1

СИНТАКСИЧЕСКИЙ АНАЛИЗАТОР НА ОСНОВЕ ДЕРЕВЬЕВ ЗАВИСИМОСТЕЙ И НЕЙРОННЫХ СЕТЕЙ ДЛЯ БИБЛИОТЕКИ NLTK

ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ RU.17701729.501310-01 51 1-1

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Листов 13



# 2017

Оглавление

1. ОБЪЕКТ ИСПЫТАНИЙ 68
   1. Наименование программы 68
   2. Область применения программы 68
   3. Обозначение испытуемой библиотеки 68
2. ЦЕЛЬ ИСПЫТАНИЙ 69
3. ТРЕБОВАНИЯ К ПРОГРАММЕ 70
   1. Требования к функциональным характеристикам 70
   2. Требования к формату входных и выходных данных 70
   3. Требования к составу и параметрам технических средств 71
   4. Требования к информационной и программной совместимости 71
4. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ 72
5. СРЕДСТВА И ПОРЯДОК ИСПЫТАНИЙ 73
   1. Порядок проведения испытаний 73
   2. Условия проведения испытаний 73
6. МЕТОДЫ ИСПЫТАНИЙ 74
   1. Испытание выполнения требований к программной документации 74
   2. Испытание выполнения требований к функциональным характеристикам 74
      1. Синтаксический анализ предложения 74
      2. Обучение анализатора 75
      3. Чтение анализатора из файла с параметрами 75
      4. Определение точности анализатора на файле формата CoNLL-U 75
      5. Анализ предложений из файла формата CoNLL-U 76
      6. Синтаксический анализ неразмеченного предложения 76
         1. ОБЪЕКТ ИСПЫТАНИЙ
   3. Наименование программы

Наименование программы – «Синтаксический анализатор на основе деревьев зависимостей и нейронных сетей для библиотеки NLTK».

* 1. Область применения программы

Синтаксические анализаторы используются для определения синтаксической структуры предложения. Современные синтаксические анализаторы основаны на грамматике зависимостей. В грамматике зависимостей все отношения между членами предложения считаются подчинительными. Структура предложения представляется в виде дерева зависимостей – иерархии членов предложения, где на вершине сказуемое, а остальные слова его подчиненные.

Синтаксические анализаторы применяются при решении таких задач, как генерация текста и речи, извлечение информации, определение тональности текста, машинный перевод. Исследования показывают, что знание зависимостей между членами предложения значительно улучшает точность перевода. Синтаксические анализаторы также используются для извлечения семантических отношений в предложениях. Текстовые процессоры используют синтаксические анализаторы для увеличения эффективности процесса корректирования написания слов.

* 1. Обозначение испытуемой библиотеки Обозначение испытуемой программы – stanfordnndep.
     + 1. ЦЕЛЬ ИСПЫТАНИЙ

Цель проведения испытаний – проверка соответствия характеристик разработанной программы функциональным требованиям и отдельным требованиям к надежности, изложенных в документе Техническое задание к данной программе.

* + - 1. ТРЕБОВАНИЯ К ПРОГРАММЕ
  1. Требования к функциональным характеристикам

Программа должна предоставлять возможность выполнения перечисленных ниже функций:

1. Метод, принимающий предложение и возвращающий список синтаксических зависимостей между всеми словами этого предложения и типы этих зависимостей. Входное предложение должно быть размечено (для всех слов должна быть указана часть речи).
2. Метод, обучающий нейронную сеть на основе размеченного корпуса текстов и сохраняющий полученные параметры в файле. В корпусе текстов для всех предложений должны быть указаны отношения между словами предложения, части речи слов, начало и конец предложения.
3. Метод, читающий файл с параметрами синтаксического анализатора и возвращающий синтаксический анализатор.
4. Метод, определяющий точность синтаксического анализатора. Метод должен принимать размеченный корпус текстов, и вычислять на нем показатели UAS и LAS.
5. Метод, выполняющий синтаксический анализ неразмеченного предложения (частеречная разметка предложения должна выполняться в методе).
6. Метод, выполняющий синтаксический анализ предложений из файла формата CoNLL-U.
   1. Требования к формату входных и выходных данных

В методе 1) из 4.1. язык входного предложения должен совпадать с языком модели, используемой синтаксическим анализатором. Также должны совпадать наборы меток частей речи, используемые во входном предложении и в корпусе текстов, на основе которого была получена модель анализатора.

В методах 2) и 4) из 4.1. входной корпус должен иметь формат CoNLL-U.

В методе 5) из 4.1. для входного предложения должны быть указаны границы его членов.

Формат входного файла в методе 3) из 4.1. должен совпадать с форматом файла, получаемого на выходе метода 2) из 4.1.

* 1. Требования к составу и параметрам технических средств

Для надежной и бесперебойной работы методов рекомендуется использовать компьютер со следующими характеристиками:

1. Рекомендуемый объем оперативной памяти для надежной и бесперебойной работы точного анализатора – 0.5 ГБ или больше. Объем памяти, необходимой для анализа предложения зависит от параметров анализатора.
2. Рекомендуемый объем оперативной памяти для обучения анализатора – 1 ГБ оперативной памяти или больше. Объем необходимой памяти зависит от размера обучающих данных. Для обучения анализатора на основе корпуса текстов с размером 80 МБ и файла векторных представлений слов с размером 0.5 ГБ необходимо примерно 3 ГБ оперативной памяти.
   1. Требования к информационной и программной совместимости

Для работы программы необходим следующий состав программных средств:

1. Python 2.7 или Python 3.4+.
2. Библиотека NumPy.
   * + 1. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ
3. «Синтаксический анализатор на основе деревьев зависимостей и нейронных сетей для библиотеки NLTK». Техническое задание (ГОСТ 19.201 - 78);
4. «Синтаксический анализатор на основе деревьев зависимостей и нейронных сетей для библиотеки NLTK». Программа и методика испытаний (ГОСТ 19.301 - 78);
5. «Синтаксический анализатор на основе деревьев зависимостей и нейронных сетей для библиотеки NLTK». Текст программы (ГОСТ 19.401 - 78);
6. «Синтаксический анализатор на основе деревьев зависимостей и нейронных сетей для библиотеки NLTK». Руководство программиста (ГОСТ 19.504 - 79).
   * + 1. СРЕДСТВА И ПОРЯДОК ИСПЫТАНИЙ
   1. Порядок проведения испытаний

Испытания должны проводиться в следующем порядке:

1) проверка требований к программной документации;

4) проверка требований к функциональным характеристикам.

* 1. Условия проведения испытаний

Для проведения испытаний нужен разработчик, имеющий следующие навыки:

1. практические навыки работы с библиотеками в Python;
2. базовое понимание работы алгоритмов машинного обучения и синтаксического анализа.
   * + 1. МЕТОДЫ ИСПЫТАНИЙ
   1. Испытание выполнения требований к программной документации

Состав программной документации проверяется визуально, проверяется наличие всех подписей. Также визуально проверяется соответствие документации требованиям ГОСТ.

* 1. Испытание выполнения требований к функциональным характеристикам
     1. Синтаксический анализ предложения

В метод анализа передается размеченное предложение (для каждого члена указана часть речи). Метод возвращает дерево разбора предложения – объект класса DependencyTree. В поле head возвращенного объекта для каждого члена предложения должен храниться номер его родителя в, а в поле label – тип соответствующей синтаксической зависимости. Метки частей речи в предложении должны совпадать с метками, используемыми алгоритмом частеречной разметки. Перед вызовом этого метода анализатор следует обучить или прочитать его параметры из файла.

На рис. 6.2.1 приведен пример ожидаемого поведения испытуемой программы.

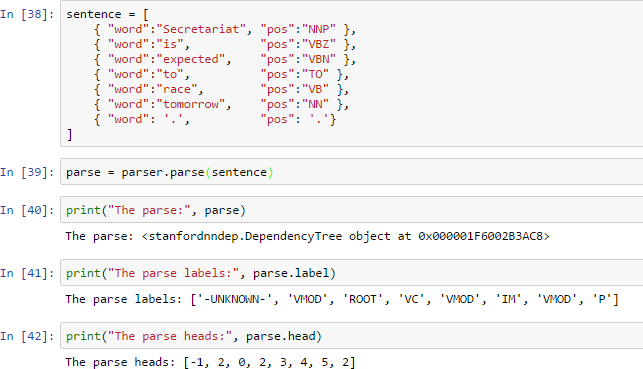


Рисунок 6.2.1. Синтаксический анализ размеченного предложения «Secretariat is expected to race tomorrow.» испытуемой программой.

* + 1. Обучение анализатора

Перед обучением, в параметрах анализатора указывается количество тренировочных итераций, размер вектора представления в соответствии с используемой моделью векторных представлений.

В метод обучения анализатора передаются пути к файлам с обучающей выборкой (формат – CoNLL-U), тестовой выборкой (формат – CoNLL-U), название файла, где нужно сохранить обученную модель, и путь к файлу с векторными представлениями слов языка, для которого обучается анализатор. У метода нет возвращаемого значения, но в результате его выполнения получается и сохраняется модель анализатора.

На рис. 6.2.2 приведен пример ожидаемого поведения испытуемой программы.

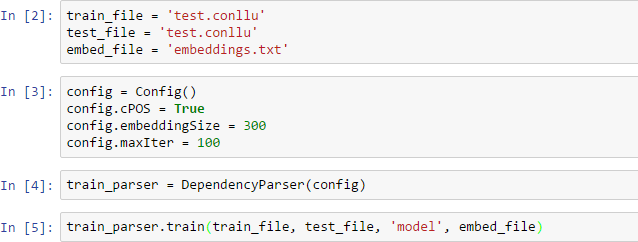


Рисунок 6.2.2. Обучение синтаксического анализатора

* + 1. Чтение анализатора из файла с параметрами

Для создания объекта синтаксического анализатора в соответствующий метод передается путь к файлу с сохраненной моделью анализатора. Метод возвращает синтаксический анализатор с параметрами из этого файла.

На рис. 6.2.3 приведен пример ожидаемого поведения испытуемой программы.

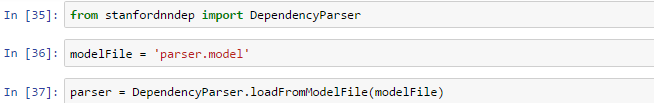


Рисунок 6.2.3. Создание синтаксического анализатора на основе файла с параметрами

* + 1. Определение точности анализатора на файле формата CoNLL-U

В метод определения точности анализатора передается путь к файлу формата CoNLL-U и путь к файлу, где нужно сохранить построенные анализатором деревья разбора. Метод

вычисляет и выводит значение метрик UAS и LAS (в процентах), а также скорость обработки предложений анализатором – количество обработанных предложений в секунду. Метод возвращает значение UAS (в процентах). Перед вызовом этого метода анализатор следует обучить или прочитать его параметры из файла.

На рис. 6.2.4 приведен пример ожидаемого поведения испытуемой программы.

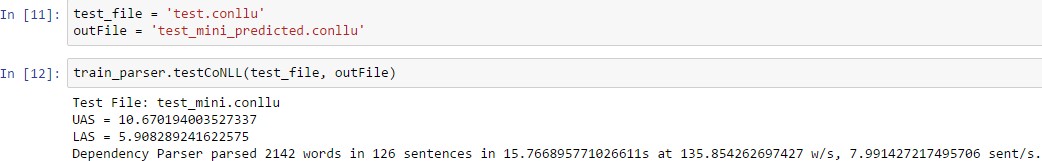


Рисунок 6.2.4. Синтаксический анализ предложений из файла формата CoNLL-U и определение точности анализатора.

* + 1. Анализ предложений из файла формата CoNLL-U

Анализ предложений из файла формата CoNLL-U выполняется методом, описанным в

6.2.4.

* + 1. Синтаксический анализ неразмеченного предложения



Рисунок 6.2.6. Синтаксический анализ неразмеченного предложения

В метод анализа передается предложение (части речи не указаны) и параметр, указывающий, что предложение не размечено. Метод возвращает дерево разбора предложения – объект класса DependencyTree. В поле head возвращенного объекта для каждого члена предложения должен храниться номер его родителя в, а в поле label – тип соответствующей синтаксической зависимости. Перед вызовом этого метода анализатор следует обучить или прочитать его параметры из файла. Также нужно обучить или задать частеречный тэггер анализатора, который используется для определения частей речи членов предложения.

На рис. 6.2.6 приведен пример ожидаемого поведения испытуемой программы.

# ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в документе | № документа | Входящий № сопроводитель ного  документа и дата | Подпись | Дата |
| измененных | замененных | новых | аннулиро ванных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Приложение D. Текст Программы

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук Департамент программной инженерии

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Доцент факультета компьютерных наук базовой кафедры «Системное программирование» НИУ ВШЭ,  канд. физ.-мат. наук  Д.Ю. Турдаков  « » 2017г. | УТВЕРЖДАЮ  Академический руководитель образовательной программы  «Программная инженерия»  В.В. Шилов  « » 2017г. |

СИНТАКСИЧЕСКИЙ АНАЛИЗАТОР НА ОСНОВЕ ДЕРЕВЬЕВ ЗАВИСИМОСТЕЙ И НЕЙРОННЫХ СЕТЕЙ ДЛЯ БИБЛИОТЕКИ NLTK



|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

ТЕКСТ ПРОГРАММЫ

# ЛИСТ УТВЕРЖДЕНИЯ RU.17701729.501310-01 12 1-1-ЛУ



Исполнитель: студент группы БПИ133

/Гукасян Ц.Г./

«\_ » 2017г.



# УТВЕРЖДЕНО RU.17701729.501310-01 12 1-1

СИНТАКСИЧЕСКИЙ АНАЛИЗАТОР НА ОСНОВЕ ДЕРЕВЬЕВ ЗАВИСИМОСТЕЙ И НЕЙРОННЫХ СЕТЕЙ ДЛЯ БИБЛИОТЕКИ NLTK

ТЕКСТ ПРОГРАММЫ RU.17701729.501310-01 12 1-1

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Листов 4



Оглавление

1. Класс DependencyParser 83
2. Класс DependencyTree 83
3. Класс Dataset 83
4. Класс Example 83
5. Класс ParsingSystem 83
6. Класс ArcStandard 83
7. Класс Configuration 83
8. Класс Classifier 83
9. Класс CostFunction 83
10. Класс Cost 83
11. Класс Config 83
12. Вспомогательные методы 83

Текст программы находится в архиве ВКР\_ГУКАСЯН.zip на носителе информации типа компакт-диск в связи с большим объемом исходного кода.

# ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в документе | № документа | Входящий № сопроводитель ного  документа и дата | Подпис ь | Дата |
| измененн ых | замененны х | новых | аннулиро ванных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Приложение E. Диаграмма вариантов использования

