Методы предобработки текстовых постов

Эйрих М., студент кафедры информатики СПбГУ,
michael.eirich@mail.ru
Олисеенко В.Д., ассистент кафедры информатики СПбГУ,
vdo@dscs.pro
Абрамов М. В., к.т.н., доц. кафедры информатики СПбГУ,
mva@dscs.pro

Аннотация

Статья описывает методы предобработки текстовых данных на русском языке, которые необходимы для использования в машинном обучении. Методы включают удаление стоп-слов, токенизацию, нормализацию, стемминг и лемматизацию, и помогают улучшить качество анализа и классификации данных. В статье представляются примеры использования морфологических анализаторов и библиотек, для решения различных задач в области NLP [1].

Введение

В настоящее время предобработка данных остается одной из основных проблем при работе с большими объемами информации, что требует разработки более эффективных подходов к созданию и преобразованию данных. Это особенно важно при создании приложений для работы с данными, которые должны быть способны обрабатывать и анализировать большие объемы информации.

Предварительная обработка данных является наиболее важным этапом в анализе данных, поскольку, если этот этап пропустить, то аналитические алгоритмы, используемые для дальнейшего анализа, не будут работать правильно. Это может привести к некорректным или неэффективным результатам, что известно как принцип GIGO — garbage in, garbage out[1].

В статье рассмотрены различные методы предварительной обработки текстовых данных на русском языке, необходимые для эффективного использования алгоритмов машинного обучения. Также подробно описаны методы, применяемые в разработаном приложении, которые могут помочь улучшить качество обработки текста и повысить точность аналитических моделей.

[1] GIGO — https://wires.onlinelibrary.wiley.com/doi/full/10.1002/widm.1456

Методология обработки данных

Стоп-слова

Одной из основных форм предварительной обработки является фильтрация ненужных данных [2]. В обработке естественного языка "бесполезные слова" называются стоп-словами [3]. Стоп-слова – это часто используемые слова, например "и", "или", "а", "в" и т.д., которые поисковая система должна игнорировать как при индексировании записей для поиска, так и при получении их в результате поискового запроса, за исключением строгого поиска по определенной фразе. При анализе текстовых данных и построении моделей NLP эти слова не придают документу особой важности. В случаях, когда необходимо классифицировать текст, например, при фильтрации спама [4] или генерации заголовков (комментариев) [5] к изображению или тегам, можно использовать техники удаления стоп-слов. Напротив, в таких задачах, как машинный перевод, языковое моделирование, краткое резюме текста, рекомендуется оставлять стоп-слова, так как они имеют большое значение (листинг 1).

```
import nltk
from nltk.corpus import stopwords

nltk.download("stopwords")
stop_words = stopwords.words("russian")
```

Листинг 1. Сбор стоп-слов с помощью NLTK

Токенизация

После фильтрации данных необходимо удалить знаки препинания и служебные символы. Это можно сделать вручную, написав код для проверки и удаления ненужных символов, либо использовать инструменты из NLTK (Natural Language Toolkit)[2]. Кроме того, этот процесс может быть объединен с токенизацией предложений на слова - выделением отдельных токенов каждого предложения. Токенизация предложений должна учитывать особенности языка, с которым вы работаете [6]. Например, при использовании инструментов NLTK для русского языка мы можем получить неожиданные результаты. Например, слово "Санкт-Петербург" может быть разделено на два отдельных слова "Санкт" и "Петербург", что

[2] NLTK — https://www.nltk.org

может привести к потере смысла в некоторых случаях. Поэтому на этом этапе необходимо быть особенно внимательным (листинг 2).

```
Не ветер, а какой-то ураган!
```

Листинг 2. Исходный текст до токенизации и удаления стоп-слов

```
['Не', 'ветер', ',', 'какой-то', 'ураган', '!']
```

Листинг 3. Текст после токенизации и удаления стоп-слов

Нормализация слов

В любом естественном языке слова могут быть записаны в более чем в одной форме, в зависимости от ситуации.

Например:

- «Я был на конференции вчера».

- «Я буду выступать на конференции 28 апреля».

- «Я иногда бываю на конференциях».

Во всех этих предложениях мы видим, что слово «быть» употребляется в нескольких различных формах. Для нас, людей, это легко понять, что «быть» представляет собой некую деятельность, независимо от формы, в которой мы видим это слово — «был», «бываем», «был», «бываем» и т.д. Однако для компьютеров все эти формы являются разными словами, поэтому необходима их нормализация к корневому слову «быть». Нормализация — это процесс приведения слова к единой канонической форме [7], который может быть выполнен двумя способами — стеммингом [8] и лемматизацией [9].

Стемминг и Лемматизация

Варианты применения алгоритмов стемминга и лемматизации зависят от цели и задач, которые перед ними ставятся.

Стемминг часто используется в поисковых системах для уменьшения количества форм слова и упрощения поиска по ключевым словам. Однако этот метод не учитывает контекст и может приводить к ошибкам, так как не всегда корневая форма совпадает с исходным словом.

Лемматизация используется для более точной обработки текста, так как она учитывает контекст и грамматические отношения слов в предложении. Она часто применяется в задачах обработки естественного языка, таких как машинный перевод, анализ тональности, категоризация текста и т.д. Однако этот метод требует больше вычислительных ресурсов и времени на обработку, чем стемминг.

Также есть комбинированные методы, которые используют как стемминг, так и лемматизацию в зависимости от цели обработки текста и типа задачи.

Для неанглийских слов можно использовать стеммер Snowball[3]. На самом деле, Snowball — это язык для создания стеммеров и был добавлен в NLTK версии 2.0b9 в виде отдельного класса SnowballStemmer. Этот стеммер поддерживает следующие языки: датский, английский, финский, немецкий, испанский, шведский, финский, самый важный для нас русский и некоторые другие языки. Так как NLTK имеет мало возможностей для русского языка, мы рассмотрим только стемминг с помощью SnowballStemmer. Рассмотрим исходный текст и текст полученный после стемминга. Как мы видим, после выполнения обработки слов с помощью стемминга, большинство слов были обрезаны (листинг 4).

```
from nltk.stem import SnowballStemmer
snowball = SnowballStemmer(language="russian")

# Функция обработки
def snowball_text(text):

  # Приводим весь текст к нижнему регистру
  text = text.lower()
  # Разбиваем текст на слова
  words = text.split()
  # Получаем токены
  tokens = [snowball.stem(word) for word in words]

  # Объединяем токены в текст
  text = ' '.join(tokens)

  # Возвращаем лемматизированный текст
  return text

# Проверка работоспособности функции лемматизации на основе SnowballStemmer
text = "Никто так не нуждается в отпуске, как человек, только что вернувшийся из отпуска (с)"
snowball_text(text)
```

```
'никт так не нужда в отпуске, как человек, тольк что вернувш из отпуск (с)'
```

Листинг 4. Стемминг с использованием SnowballStemmer из NLTK

Когда доступны два варианта, лемматизация всегда будет лучшим

[3]Snowball — https://www.nltk.org/_modules/nltk/stem/snowball.html

вариантом, чем стемминг [10]. Стемминг алгоритмы являются оптимизированным способом идентификации родственных слов с помощью относительно короткого алгоритма и без необходимости в словарных данных для каждого языка. Недостатком является то, что он не всегда точен: иногда он соединяет родственными отношениями слова, которые не происходят от одного и того же слова, но, с другой стороны, не идентифицирует родственные формы конкретного слова. В свою очередь, лемматизация всегда даст лучший результат, потому что лемматизаторы полагаются на правильные языковые данные (словари) для идентификации слова с его леммой. Кроме того, результатом всегда будет другой элемент словаря (инфинитивы, формы единственного числа и т.д.), а не "основа", с которой иногда могут возникнуть трудности.

Рассмотрим лемматизацию с помощью морфологического анализатора pymorphy2[4] (листинг 5).

```
import pymorphy2
morph = pymorphy2.MorphAnalyzer()

# Функция обработки
def pymorphy2_text(text):

  # Приводим весь текст к нижнему регистру
  text = text.lower()
  # Разбиваем текст на слова
  words = text.split()
  # Получаем токены
  tokens = [morph.parse(word)[0].normal_form for word in words]

  # Объединяем токены в текст
  text = ' '.join(tokens)

  # Возвращаем лемматизированный текст
  return text

# Проверка работоспособности функции лемматизации на основе pymorphy2
text = "Никто так не нуждается в отпуске, как человек, только что вернувшийся из отпуска (с)"
pymorphy2_text(text)
```

```
'никто так не нуждаться в отпуске, как человек, только что вернуться из отпуск (с)'
```

Листинг 5. Лемматизация с помощью PyMorphy2

Заключение

В данной работе были рассмотрены основные методы предварительной обработки данных, необходимые для их эффективного использования в задачах машинного обучения. Были

[4]Pymorphy2 — https://pymorphy2.readthedocs.io/en/stable/

подробно описаны такие методы, как токенизация, удаление стоп-слов и нормализация. Кроме того, были представлены различные подходы к реализации этих методов с использованием библиотек NLKT, SnowballStemmer и pymporphy2. Однако, следует помнить, что выбор конкретного метода зависит от задачи и свойств исходных данных.

Литература

1. Ofer D., Brandes N., Linial M. The language of proteins: NLP, machine learning & protein sequences //Computational and Structural Biotechnology Journal. – 2021. – Т. 19. – С. 1750-1758.
2. Maharana K., Mondal S., Nemade B. A review: Data pre-processing and data augmentation techniques //Global Transitions Proceedings. – 2022.
3. Shelke N. et al. An efficient way of text-based emotion analysis from social media using LRA-DNN //Neuroscience Informatics. – 2022. – С. 100048.
4. Dada E. G. et al. Machine learning for email spam filtering: review, approaches and open research problems //Heliyon. – 2019. – Т. 5. – №. 6. – С. e01802.
5. Huang Y. et al. Towards automatically generating block comments for code snippets //Information and Software Technology. – 2020. – Т. 127. – С. 106373.
6. Aso M. et al. Acoustic model-based subword tokenization and prosodic-context extraction without language knowledge for text-to-speech synthesis //Speech Communication. – 2020. – Т. 125. – С. 53-60.
7. Mehmood K. et al. An unsupervised lexical normalization for Roman Hindi and Urdu sentiment analysis //Information Processing & Management. – 2020. – Т. 57. – №. 6. – С. 102368.
8. Singh J., Gupta V. A novel unsupervised corpus-based stemming technique using lexicon and corpus statistics //Knowledge-Based Systems. – 2019. – Т. 180. – С. 147-162.
9. Freihat A. A. et al. Towards an optimal solution to lemmatization in Arabic //Procedia computer science. – 2018. – Т. 142. – С. 132-140.

10. Moon S., Chi S., Im S. B. Automated detection of contractual risk clauses from construction specifications using bidirectional encoder representations from transformers (BERT) //Automation in Construction. – 2022. – T. 142. – C. 104465.