Лабораторная работа 3. Обработка естественного языка

Natural Language Processing (далее – NLP) – обработка естественного языка – подраздел информатики и искусственного интеллекта (AI, artificial intelligence), посвященный тому, как компьютеры анализируют естественные (человеческие) языки. NLP позволяет применять алгоритмы машинного обучения для текста и речи. Наиболее известными примерами являются голосовые помощники (Cortana, Siri), фильтрация спама, чат-боты и др.

В целом задачи NLP условно разделяют на 3 группы:

- синтаксические задачи с четко определенными условиями и критериями результата (разметка по частям речи, морфологическая сегментация, стемминг и лемматизация, выделение границ предложения, разрешение омонимических случаев и др.);
- семантические задачи с четко определенной задачей и критериями правильного ответа (информационный поиск, анализ тональности, выделение отношений и фактов, ответы на вопросы, языковое модели как средство предсказания следующего слова или символа и др.);
- семантические задачи, требующие не только понимания, но и порождения текста (порождение текста, автоматическое реферирование, машинный перевод, диалоговые модели и др.).

Получение вектора признаков.

Одним из первых (и, возможно, главных) вопросов, возникающих при анализе текстовой информации, — каким образом преобразовать данные для подачи на вход модели принятия решений. Процесс преобразования информации к такому виду называется «извлечение признаков».

Одним из простейших вариантов является «one-hot» представление. Каждое слово в словаре представляется в виде вектора, размер которого равен числу слов в словаре. При этом все элементы вектора, кроме одного, равны нулю, а элемент в позиции, соответствующей номеру слова в словаре, равен единице. Недостатками такого подхода являются большая размерность данных и отсутствие семантической и морфологической зависимостей между словами. Тем не менее, он достаточно успешно применяется для анализа текстов с небольшим объемом словаря с соответствующей предварительной обработкой данных.

Более сложным подходом к преобразованию слов в векторный вид является построение распределенного представления слов. При этом каждому слову сопоставляется вектор из вещественных чисел, являющийся элементом евклидова пространства. Эти векторы далее служат входами последующих

моделей, и базовое предположение состоит в том, что геометрические соотношения в этом пространстве будут соответствовать семантическим соотношениям между словами. Например, ближайшие соседи сло́ва по евклидову расстоянию окажутся его синонимами или другими тесно связанными на некотором основании словами.

Подобные распределения слов и их соответствующие векторные представления могут быть построены на основе различных подходов и принципов, имеющих общее название «word embedding». Одним из классических подходов являются модели word2vec.

Вектор признаков, подаваемый на вход модели, может быть построен не только на описании самих слов, но и на описании более сложных составных структур, таких как предложения, тексты или документы. Наиболее простой и популярной моделью формирования такого представления является «мешок слов» («bag of words»).

Мешок слов — это модель текстов на естественном языке, в которой каждый документ или текст выглядит как неупорядоченный набор слов без сведений о связях между ними. Его можно представить в виде матрицы, каждая строка в которой соответствует отдельному документу или тексту, а каждый столбец — определенному слову. Ячейка на пересечении строки и столбца содержит количество вхождений слова в соответствующий документ. Таким образом, получается вектор признаков для каждого документа по словарю, его можно получить также просуммировав все «one-hot» представления, для каждого слова, встречающегося в тексте.

Работа с текстом в Python.

Как и во многих других языках в Python для работы с текстом есть отдельный тип данных «string». Одним из наиболее популярных средств в Python для более высокоуровневой обработки текста является библиотека NLTK (Natural Language Toolkit, https://www.nltk.org/). Чтобы ее установить, введите в командную строку «`pip install nltk`».

Для добавления nltk в свой код, как и всегда, необходимо ее проимпортировать: `import nltk`.

Одной из первых задач при работе с текстом является его разделение на отдельные лексически значимые части, что обычно называют «токенизацей». Токенизацию можно делать по предложениям, словам или на основании каких-то более сложных принципов. Несмотря на то, что на первый взгляд задача токенизации не выглядит сложной, в процессе анализа возникает достаточно много дополнительных трудностей, вызванных наличием точек не только в конце предложения, а еще и в сокращениях, использование заглавных символов для имен собственных, использование дефисов и многое другое.

Функции для токенизации по предложениям и словам в библиотеке scikit-learn: `sent_tokenize(text)` и `word_tokenize(sentence)`.

Кроме разделения текста на отдельные структурные единицы полезным также будет, в частности, в случае разделения на слова, привести все к нижнему регистру (функция `text.lower()`) и опустить имеющуюся в тексте пунктуацию (метод `text.translate` с параметром `str.maketrans('', '', string.punctuation)`).

Также при анализе текста и его разделении на отдельные слова попадается достаточно много форм одного слова и однокоренных слов, что может быть проблемой при использовании моделей принятия решений, основанных на частоте встречания слов. Данные задачи называются лемматизацией и стеммингом текста.

Лемма – это словарная форма слова. Процессы лемматизации и стемминга отличаются принципом работы и нахождения лемм. При стемминге убираются словообразовательные суффиксы и изменяются окончания, при этом при лемматизации использует словарь и морфологический анализ. Соответственно, разница заключается в словах, которые полностью изменяют написание при изменении формы или имеют чередование букв в корне (good и better). Функции для лемматизации и стемминга в nltk: WordNetLemmatizer(), PorterStemmer().

В любом тексте встречаются слова, такие как артикли, междометия, союзы и т.д., которые не несут смысловой нагрузки, и, соответственно, будут вносить больше шума при построении моделей, чем улучшать ее качество. В NLTK есть предустановленный список таких слов, называемых стоп-словами. Перед первым использованием вам понадобится его скачать, введите в командную строку:

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

После скачивания можно импортировать пакет stopwords (`from nltk.corpus import stopwords`) и посмотреть на сами слова:

```
print(stopwords.words('english'))
```

Пример того, как можно убрать стоп-слова:

```
stop_words = set(stopwords.words('english'))
```

```
    sentence = ‘Once you know all the elements, it's not difficult
to pull together a sentence.’
    words = nltk.word_tokenize(sentence)
    without_stop_words = [word for word in words if not word in
stop_words]
```

Пример функции по обработке текста с удалением стоп-слов и знаков препинания:

```
    def text_process(text):
        text     =     text.translate(str.maketrans('',     '',
string.punctuation))
        text = [word for word in text.split() if word.lower() not
in stopwords.words('english')]
        return " ".join(text)
```

Вызвать данную функцию на каждом элементе набора текстов `text_feat` можно с помощью функции `apply`:

```
    text_feat = text_feat.apply(text_process)
```

Чтобы перейти к созданию вектора признаков с помощью «мешка слов», необходимо:

```
    from sklearn.feature_extraction.text import CountVectorizer
    vectorizer = CountVectorizer()
    features = vectorizer.fit_transform(texts)
```

Посмотреть, что получилось в итоге можно следующим образом:

```
    feature_names = vectorizer.get_feature_names()
    pd.DataFrame(features.toarray(), columns = feature_names)
```

При увеличении размера текстов увеличивается также и словарь, что в целом может оказывать негативное влияние на модель. Одним из недостатков является то, что итоговая матрица получается очень разреженной. Основные методы борьбы с этой проблемой заключаются в предварительной обработке слов (приведение к одному регистру, исключение стоп-слов, лемматизация и т.д.), а также в использовании N-грамм (последовательностей N слов вместо отдельных слов).

Кроме подсчета анализа на встречаемость каждого слова в документах (0 или 1), можно подсчитать также количество вхождений слова в каждый документ и частоту встречания по тексту. У частотного анализа проблема заключается в том, что наибольшую частоту встречания (а, соответственно, и

наибольший вес в модели принятия решений в дальнейшем) будут иметь слова, не несущие смысловую нагрузку документа. Один из способов решения этой проблемы является введение более сложных метрик, называемых TF-IDF, и понижающих оценку слова, которое часто встречается во всех схожих документах.

TF-IDF (сокращение от term frequency — inverse document frequency) – это статистическая мера для оценки важности слова в документе, который является частью некоторого набора данных. Метрики этой модели:

- TF (term frequency — частота слова) – частота слова в каждом документе относительно количества всех слов в этом документе
- IDF (inverse document frequency — обратная частота документа) – логарифм доли документов, в которых это слово встречается хотя бы раз. Логарифм используется для инвертации логики если слово встречается в каждом документе, то скорее оно не очень информативно.
- TFIDF – произведение метрик TF и IDF. Определяется для слова, документа и набора документов.

Векторизация в итоге осуществляется по всем словам. Длина выходного вектора – количество уникальных слов среди всех документов. Для каждого документа выходной вектор – это TFIDF слов в этом документе. Пример использования векторизации на основе этой метрики:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer("english")
features = vectorizer.fit_transform(texts)
```

Модели принятия решений.

В качестве модели принятия решения при приведении входных данных к нужному виду может использоваться один из способов, рассмотренных в предыдущей лабораторной работе: случайные леса, нейронные сети прямого распространения, наивный байесовский классификатор, логистическая регрессия, метод k ближайших соседей. Кроме этого, для обработки текста часто используется архитектуры нейронных сетей с памятью. Наиболее часто используемыми моделями являются сети долгой краткосрочной памяти (LSTM - long short-term memory).

Для работы с сетями долгой краткосрочной памяти в лабораторной работе предлагается использовать библиотеку Keras. Keras предоставляет

унифицированный интерфейс над несколькими «бэкэндами» (backend) – реализациями работы с вычислениями в описательном виде, позволяющие запускать их как на центральном процессоре, так и на видеокартах. Самый популярный бэкэнд, который используется в Keras по умолчанию – TensorFlow, который необходимо установить для корректной работы Keras (`pip install tensorflow`). Саму библиотеку Keras устанавливать не нужно, так как после версии Tensorflow 2.0 она была интегрирована. После этого необходимо проимпортировать необходимые в дальнейшем функции, как показано в примерах далее.

Для работы с текстом в Keras предусмотрен набор служебных классов из пространства keras.preprocessing.text. Наиболее простой вариант обработки предоставляет класс Tokenizer:

```
from tensorflow.keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer(num_words=1000)
tokenizer.fit_on_texts(X_train)
sequences = tokenizer.texts_to_sequences(X_train)
```

Tokenizer представляет собой счётчик уникальных слов, который каждому слову в предложении ставит в соответствие некоторое число. При этом нумеруются только «num_words» наиболее встречаемых слов.

Для подачи на вход модели все векторы должны иметь одинаковую длину (`maxlen=150`). При этом векторы меньшего размера дополняются нулями, а большего обрезаются:

```
sequences_matrix = pad_sequences(sequences, maxlen=150)
```

Создание модели осуществляется с помощью класса Sequential, который позволяет задать произвольную конфигурацию последовательных слоев нейронной сети любой сложности. Пример создания модели:

```
from tensorflow.keras import layers, losses, metrics, optimizers
from tensorflow.keras.models import Sequential

classifier = Sequential([
    layers.Embedding(input_dim=1000,                output_dim=32,
input_length=sequence_length),
    layers.LSTM(64),
    layers.Dense(1, activation='sigmoid')
])
classifier.summary()
```

Embedding-слой преобразует численные значения слов, полученные при токенизации, в «one-hot» векторы, а затем снижает размерность данных преобразовывая их в менее разреженный вид (`output_dim=32`).

LSTM-слой создает указанное количество нейронов с долгой краткосрочной памятью.

Dense-слой – полносвязный слой с указанным количеством нейронов и функцией активации.

Для получения описания созданной модели можно использовать функцию summary(). Функция compile() преобразует описание модели в набор инструкций для «бэкэнда» и готовит тем самым модель к использованию. Для обучения сети используется функция fit.

```
    classifier.compile(loss=losses.binary_crossentropy,
optimizer=optimizers.RMSprop(), metrics=['accuracy'])
    classifier.fit(sequences_matrix,    y_train,    batch_size=128,
epochs=5, validation_split=0.2)
```

Keras позволяет использовать при обучении только метрику accuracy. Чтобы получить другие метрики, необходимо проанализировать результаты, полученные на тестовой выборке, при этом входной набор данных для тестовой выборки должен быть преобразован по тому же словарю, что и для обучающей части выборки.

Задача 1. Анализ sms-сообщений

При нежелательной рассылке через sms сообщения на основе баз данных пользователей сотовых операторов высылаются спамерские сообщения рекламного либо мошеннического характера. Ваша задача заключается в разработке классификатора, который на основе анализа данных сообщений способен отличить сообщение со спамом.

Датасет содержит 5574 sms-сообщения на английском языке, разделенных на спам («spam») и обычное сообщение («ham») в столбце «v1».

Источник данных:

https://www.kaggle.com/uciml/sms-spam-collection-dataset

Задача 2. Анализ писем (e-mail)

Спам рассылки по электронной почте грозят пользователям не только потраченным временем, но и содержат дополнительные возможности для злоумышленников, такие как фишинг, вымогание денег через ложную информацию, рассылка вирусных и троянских программ и т.д. Ваша задача заключается в разработке классификатора, который на основе анализа данных сообщений способен отличить письма со спамом.

Датасет содержит письма на английском языке, помеченные как содержащие или не содержащие спам.

Источник данных:

https://www.kaggle.com/karthickveerakumar/spam-filter

Задача 3. Анализ sms-сообщений

При нежелательной рассылке через sms сообщения на основе баз данных пользователей сотовых операторов высылаются спамерские сообщения рекламного либо мошеннического характера. Ваша задача заключается в разработке классификатора, который на основе анализа данных сообщений способен отличить сообщение со спамом.

Датасет содержит 5156 sms-сообщений, разделенных на спам («spam») и обычное сообщение («ham») в столбце «type».

Источник данных:

https://www.kaggle.com/datasets/shravan3273/sms-spam

Задача 4. Анализ писем (e-mail)

Спам рассылки по электронной почте грозят пользователям не только потраченным временем, но и содержат дополнительные возможности для злоумышленников, такие как фишинг, вымогание денег через ложную информацию, рассылка вирусных и троянских программ и т.д. Ваша задача заключается в разработке классификатора, который на основе анализа данных сообщений способен отличить письма со спамом.

Датасет содержит письма, помеченные как содержащие или не содержащие спам.

Источник данных:

https://www.kaggle.com/datasets/nitishabharathi/email-spam-dataset

Задача 5. Анализ комментариев

В последнее время наряду с рассылкой спама посредством электронной почты, а также мессенджеров, возросло количество спама в комментариях на различных популярных сайтах, ресурсах, к постам в социальных сетях и т.д. Анализ комментариев на предмет спама позволяет оградить пользователей от мошенничества и дезинформации в сети Интернет. Ваша задача заключается в разработке классификатора, который на основе анализа данных о комментарии, оставленном пользователем, способен отличить содержание со спамом.

Источник данных:

https://www.kaggle.com/goneee/youtube-spam-classifiedcomments

Задание к лабораторной 3.

1. Получите задание в соответствии с вашим вариантом:

№ варианта	Используемые методы для решения
1	• Деревья принятия решений • LSTM
2	• Случайные леса • LSTM
3	• Метод опорных векторов • LSTM
4	• Логистическая регрессия • LSTM
5	• Метод k ближайших соседей • LSTM
6	• Наивный байесовский классификатор • LSTM

2. Проанализируйте датасет. Опишите его характеристики: имеющиеся колонки, размер.
3. Решите задачу бинарной классификации на предложенном датасете методами, соответствующими вашему варианту, на основе библиотек scikit-learn и keras/tensorflow. Оцените и обоснуйте полученные результаты.
4. Попробуйте улучшить работу алгоритмов с помощью изменения различных параметров, дополнительной обработки датасета и т.д.
5. Представьте ваше исследование в виде Блокнота JupyterLab, содержащего все пункты задания.