МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Кубанский государственный университет»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра вычислительных технологий

МЕТОДИЧЕСКИЕ УКАЗАНИЯ к лабораторным работам

по дисциплине

Б1.В.04 «ОСНОВЫ КОМПЬЮТЕРНОЙ ЛИНГВИСТИКИ»

Направление подготовки
02.03.02 **Фундаментальная информатика и информационные технологии**
Профиль бакалавриата Математическое и программное обеспечение
компьютерных технологий

Краснодар – 2021 г.

СОДЕРЖАНИЕ

Введение

Компьютерная лингвистика – междисциплинарная область, которая возникла на стыке таких наук, как лингвистика, математика, информатика и искусственный интеллект. В своем развитии она до сих пор вбирает и применяет разработанные в этих науках методы и инструменты.

Задача компьютерной лингвистики – разработка методов и средств построения **лингвистических процессоров** для различных прикладных задач по автоматической обработке текстов на естественном языке. Разработка лингвистического процессора для некоторой прикладной задачи предполагает формальное описание лингвистических свойств обрабатываемого текста, которое может рассматриваться как **модель текста** или **модель языка**.

Сложности моделирования естественного языка

Сложность моделирования в компьютерной лингвистике связана с тем, что естественный язык – большая открытая многоуровневая система знаков, возникшая для обмена информацией в процессе практической деятельности человека, и постоянно изменяющаяся в связи с этой деятельностью.

Текст на естественном языке составлен из отдельных единиц, и возможно несколько способов разбиения текста на единицы, относящихся к разным уровням.

Общепризнано существование следующих уровней:

1) уровень предложений – **синтаксический уровень**;
2) уровень слов – **морфологический уровень**;
3) уровень фонем – **фонологический уровень**.

Общие этапы и модули обработки текстов

Сложность формального описания естественного языка и его обработки ведет к разбиению этого процесса на отдельные этапы, соответствующие уровням языка. Большинство современных лингвистических процессоров относятся к модульному типу, в котором каждому уровню анализа или синтеза текста соответствует отдельный модуль процессора. В случае анализа текста отдельные модули лингвистического процессора выполняют:

1) **графематический анализ**, то есть выделение в тексте предложений и словоформ, точнее **токенов** – переход от символов к словам;

2) **морфологический анализ** – переход от словоформ к их леммам или **основам**;

3) **синтаксический анализ** – выявление синтаксических связей слов и грамматической структуры предложений;

4) **семантический и прагматический анализ**, при котором определяется смысл фраз и соответствующая реакция системы, в рамках которой работает лингвистический процессор.

Таким образом, лингвистический процессор можно рассматривать как многоэтапный преобразователь, переводящий в случае анализа текста каждое его предложение во внутреннее представление его смысла и наоборот в случае синтеза.

Перед началом работы…

Установка необходимого ПО

1. Язык Python. Инструкция по установке:
https://pythonworld.ru/osnovy/skachat-python.html
2. Любая IDE для языка Python. Инструкция для Wing IDE:
https://docplayer.ru/28165304-Nachalo-raboty-v-python-3-i-wing-ide-101.html

3. Пакет NLTK. Инструкция к установке (на английском языке):
https://www.nltk.org/install.html

4. Пакет pymorphy2. Для установки необходимо перейти в папку, куда установлен Python, затем в папку Scripts, открыть командную строку и ввести следующее:

pip install pymorphy2

Аналогичным образом устанавливаются другие пакеты, которые будут перечислены по ходу работ.

ЛАБОРАТОРНАЯ РАБОТА №1

Графематический анализ

Цель. Освоить основы работы с NLTK. Посмотреть возможности NLTK.

1.1. NLTK

NLTK – это ведущая платформа для создания программ на Python для работы с данными на естественном языке. Он предоставляет простые в использовании интерфейсы для более чем 50 корпусов и лексических ресурсов, таких как WordNet, наряду с набором библиотек обработки текста для классификации, токенизации, обработки по меткам, разметки, анализа и семантического мышления, оболочек для промышленных библиотек NLP, и активный дискуссионный форум.

Благодаря практическому руководству, в котором представлены основы программирования, а также темы компьютерной лингвистики и исчерпывающая документация по API, NLTK подходит для лингвистов, инженеров, студентов, преподавателей, исследователей и пользователей отрасли. NLTK доступен для Windows, Mac OS X и Linux. Более того, NLTK – это бесплатный проект с открытым исходным кодом, управляемый сообществом.

NLTK был назван «прекрасным инструментом для обучения и работы в области компьютерной лингвистики с использованием Python» и «удивительной библиотекой для игры на естественном языке».

1.1.1. Пакет sent_tokenize

Как следует из названия, этот пакет разделит входной текст на предложения.

Пример 1. Разделение входного текста на предложения

```
from nltk import sent_tokenize

text = "Предложение. Предложение, которое содержит запятую.
Восклицательный знак! Вопрос?"
sents = sent_tokenize(text)
print(sents) # ['Предложение.', 'Предложение, которое
содержит запятую.', 'Восклицательный знак!', 'Вопрос?']
```

1.1.2. Пакет word_tokenize

Этот пакет делит введенный текст на слова.

Пример 2. Разделение введенного текста на слова

```
from nltk import word_tokenize

sent = "В этом предложении есть много слов, мы их разделим."
print(word_tokenize(sent)) # ['В', 'этом', 'предложении',
'есть', 'много', 'слов', ',', 'мы', 'их', 'разделим', '.']
```

1.1.3. Пакет stopwords

В NLTK есть предустановленный список стоп-слов. Перед первым использованием вам понадобится его скачать.

Пример 3. Скачивание списка стоп-слов

```
from nltk import download

download('stopwords')
```

После скачивания можно импортировать пакет stopwords и посмотреть на сами слова.

Пример 4. Просмотр стоп-слов

```
from nltk.corpus import stopwords

print(stopwords.words('russian')) # ['и', 'в', 'во', 'не',
'что', 'он', 'на', 'я', 'с', 'со', 'как', 'а', 'то', 'все',
'она', 'так', 'его', 'но', 'да', 'ты', 'к', 'у', 'же', 'вы',
'за', 'бы', 'по', 'только', 'ее', 'мне', 'было', 'вот',
'от', 'меня', 'еще', 'нет', 'о', 'из', 'ему', 'теперь',
'когда', 'даже', 'ну', 'вдруг', 'ли', 'если', 'уже', 'или',
'ни', 'быть', 'был', 'него', 'до', 'вас', 'нибудь', 'опять',
'уж', 'вам', 'ведь', 'там', 'потом', 'себя', 'ничего', 'ей',
'может', 'они', 'тут', 'где', 'есть', 'надо', 'ней', 'для',
'мы', 'тебя', 'их', 'чем', 'была', 'сам', 'чтоб', 'без',
'будто', 'чего', 'раз', 'тоже', 'себе', 'под', 'будет', 'ж',
'тогда', 'кто', 'этот', 'того', 'потому', 'этого', 'какой',
'совсем', 'ним', 'здесь', 'этом', 'один', 'почти', 'мой',
'тем', 'чтобы', 'нее', 'сейчас', 'были', 'куда', 'зачем',
'всех', 'никогда', 'можно', 'при', 'наконец', 'два', 'об',
'другой', 'хоть', 'после', 'над', 'больше', 'тот', 'через',
'эти', 'нас', 'про', 'всего', 'них', 'какая', 'много',
'разве', 'три', 'эту', 'моя', 'впрочем', 'хорошо', 'свою',
'этой', 'перед', 'иногда', 'лучше', 'чуть', 'том', 'нельзя',
'такой', 'им', 'более', 'всегда', 'конечно', 'всю', 'между']
```

Рассмотрим, как можно удалить стоп-слова из предложения.

Пример 5. Удаление стоп-слов из предложения

```
from nltk import word_tokenize
from nltk.corpus import stopwords

stop_words = set(stopwords.words('russian'))
sentence = 'Нарды - одна из старейших известных настольных
игр.'

words = word_tokenize(sentence)
```

```
without_stop_words = [word for word in words if not(word in
stop_words)]
print(without_stop_words) # ['Нарды', '–', 'одна',
'старейших', 'известных', 'настольных', 'игр', '.']
```

1.2. Задания к лабораторной работе №1

1. Выполните примеры 1-5.
2. Разработайте графематический анализатор.
3. Сформируйте отчет с текстом программы и скринами результатов.

Требования к отчету к ЛР №1

1. Листинг к заданию 2.
2. Краткое описание пакетов и функций.
3. Готовность ответить на вопросы по примерам из текста ЛР и домашних заданий.

ЛАБОРАТОРНАЯ РАБОТА №2

Морфологический анализ

Цель. Освоить основы работы с pymorphy2. Посмотреть возможности pymorphy2.

2.1. pymorphy2

pymorphy2 написан на языке Python. Он умеет:
1) приводить слово к нормальной форме;
2) ставить слово в нужную форму;
3) возвращать грамматическую информацию о слове.

При работе используется словарь OpenCorpora; для незнакомых слов строятся гипотезы. Библиотека достаточно быстрая: в настоящий момент скорость работы – от нескольких тысяч слов в секунду до более, чем 100 тысяч слов в секунду; потребление памяти – от 10 до 20 Мб; полностью поддерживается буква ё.

2.1.1. Морфологический анализ

Морфологический анализ – это определение характеристик слова на основе того, как это слово пишется. При морфологическом анализе не используется информация о соседних словах.

В pymorphy2 для морфологического анализа русских слов есть класс MorphAnalyzer. С помощью метода MorphAnalyzer.parse() можно разобрать слово.

Пример 1. Разбор слова

```
from pymorphy2 import MorphAnalyzer

morph = MorphAnalyzer()
print(morph.parse('стали')) # [
Parse(word='стали', tag=OpencorporaTag('VERB,perf,intr
plur,past,indc'), normal_form='стать', score=0.984662,
methods_stack=((<DictionaryAnalyzer>, 'стали', 904, 4),)),
Parse(word='стали', tag=OpencorporaTag('NOUN,inan,femn
sing,gent'), normal_form='сталь', score=0.003067,
methods_stack=((<DictionaryAnalyzer>, 'стали', 13, 1),)),
Parse(word='стали', tag=OpencorporaTag('NOUN,inan,femn
sing,datv'), normal_form='сталь', score=0.003067,
methods_stack=((<DictionaryAnalyzer>, 'стали', 13, 2),)),
Parse(word='стали', tag=OpencorporaTag('NOUN,inan,femn
sing,loct'), normal_form='сталь', score=0.003067,
methods_stack=((<DictionaryAnalyzer>, 'стали', 13, 5),)),
Parse(word='стали', tag=OpencorporaTag('NOUN,inan,femn
plur,nomn'), normal_form='сталь', score=0.003067,
methods_stack=((<DictionaryAnalyzer>, 'стали', 13, 6),)),
Parse(word='стали', tag=OpencorporaTag('NOUN,inan,femn
plur,accs'), normal_form='сталь', score=0.003067,
methods_stack=((<DictionaryAnalyzer>, 'стали', 13, 9),))
]
```

В этом примере слово «стали» может быть разобрано и как глагол, и как существительное. На основе одной лишь информации о том, как слово пишется, понять, какой разбор правильный, нельзя, поэтому анализатор может возвращать несколько вариантов разбора.

У каждого разбора есть нормальная форма, которую можно получить, обратившись к атрибутам normal_form или normalized.

Пример 2. Получение нормальной формы

```
from pymorphy2 import MorphAnalyzer

morph = MorphAnalyzer()
p = morph.parse('стали')[0]
print(p.normal_form) # стать
print(p.normalized) # Parse(word='стать',
tag=OpencorporaTag('INFN,perf,intr'), normal_form='стать',
score=1.0, methods_stack=((<DictionaryAnalyzer>, 'стать',
904, 0),))
```

Кроме того, у каждого разбора есть тег.

Пример 3. Тег

```
from pymorphy2 import MorphAnalyzer

morph = MorphAnalyzer()
p = morph.parse('стали')[0]
print(p.tag) # VERB,perf,intr plur,past,indc
```

Тег – это набор граммем, характеризующих данное слово. Например, тег, представленный в примере, означает, что слово – глагол совершенного вида, непереходный, множественного числа, прошедшего времени, изъявительного наклонения.

pymorphy2 умеет разбирать не только словарные слова; для несловарных слов автоматически задействуется предсказатель. Например, попробуем разобрать слово «бутявковедами» – pymorphy2 поймет, что это форма творительного падежа множественного числа существительного «бутявковед», и что «бутявковед» – одушевленный и мужского рода.

Пример 4. Разбор слова

```
from pymorphy2 import MorphAnalyzer

morph = MorphAnalyzer()
print(morph.parse('бутявковедами')) #
[Parse(word='бутявковедами',
tag=OpencorporaTag('NOUN,anim,masc plur,ablt'),
normal_form='бутявковед', score=1.0,
methods_stack=((<FakeDictionary>, 'бутявковедами', 52, 10),
(<KnownSuffixAnalyzer>, 'едами')))]
```

2.1.2. Работа с тегами

Для того, чтоб проверить, есть ли в данном теге отдельная граммема или все граммемы из указанного множества, можно использовать оператор in.

Пример 5. Проверка, есть ли в данном теге отдельная граммема или все граммемы из указанного множества

```
from pymorphy2 import MorphAnalyzer

morph = MorphAnalyzer()
p = morph.parse('стали')[0]
print('VERB' in p.tag)
print('NOUN' in p.tag)
print({'plut', 'past'} in p.tag)
print({'NOUN', 'plur'} in p.tag)
```

Кроме того, у каждого тега есть атрибуты, через которые можно получить часть речи, число и другие характеристики.

Пример 6. Получение характеристик

```
from pymorphy2 import MorphAnalyzer

morph = MorphAnalyzer()
p = morph.parse('стали')[0]
# часть речи
print(p.tag.POS) # VERB
# одушевленность
print(p.tag.animacy) # None
# вид: совершенный или несовершенный
print(p.tag.aspect) # perf
# падеж
print(p.tag.case) # None
# род (мужской, женский, средний)
print(p.tag.gender) # None
# включенность говорящего в действие
print(p.tag.involvement) # None
# наклонение (повелительное, изъявительное)
print(p.tag.mood) # indc
# число (единственное, множественное)
print(p.tag.number) # plur
# лицо (1, 2, 3)
print(p.tag.person) # None
# время (настоящее, прошедшее, будущее)
print(p.tag.tense) # past
# переходность (переходный, непереходный)
print(p.tag.transitivity) # intr
# залог (действительный, страдательный)
print(p.tag.voice) # None
```

Если запрашиваемая характеристика для данного тега не определена, то возвращается None.

В написании граммем достаточно просто ошибиться; для борьбы с ошибками pymorphy2 выкидывает исключение, если встречает недопустимую граммему.

Пример 7. Недопустимая граммема

```
from pymorphy2 import MorphAnalyzer

morph = MorphAnalyzer()
p = morph.parse('стали')[0]
print('foobar' in p.tag) # ValueError: Grammeme is unknown:
foobar
print({'NOUN', 'foo', 'bar'} in p.tag) # ValueError:
Grammemes are unknown: {'bar', 'foo'}
```

Это работает и для атрибутов.

Пример 8. Недопустимый атрибут

```
from pymorphy2 import MorphAnalyzer

morph = MorphAnalyzer()
p = morph.parse('стали')[0]
print(p.tag.POS == 'plur') # 'plur' is not a valid grammeme
for this attribute
```

2.1.3. Склонение слов

pymorphy2 умеет склонять слова. Чтобы просклонять слово, его нужно сначала разобрать – понять, в какой форме оно стоит в настоящий момент и какая у него лексема.

Пример 9. Разбор слова

```
from pymorphy2 import MorphAnalyzer

morph = MorphAnalyzer()
butyavka = morph.parse('бутявка')[0]
print(butyavka) # Parse(word='бутявка',
tag=OpencorporaTag('NOUN,inan,femn sing,nomn'),
normal_form='бутявка', score=1.0,
methods_stack=((<DictionaryAnalyzer>, 'явка', 8, 0),
(<UnknownPrefixAnalyzer>, 'бут')))
```

Для склонения можно использовать метод Parse.inflect().

Пример 10. Склонение

```
from pymorphy2 import MorphAnalyzer

morph = MorphAnalyzer()
butyavka = morph.parse('бутявка')[0]
# нет кого? (родительный падеж)
```

```
print(butyavka.inflect({'gent'})) # Parse(word='бутявки',
tag=OpencorporaTag('NOUN,inan,femn sing,gent'),
normal_form='бутявка', score=1.0,
methods_stack=((<DictionaryAnalyzer>, 'явки', 8, 1),
(<UnknownPrefixAnalyzer>, 'бут')))
# кого много?
print(butyavka.inflect({'plur', 'gent'})) #
Parse(word='бутявок', tag=OpencorporaTag('NOUN,inan,femn
plur,gent'), normal_form='бутявка', score=1.0,
methods_stack=((<DictionaryAnalyzer>, 'явок', 8, 8),
(<UnknownPrefixAnalyzer>, 'бут')))
```

С помощью атрибута Parse.lexeme можно получить лексему слова.

Пример 11. Лексема слова

```
from pymorphy2 import MorphAnalyzer

morph = MorphAnalyzer()
butyavka = morph.parse('бутявка')[0]
print(butyavka.lexeme) # [
Parse(word='бутявка', tag=OpencorporaTag('NOUN,inan,femn
sing,nomn'), normal_form='бутявка', score=1.0,
methods_stack=((<DictionaryAnalyzer>, 'явка', 8, 0),
(<UnknownPrefixAnalyzer>, 'бут'))),
Parse(word='бутявки', tag=OpencorporaTag('NOUN,inan,femn
sing,gent'), normal_form='бутявка', score=1.0,
methods_stack=((<DictionaryAnalyzer>, 'явки', 8, 1),
(<UnknownPrefixAnalyzer>, 'бут'))),
Parse(word='бутявке', tag=OpencorporaTag('NOUN,inan,femn
sing,datv'), normal_form='бутявка', score=1.0,
methods_stack=((<DictionaryAnalyzer>, 'явке', 8, 2),
(<UnknownPrefixAnalyzer>, 'бут'))),
Parse(word='бутявку', tag=OpencorporaTag('NOUN,inan,femn
sing,accs'), normal_form='бутявка', score=1.0,
methods_stack=((<DictionaryAnalyzer>, 'явку', 8, 3),
(<UnknownPrefixAnalyzer>, 'бут'))),
Parse(word='бутявкой', tag=OpencorporaTag('NOUN,inan,femn
sing,ablt'), normal_form='бутявка', score=1.0,
methods_stack=((<DictionaryAnalyzer>, 'явкой', 8, 4),
(<UnknownPrefixAnalyzer>, 'бут'))),
Parse(word='бутявкою', tag=OpencorporaTag('NOUN,inan,femn
sing,ablt,V-oy'), normal_form='бутявка', score=1.0,
methods_stack=((<DictionaryAnalyzer>, 'явкою', 8, 5),
(<UnknownPrefixAnalyzer>, 'бут'))),
Parse(word='бутявке', tag=OpencorporaTag('NOUN,inan,femn
sing,loct'), normal_form='бутявка', score=1.0,
methods_stack=((<DictionaryAnalyzer>, 'явке', 8, 6),
(<UnknownPrefixAnalyzer>, 'бут'))),
Parse(word='бутявки', tag=OpencorporaTag('NOUN,inan,femn
plur,nomn'), normal_form='бутявка', score=1.0,
methods_stack=((<DictionaryAnalyzer>, 'явки', 8, 7),
(<UnknownPrefixAnalyzer>, 'бут'))),
```

```
Parse(word='бутявок', tag=OpencorporaTag('NOUN,inan,femn
plur,gent'), normal_form='бутявка', score=1.0,
methods_stack=((<DictionaryAnalyzer>, 'явок', 8, 8),
(<UnknownPrefixAnalyzer>, 'бут'))),
Parse(word='бутявкам', tag=OpencorporaTag('NOUN,inan,femn
plur,datv'), normal_form='бутявка', score=1.0,
methods_stack=((<DictionaryAnalyzer>, 'явкам', 8, 9),
(<UnknownPrefixAnalyzer>, 'бут'))),
Parse(word='бутявки', tag=OpencorporaTag('NOUN,inan,femn
plur,accs'), normal_form='бутявка', score=1.0,
methods_stack=((<DictionaryAnalyzer>, 'явки', 8, 10),
(<UnknownPrefixAnalyzer>, 'бут'))),
Parse(word='бутявками', tag=OpencorporaTag('NOUN,inan,femn
plur,ablt'), normal_form='бутявка', score=1.0,
methods_stack=((<DictionaryAnalyzer>, 'явками', 8, 11),
(<UnknownPrefixAnalyzer>, 'бут'))),
Parse(word='бутявках', tag=OpencorporaTag('NOUN,inan,femn
plur,loct'), normal_form='бутявка', score=1.0,
methods_stack=((<DictionaryAnalyzer>, 'явках', 8, 12),
(<UnknownPrefixAnalyzer>, 'бут')))
]
```

2.1.4. Постановка слов в начальную форму

Как уже было написано, нормальную форму слова можно получить через атрибуты Parse.normal_form и Parse.normalized.

Но что считается за нормальную форму? Например, возьмем слово «думающим». Иногда мы захотим нормализовать его в «думать», иногда – в «думающий», иногда – в «думающая».

Пример 12. Нормальная форма

```
from pymorphy2 import MorphAnalyzer

m = MorphAnalyzer()
print(m.parse('думающему')[0].normal_form)
```

pymorphy2 сейчас использует алгоритм нахождения нормальной формы, который работает наиболее быстро (берется первая форма в лексеме) – поэтому, например, все причастия сейчас нормализуются в инфинитивы. Это можно считать деталью реализации.

Если требуется нормализовывать слова иначе, можно воспользоваться методом Parse.inflect().

Пример 13. Склонение

```
from pymorphy2 import MorphAnalyzer

m = MorphAnalyzer()
print(m.parse('думающему')[0].inflect({'sing',
'nomn'}).word) # думающий
```

2.1.5. Согласование слов с числительными

Слово нужно ставить в разные формы в зависимости от числительного, к которому оно относится. Например: «1 бутявка», «2 бутявки», «5 бутявок»

Для этих целей можно использовать метод Parse.make_agree_with_number().

Пример 14. Формы в зависимости от числительного

```
print(butyavka.make_agree_with_number(1).word) # бутявка
print(butyavka.make_agree_with_number(2).word) # бутявки
print(butyavka.make_agree_with_number(5).word) # бутявок
```

2.2. Задания к лабораторной работе №2

1. Выполните примеры 1-14.
2. Разработайте морфологический анализатор.
3. Сформируйте отчет с текстом программы и скринами результатов.

Требования к отчету к ЛР №2

1. Листинг к заданию 2.
2. Краткое описание пакетов и функций.
3. Готовность ответить на вопросы по примерам из текста ЛР и домашних заданий.

ЛАБОРАТОРНАЯ РАБОТА №3

Синтаксический анализ

Цель. Освоить основы работы с Natasha. Посмотреть возможности Natasha.

3.1. Natasha

Natasha – это аналог Томита-парсера для Python, набор готовых правил для извлечения имён, адресов, дат, сумм денег и других сущностей.

Раньше библиотека Natasha решала задачу NER для русского языка, была построена на правилах, показывала среднее качество и производительность. Сейчас Natasha – это целый большой проект, состоит из 9 репозиториев. Библиотека Natasha объединяет их под одним интерфейсом, решает базовые задачи обработки естественного русского языка: сегментация на токены и предложения, предобученные эмбеддинги, анализ морфологии и синтаксиса, лемматизация, NER. Все решения показывают топовые результаты в новостной тематике, быстро работают на CPU.

Natasha похожа на другие библиотеки-комбайны: SpaCy, UDPipe, Stanza. SpaCy инициализирует и вызывает модели неявно.

Интерфейс Natasha более многословный. Пользователь явно инициализирует компоненты: загружает предобученные эмбеддинги, передаёт их в конструкторы моделей. Сам вызывает методы segment, tag_morph, parse_syntax для сегментации на токены и предложения, анализа морфологии и синтаксиса.

Пример 1. Синтаксический анализ предложения

```
>>> from natasha import (
    Segmenter,

    NewsEmbedding,
    NewsMorphTagger,
    NewsSyntaxParser,

    Doc
)

>>> segmenter = Segmenter()
```

```
>>> emb = NewsEmbedding()
>>> morph_tagger = NewsMorphTagger(emb)
>>> syntax_parser = NewsSyntaxParser(emb)

>>> text = 'международное признание образовательных программ
российских вузов'
>>> doc = Doc(text)

>>> doc.segment(segmenter)
>>> doc.tag_morph(morph_tagger)
>>> doc.parse_syntax(syntax_parser)

>>> print(doc.tokens[:5])
[DocToken(stop=13, text='международное', id='1_1',
head_id='1_2', rel='amod', pos='ADJ',
feats=<Inan,Acc,Pos,Neut,Sing>), DocToken(start=14, stop=23,
text='признание', id='1_2', head_id='1_0', rel='root',
pos='NOUN', feats=<Inan,Nom,Neut,Sing>), DocToken(start=24,
stop=39, text='образовательных', id='1_3', head_id='1_4',
rel='amod', pos='ADJ', feats=<Gen,Pos,Plur>), DocToken(start=40,
stop=48, text='программ', id='1_4', head_id='1_2', rel='nmod',
pos='NOUN', feats=<Inan,Gen,Fem,Plur>), DocToken(start=49,
stop=59, text='российских', id='1_5', head_id='1_6', rel='amod',
pos='ADJ', feats=<Gen,Pos,Plur>)]
>>> doc.sents[0].syntax.print()
      ┌─► международное   amod
   ┌──└── признание
   │  ┌─► образовательных amod
┌──└─►└── программ         nmod
│     ┌─► российских       amod
└────►└── вузов            nmod
```

3.2. Задания к лабораторной работе №3

1. Поэкспериментируйте с примером.
2. Разработайте синтаксический анализатор.
3. Сформируйте отчет с текстом программы и скринами результатов.

Требования к отчету к ЛР №3

1. Листинг к заданию 2.
2. Краткое описание пакетов и функций.
3. Готовность ответить на вопросы по примерам из текста ЛР и домашних заданий.

ЛАБОРАТОРНАЯ РАБОТА №4

Частотный критерий семантической близости

Цель. Изучить и запрограммировать сравнение текстовых фрагментов на семантическую близость на основе частотного критерия.

4.1. Теоретические сведения

Пусть дан текстовый фрагмент $q = a_1 a_2 \ldots a_n$, состоящий из слов естественного языка. Данный образец сравним на семантическую близость с текстовым фрагментом (ТФ) t вида $t = b_1 b_2 \ldots b_m$. Введем критерий семантической близости (КСБ) следующим образом:

$$Cprox\left(S(q),\ S(t)\right) \in [0,\ 1]. \quad (1)$$

Представим КСБ в виде:

$$Cprox\left(S(q),\ S(t)\right) = \frac{p}{n}, \quad (2)$$

где n представляет число слов в ТФ q, а p – число слов из q, совпадающих со словами в t. Обозначим такую меру семантической близости как φ.

Пример 1. Сравнение текстовых фрагментов на семантическую близость

Пусть $q =$«международное признание образовательных программ российских вузов». Рассмотрим сравнение этого фрагмента с некоторыми другими.

$t_1 =$«правительство России проводит политику, направленную на международное признание образовательных программ российских вузов». В этом фрагменте присутствуют все слова из q, поэтому в данном случае $\varphi = 1$.

$t_2 =$«образовательные программы российских вузов нуждаются в международном признании». В этом фрагменте также присутствуют все слова из q, поэтому $\varphi = 1$.

$t_3 =$«международное признание вузов увеличивает возможности России по привлечению иностранных студентов». Здесь присутствует 3 из 6 слова из q, поэтому $\varphi = 0{,}5$.

4.2. Выполнение работы

1. Установим интерпретатор языка Python, а также библиотеки NLTK и pymorphy2, если это ещё не сделано.

2. Рассмотрим некоторые функции пакета pymorphy2, которые понадобятся при выполнении работы.

Введём следующие команды:

```
import pymorphy2 as pm
m = pm.MorphAnalyzer()
m.parse('международное')
```

Первая команда импортирует пакет pymorphy2, вторая создаёт экземпляр морфологического анализатора, третья осуществляет разбор слова «международное». При этом интерпретатор выдаст информацию об этом слове, в том числе морфологические признаки и начальную форму. Если существует несколько вариантов разбора, они все будут предоставлены в порядке убывания вероятности.

3. Функция word_tokenize() пакета NLTK осуществляет токенизацию фрагмента текста. Пример:

```
from nltk import word_tokenize

sent = "международное признание образовательных программ российских вузов"
words = word_tokenize(sent)
print(words) # ['международное', 'признание', 'образовательных', 'программ', 'российских', 'вузов']
```

4. Чтобы получить список слов фрагмента текста в начальной форме, можно использовать следующую конструкцию:

```
sent = "международное признание образовательных программ российских вузов"
words = [m.parse(word)[0].normal_form for word in word_tokenize(sent)]
print(words) # ['международный', 'признание', 'образовательный', 'программа', 'российский', 'вуз']
```

При этом из всех вариантов начальной формы слова выбирается наиболее вероятный.

5. Объединив всё изложенное выше напишем следующую программу, осуществляющую сравнение фрагментов на семантическую близость по частотному критерию:

```
import pymorphy2 as pm
from nltk import word_tokenize

def freq_semantic(fragment_1, fragment_2):
    m = pm.MorphAnalyzer()
    words_1 = set([m.parse(word)[0].normal_form for word in word_tokenize(fragment_1)])
    words_2 = set([m.parse(word)[0].normal_form for word in word_tokenize(fragment_2)])
    return len(words_1 & words_2) / len(words_1)

q = input("Первый фрагмент: ")
t = input("Второй фрагмент: ")
print(freq_semantic(q, t))
```

Ниже приведены результаты сравнения фрагмента q с фрагментами t_2 и t_3.

```
Первый фрагмент: международное признание образовательных программ российских вузов
Второй фрагмент: образовательные программы российских вузов нуждаются в международном признании
1.0

Первый фрагмент: международное признание образовательных программ российских вузов
Второй фрагмент: международное признание вузов увеличивает возможности России по привлечению иностранных студентов
0.5
```

4.3. Задания к лабораторной работе №4

1. Сравнить фрагмент q = «проблема автоматической обработки текстов» со следующим фрагментом вручную и с помощью программы:

Вариант	Фрагмент
1	в настоящее время проблема автоматической обработки текстов становится актуальной
2	компьютерная лингвистика изучает проблемы автоматической обработки текстов
3	работа посвящена методам автоматической обработки текстов
4	автоматическая обработка текстов помогает при большом объёме данных
5	автоматическая обработка текстов считается сложной задачей
6	автоматическая обработка текстов и документов
7	проблема автоматической обработки изображений иногда решается с использованием нейронных сетей
8	обработка текстов на естественном языке является сложной задачей
9	обработка больших текстов человеком является трудоёмким занятием
10	иногда школьники считают чтение больших текстов проблемой
11	обработка деталей при автоматической сборке требует тонкой настройки
12	текст посвящён проблеме самосознания личности

Вариант: (№ п/п % 12) + 1.

2. Переписать программу так, чтобы после введения запроса q ввод фрагментов осуществлялся до тех пор, пока не будет введена пустая строка. Фрагменты отделяются друг от друга переносом строки и сохраняются в список. Допускается также вычисление семантической близости по мере ввода фрагментов без их сохранения.

3. Оформить отчёт о проделанной работе. В отчёте обязательно должны быть скриншоты входных и выходных данных.

Требования к отчету к ЛР №4

1. Листинг к заданиям 1-2.
2. Краткое описание пакетов и функций.
3. Готовность ответить на вопросы по примерам из текста ЛР и домашних заданий.

4.4. Используемые источники

1. Вишняков Ю.М., Вишняков Р.Ю. Вычислительная теория семантической интерпретации текстов научно-технического стиля естественного языка // Сборник трудов Международной научно-технической конференции «Актуальные проблемы прикладной математики, информатики и механики», Воронеж, 18–20 декабря 2017 г. – Воронеж: Издательство «Научно-исследовательские публикации», 2017. С. 219–226.

ЛАБОРАТОРНАЯ РАБОТА №5

Семантическая близость в рамках технологии Word2Vec

Цель. Изучить технологию Word2Vec и опробовать её на различных примерах.

5.1. Теоретические сведения

Word2Vec – технология, разработанная Google, для нахождения сематнических связей между словами.

Word2Vec включает в себя набор алгоритмов для расчета векторных представлений слов, предполагая, что слова, используемые в похожих контекстах, значат похожие вещи, т.е. семантически близки.

Технология Word2Vec использует два разных метода:

1. CBOW – предсказание слова на основании близлежащих слов.
2. Skip-gram – предсказание близлежащих слов на основании одного слова.

Метод представления слов в виде векторов используется для кластеризация слов и выявления их семантической близости, т.е. разделяет несвязанные слова и соединяет связанные, что помогает в задачах кластеризации и классификации текстов.

Принцип работы:

1) читается корпус, и рассчитывается встречаемость каждого слова в корпусе;
2) массив слов сортируется по частоте и удаляются редкие слова;
3) строится дерево Хаффмана (для кодирования словаря – это значительно снижает вычислительную и временную сложность алгоритма);
4) из корпуса читается т.н. субпредложение (базовый элемент корпуса – предложение, абзац, статья) и проводится субсэмплирование (процесс изъятия наиболее частотных слов) из анализа;
5) по субпредложению проходим окном (максимальная дистанция между текущим и предсказываемым словом в предложении);
6) применяется нейросеть прямого распространения с функцией активации иерархический софтмакс и/или негативное сэмплирование.

Во время обучения алгоритм формирует оптимальный вектор для каждого слова с помощью CBOW или skip-gram.

5.2. Выполнение работы

Для работы с Word2Vec необходимо установить библиотеку gensim.

Модуль genism.downloader позволит загружать готовые модели для работы. Например, следующий код выведет 10 слов, наиболее сходных к слову «текст» и степень сходства:

```
import gensim
import gensim.downloader as api
import string
import pymorphy2

model = api.load("word2vec-ruscorpora-300")
result = model.most_similar("текст_NOUN")[:10]
```

```
for item in result:
    print(item)
```

Результат выполнения:

```
('редакторский::правка_NOUN', 0.6438080072402954)
('цитата_NOUN', 0.5979090929031372)
('орфоэпический_ADJ', 0.5947705507278442)
('rtf_NOUN', 0.5915514230728149)
('зачеркивание_NOUN', 0.5874521136283875)
('инфп_NOUN', 0.5868774056434631)
('текстуальный_ADJ', 0.585808515548706)
('отрывок_NOUN', 0.5752384066581726)
('подлинник_NOUN', 0.5750516057014465)
('курсивный_ADJ', 0.574798047542572)
```

Обратите внимание, что слова в используемом корпусе записываются вместе с частью речи через подчёркивание. Чтобы не записывать часть речи вручную, можно, например, воспользоваться модулем pymorphy2.

Можно искать слова, наиболее близкие к нескольким словам:

```
result = model.most_similar(positive=["образовательный_ADJ",
"программа_NOUN"], topn=5)
for item in result:
    print(item)
```

Однако с увеличением количества слов точность такого метода существенно уменьшается.

Другая функция n_similarity(ws1, ws2) находит косинусное сходство между наборами слов. Однако в этом случае текст представляется как набор входящих в него слов, связи между словами не учитываются. ws1 и ws2 – списки слов.

Пример для фрагментов q = «международное признание образовательных программ российских вузов» и t_1 = «образовательные программы российских вузов нуждаются в международном признании»:

```
print(sent_1)
print(sent_2)

print(model.n_similarity(sent_1, sent_2))
```

Вывод программы:

```
['международный_ADJ', 'признание_NOUN',
'образовательный_ADJ', 'программа_NOUN', 'российский_ADJ',
'вуз_NOUN']
['образовательный_ADJ', 'программа_NOUN', 'российский_ADJ',
'вуз_NOUN', 'нуждаться_VERB', 'международный_ADJ',
'признание_NOUN']
0.9765474
```

При использовании этого метода слова должны быть приведены к начальной форме, а через подчёркивание необходимо записать часть речи. Всё это можно сделать с помощью pymorphy2, однако следует отметить, что обозначения частей речи в модели и в pymorphy2 могут отличаться (например, pymorphy2 отмечает полные прилагательные как ADJF, в то время как в используемой модели Word2Vec прилагательные обозначаются как ADJ).

Также предварительно необходимо очистить текст от незначащих слов (например, предлогов и союзов), поскольку их может не быть в модели, и программа выдаст ошибку. Очистить текст можно вручную или воспользоваться набором стоп-слов NLTK:

```
from nltk.corpus import stopwords
...
sent = [word for word in sent if word not in
stopwords.words('russian')]
```

5.3. Задания к лабораторной работе №5

1. Для нескольких слов на выбор найдите наиболее близкие по смыслу слова согласно модели Word2Vec. Среди слов должно быть существительное, прилагательное и глагол.

2. Сравнить фрагмент q = «проблема автоматической обработки текстов» со следующими фрагментами с помощью программы:

Вариант	Фрагмент
1	в настоящее время проблема автоматической обработки текстов становится актуальной
2	компьютерная лингвистика изучает проблемы автоматической обработки текстов
3	работа посвящена методам автоматической обработки текстов
4	автоматическая обработка текстов помогает при большом объёме данных
5	автоматическая обработка текстов считается сложной задачей
6	автоматическая обработка текстов и документов
7	проблема автоматической обработки изображений иногда решается с использованием нейронных сетей
8	обработка текстов на естественном языке является сложной задачей
9	обработка больших текстов человеком является трудоёмким занятием
10	иногда школьники считают чтение больших текстов проблемой
11	обработка деталей при автоматической сборке требует тонкой настройки
12	текст посвящён проблеме самосознания личности

Вариант: (№ п/п % 12) + 1.

3. Дополнительное задание: написать функцию, на вход которой подаётся текст. Функция должны преобразовать текст в формат, пригодный для использования в функции n_similarity(ws1, ws2). Использовать токенизацию, очистку от стоп-слов, приведение к начальной форме и определение части речи (обратить внимание на различие обозначений и обработать эти особенности).

4. Оформить отчёт о проделанной работе.

Требования к отчету к ЛР №5

1. Листинг к заданиям 1-3.
2. Краткое описание пакетов и функций.
3. Готовность ответить на вопросы по примерам из текста ЛР и домашних заданий.

5.4. Используемые источники

1. Word2vec в картинках / Хабр: [Электронный ресурс]. URL: https://habr.com/ru/post/446530/ (дата обращения: 19 сентября 2021).

2. Word2vec обработка текстов от Google - MegaIndex.: [Электронный ресурс]. URL: https://ru.megaindex.com/support/faq/word2vec (дата обращения: 19 сентября 2021).

3. gensim.models.Word2Vec: [Электронный ресурс]. URL: https://tedboy.github.io/nlps/generated/generated/gensim.models.Word2Vec.html (дата обращения: 19 сентября 2021).

ЛАБОРАТОРНАЯ РАБОТА №6

Семантическая близость в рамках вычислительной теории семантической интерпретации

Цель. Изучить основные положения вычислительной теории семантической интерпретации, опробовать положения этой теории на примерах.

6.1. Теоретические сведения

В рамках вычислительной теории семантической интерпретации смысл текстового фрагмента определяется через функционал над смысловыми значениями входящих в него слов, а определение смысла всего фрагмента сводится к построению и выполнению вычислительной процедуры над смыслами отдельных слов.

Основные обозначения:

1) $S(\alpha)$ – множество смысловых значений слова α;

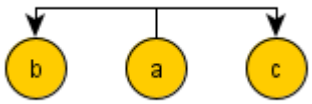
2) $S(\alpha)=\Phi\left(S(a_1),\ S(a_2),\ \ldots,\ S(a_n)\right)$, где $\alpha=a_1a_2\ldots a_n$ – текстовый фрагмент, Φ – функционал смысла.

Операции:

1. $\vec{\cap}$ – операция контекстного уточнения смысла. Так, запись $S(a)\vec{\cap}S(b)$ означает, что слово b контекстно уточняет смысл слова a.

2. $\cap$ – операция пересечения смыслов.

Пример текстового фрагмента:



И его функционала смысла:

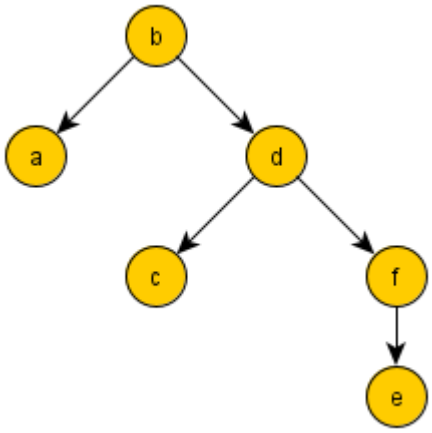
$$\left(S(a)\vec{\cap}S(b)\right)\cap\left(S(a)\vec{\cap}S(c)\right).$$

При вычислениях удобно использовать обратную польскую запись функционала смысла. ОПЗ не содержит скобок, и ее вычисления проводится за один проход слева направо. Операции выполняются по мере их встречаемости в процессе прохода выражения. Поэтому структура ОПЗ однозначно определяет структуру вычислительной процедуры. Графическое представление ОПЗ называется семантической схемой.

Пусть дан фрагмент:

$$q=\text{"}\underbrace{\text{международное}}_{a}\ \underbrace{\text{признание}}_{b}\ \underbrace{\text{образовательных}}_{c}\ \underbrace{\text{программ}}_{d}\ \underbrace{\text{российских}}_{e}\ \underbrace{\text{вузов}}_{f}\text{"}.$$

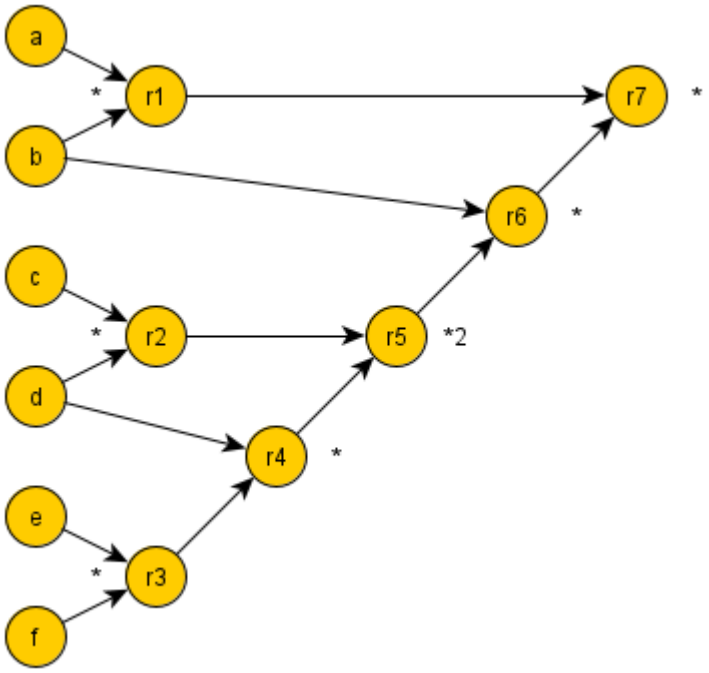
Обозначим буквами a, b, …, f слова этого фрагмента в том порядке, в каком они следуют в фрагменте. Тогда дерево зависимостей выглядит следующим образом:



ОПЗ этого фрагмента:

$$S(a)S(b)\overleftarrow{\frown}S(c)S(d)\overleftarrow{\frown}S(e)S(f)\overleftarrow{\frown}S(d)\overleftarrow{\frown}\frown S(b)\overleftarrow{\frown}\frown$$

Семантическая схема:

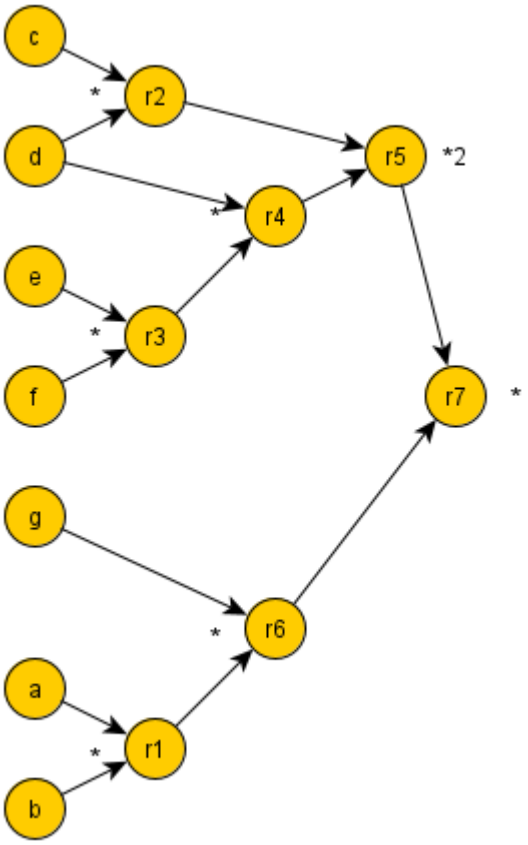


При этом операция контекстного уточнения смысла на семантической схеме обозначается *, а операция пересечения смыслов – $*n$, где n – количество смыслов, пересечение которых необходимо найти. r_1, r_2, …, r_7 в данном случае – это фрагменты, на основе которых осуществляется сравнение на семантическую близость.

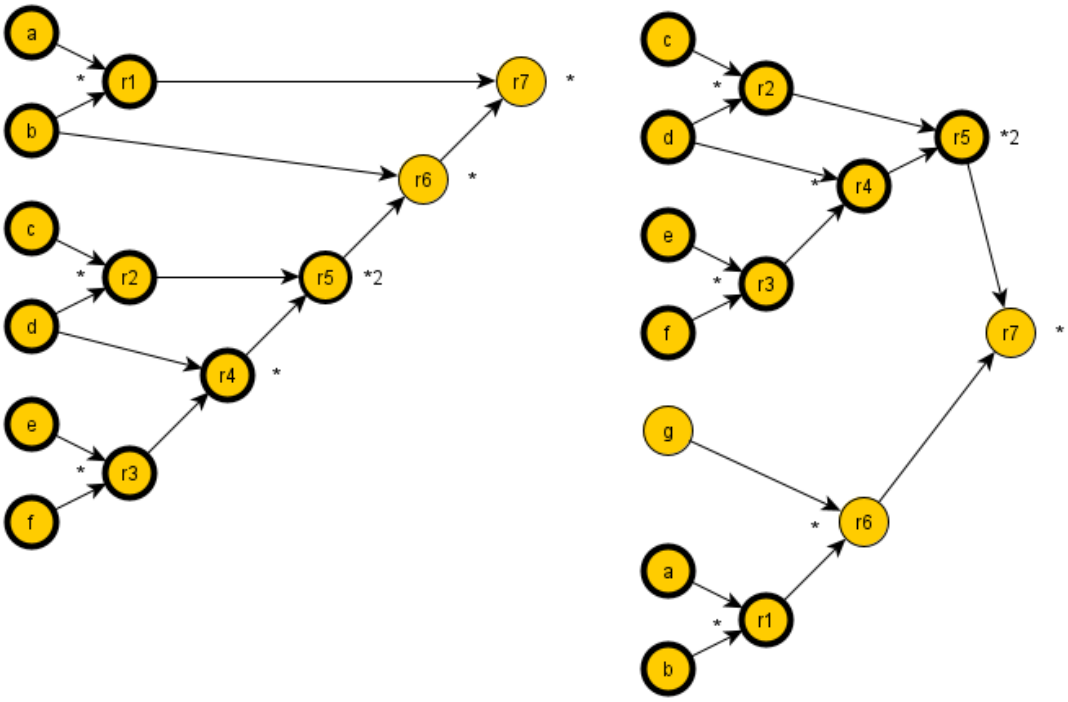
Пример сравнения:

Пусть дан фрагмент t_2 = «образовательные программы российских вузов нуждаются в международном признании».

Семантическая схема этого фрагмента выглядит следующим образом:



Общие фрагменты выделены ниже:



Как это следует из рисунка, первая семантическая схема содержит 7 узлов $r_1, r_2, \ldots, r_7$, пять из которых присутствуют во второй семантической схеме, образуя в обеих схемах одинаковые поддеревья. Поэтому семантическая близость в данном случае равна $\frac{5}{7}$.

6.2. Выполнение работы

Пример 1. Реализация алгоритма нахождения семантической близости в рамках вычислительной теории семантической интерпретации

```
from natasha import (
    Segmenter,

    NewsEmbedding,
    NewsMorphTagger,
    NewsSyntaxParser,

    Doc
)
import pymorphy2 as pm

def norm(txt):
    _, x = map(int, txt.split('_'))
    return x

def sent_to_dicts(sent):
    '''
   Преобразование предложения (формат CoNLL) в словари:
   1) словарь words - слова (ключ - id, значение - слово);
   2) словарь tree - дерево (ключ - id родителя, значение - список id сыновей).
    '''
    words = dict()
    for token in sent.tokens:
        if token.pos != 'PUNCT':
            norm_id = norm(token.id)
            words[norm_id] = token.text
            if token.rel == 'root':
                root = norm_id
    tree = {0: []}
    for k in words.keys():
        tree[k] = []
    for token in sent.tokens:
        if token.pos != 'PUNCT':
            norm_id = norm(token.id)
            norm_head_id = norm(token.head_id)
            tree[norm_head_id].append(norm_id)
    return words, root, tree

def normal_form(word):
    m = pm.MorphAnalyzer()
    return m.parse(word)[0].normal_form
```

```
def dicts_to_rpn(words, root, tree):
    '''
   Преобразование словарей в ОПЗ (с помощью нерекурсивного
DFS).
   Список rpn - ОПЗ.
   '''
    if len(words) <= 1:
        return list(words.values())
    rpn = []
    stack = [(0, root)]
    colors = {k: 0 for k in words.keys()}
    while len(stack) > 0:
        pred, curr = stack[len(stack) - 1]
        if colors[curr] == 0:
            colors[curr] = 1
        elif colors[curr] == 1:
            if len(tree[curr]) > 1:
                rpn.extend(['*', len(tree[curr])])
            if curr != root:
                if len(tree[curr]) == 0:
                    rpn.extend([words[curr], words[pred],
'*'])
                elif len(tree[curr]) > 0:
                    rpn.extend([words[pred], '*'])
            stack.pop()
        for nxt in reversed(tree[curr]):
            if colors[nxt] == 0:
                stack.append((curr, nxt))
    return rpn

def semantic_scheme(rpn):
    '''
   Преобразование ОПЗ в семантическую схему.
   Список ss - схемантическая схема.
   '''
    if len(rpn) <= 1:
        return rpn
    ss = []
    i = 0
    stack = []
    while i < len(rpn):
        if rpn[i] == '*':
            if i + 1 < len(rpn) and isinstance(rpn[i + 1],
int):
                n = rpn[i + 1]
                m = 2
                rpn[i + 1] = str(rpn[i + 1])
```

```
                else:
                    n = 2
                    m = 1
                r = ' '.join(rpn[i - n:i + m])
                rpn[i - n:i + m] = [r]
                ss.append(r)
                i -= n
            i += 1
        return ss

def sent_to_ss(sent):
    '''
    Преобразование предложения (формат CoNLL) в сем. сх.
    '''
    words, root, tree = sent_to_dicts(sent)
    for k in words.keys():
        words[k] = normal_form(words[k])
    rpn = dicts_to_rpn(words, root, tree)
    ss = semantic_scheme(rpn)
    return ss

def compare(ss_q, ss_t):
    '''
    Сравнение сем. сх.
    cprox - семантическая близость.
    '''
    ss_q = set(ss_q)
    ss_t = set(ss_t)
    p = len(ss_q & ss_t)
    m = len(ss_q)
    cprox = p / m
    return cprox

segmenter = Segmenter()

emb = NewsEmbedding()
morph_tagger = NewsMorphTagger(emb)
syntax_parser = NewsSyntaxParser(emb)

q = 'международное признание образовательных программ
российских вузов'
doc = Doc(q)

doc.segment(segmenter)
doc.tag_morph(morph_tagger)
doc.parse_syntax(syntax_parser)

sent = doc.sents[0]

ss_q = sent_to_ss(sent)

t = 'международное признание образовательных программ
поднимает репутационный рейтинг российских вузов'
```

```
doc = Doc(t)

doc.segment(segmenter)
doc.tag_morph(morph_tagger)
doc.parse_syntax(syntax_parser)

sent = doc.sents[0]

ss_t = sent_to_ss(sent)

cprox = compare(ss_q, ss_t)

print('q = {:s}'.format(q))
print('t = {:s}'.format(t))
print('Семантическая близость = {:.0%}'.format(cprox))
```

Вывод программы:

```
q = международное признание образовательных программ российских вузов
t = международное признание образовательных программ поднимает репутационный рейтинг российских вузов
Семантическая близость = 43%
```

6.3. Задания к лабораторной работе №6

1. Даны фрагменты текста: q = «проблема автоматической обработки текстов».

Вариант	Фрагмент
1	t_1 = «в настоящее время проблема автоматической обработки текстов становится актуальной»
2	t_2 = «проблема обработки текстов становится актуальной»
3	t_3 = «школьники иногда считают чтение больших текстов проблемой»
4	t_4 = «некоторые люди считают психологические проблемы выдуманными»
5	t_5 = «проблема автоматической обработки изображений иногда решается с использованием нейронных сетей»
6	t_6 = «обработка текстов на естественном языке является сложной задачей»
7	t_7 = «обработка больших текстов человеком является трудоёмким занятием»

Вариант: (№ п/п % 7) + 1.

Необходимо построить ОПЗ этих фрагментов. Допускается использование как предложенного инструмента, так и самостоятельная реализация. В отчёте необходимо описать алгоритм.

2. Сравнить фрагмент q с фрагментом t_i на семантическую близость в рамках вычислительной теории семантической интерпретации.

3. Оформить отчёт по проделанной работе.

Требования к отчету к ЛР №6

1. Листинг к заданиям 1-2.
2. Краткое описание пакетов и функций.
3. Готовность ответить на вопросы по примерам из текста ЛР и домашних заданий.

6.4. Используемые источники

1. Вишняков Ю.М., Вишняков Р.Ю. Вычислительная теория семантической интерпретации текстов научно-технического стиля естественного языка // Сборник трудов Международной научно-технической конференции «Актуальные проблемы прикладной математики, информатики и механики», Воронеж, 18–20 декабря 2017 г. – Воронеж: Издательство «Научно-исследовательские публикации», 2017. С. 219–226.

2. Вишняков Ю.М., Степанова Е.В. «Программная реализация сравнения текстовых отрезков на семантическую близость» – Прикладная математика: современные проблемы математики, информатики и моделирования: материалы II Всероссийской научно-практической конференции, г. Краснодар, 22–25 апреля 2020 г., с 201-207.