**Машинное обучение для текстов**

**Векторизация текстов**

**Введение**

Векторизация текстов — процесс преобразования текстов в числовые векторы для машинного обучения. Цель: извлечение признаков из текстов для задач классификации, анализа тональности и т.д. Основные идеи: упрощение текста через предобработку, представление в виде мешка слов или TF-IDF. Пример задачи: классификация твитов по тональности (положительная — 1, отрицательная — 0).

**Предобработка текста**

**Токенизация**

**Токенизация** (англ. tokenization) — разбиение текста на токены: отдельные фразы, слова, символы.

**Лемматизация**

**Лемматизация** (англ. lemmatization) — приведение слова к начальной форме (лемме). Библиотеки для русского: pymorphy2, UDPipe, pymystem3.

**Функция лемматизации с pymystem3:**

python

from pymystem3 import Mystem

m = Mystem()

lemmas = m.lemmatize("лемматизируй это") *# Возвращает: ['лемматизировать', ' ', 'это', '\n']*

clean\_text = " ".join(lemmas) *# Объединяет в строку: 'лемматизировать это \n'*

**Создание корпуса**

**Корпус** (англ. corpus) — набор текстов с разметкой эмоций и ключевых слов. Пример:

python

import pandas as pd

data = pd.read\_csv('tweets.csv')

corpus = data['text'].values.astype('U') *# Преобразование в Unicode*

**Регулярные выражения**

**Регулярные выражения** (англ. regular expressions) — инструмент для поиска и замены по шаблону. **Функция re.sub():** Заменяет совпадения в тексте.

python

import re

clean\_text = re.sub(r'[^а-яА-ЯёЁ ]', ' ', text) *# Оставляет кириллицу и пробелы*

clean\_text = " ".join(clean\_text.split()) *# Удаляет лишние пробелы*

Шаблон: r'[а-яА-ЯёЁ]' — кириллические буквы; ^ — negation (не эти символы).

**Мешок слов и N-граммы**

**Мешок слов** (англ. bag of words) — модель, преобразующая текст в вектор без учета порядка слов.

Матрица: строки — тексты, столбцы — уникальные слова, значения — частота вхождений. Пример вектора: Для текста "ананас в шампанский" — [2, 2, 2] (если уникальные слова: ананас, в, шампанский).

**N-граммы** — последовательности из N элементов (слов).

Униграммы (N=1): отдельные слова.

Биграммы (N=2): пары слов.

Тригаммы (N=3): тройки слов. Пример: Для "В сто сорок солнц закат пылал" триграммы — "в сто сорок", "сто сорок солнц" и т.д. Векторизация N-грамм аналогична мешку слов.

**Создание мешка слов**

**CountVectorizer** — класс для создания мешка слов.

python

from sklearn.feature\_extraction.text import CountVectorizer

count\_vect = CountVectorizer()

bow = count\_vect.fit\_transform(corpus) *# Матрица частот*

words = count\_vect.get\_feature\_names\_out() *# Словарь уникальных слов*

shape = bow.shape *# Размер матрицы*

print(bow.toarray()) *# Просмотр как массив*

Для N-грамм:

python

count\_vect = CountVectorizer(ngram\_range=(2, 2)) *# Биграммы*

**Стоп-слова**

**Стоп-слова** (англ. stopwords) — слова без смысловой нагрузки (союзы, предлоги).

python

import nltk

from nltk.corpus import stopwords

nltk.download('stopwords')

stop\_words = set(stopwords.words('russian'))

count\_vect = CountVectorizer(stop\_words=stop\_words) *# Исключает стоп-слова*

**TF-IDF**

**TF-IDF** (term frequency-inverse document frequency) — оценка важности слова.

**TF** (term frequency): Частота слова в тексте = (кол-во вхождений) / (общее кол-во слов).

**IDF** (inverse document frequency): log(общее кол-во текстов / кол-во текстов со словом). Формула: TF-IDF = TF \* IDF. Большое значение — слово уникально для текста в корпусе.

**TfidfVectorizer** — класс для расчета TF-IDF.

python

from sklearn.feature\_extraction.text import TfidfVectorizer

count\_tf\_idf = TfidfVectorizer(stop\_words=stop\_words)

tf\_idf = count\_tf\_idf.fit\_transform(corpus) *# Для обучающей выборки: fit\_transform; для тестовой: transform*

Для N-грамм:

python

count\_tf\_idf = TfidfVectorizer(stop\_words=stop\_words, ngram\_range=(2, 3)) *# Би- и триграммы*

**Классификация тональности**

**Анализ тональности** (сентимент-анализ) — выявление эмоционально окрашенных слов для классификации (положительный/отрицательный).

Признаки: TF-IDF векторы.

Целевой признак: 1 (положительный), 0 (отрицательный). Используется для оценки отзывов, реакций в соцсетях.

**Словарь терминов**

**Токенизация**: Разбиение на токены.

**Лемматизация**: Приведение к лемме.

**Корпус**: Размеченный набор текстов.

**Регулярные выражения**: Поиск по шаблону.

**Мешок слов**: Вектор без порядка слов.

**N-грамма**: Последовательность из N слов.

**Стоп-слова**: Бессмысленные слова.

**TF-IDF**: Оценка важности слова.

**Сентимент-анализ**: Анализ тональности.

**Языковые представления**

**Embeddings**

Embeddings — векторные представления текстов, где сложная структура (текст) вкладывается в простой вектор. Частный случай: **word embeddings** (эмбеддинги слов) — векторы, содержащие данные о соотношениях слов и их свойствах.

**Свойства слов**: скрытые смыслы (например, "моряк" включает "мужчина", "профессия", "морской").

**Семантика**: лексическое значение слова, отличающее его от других.

**Контекст**: окружение слов, определяющее их смысл (например, "лента" в разных предложениях).

**Близость векторов**: рассчитывается евклидовым расстоянием (меньше расстояние — сильнее сходство). Близкие векторы отражают похожие слова по смыслу и контексту.

**Зачем нужны embeddings**: Кодировать смысл слов, учитывать контекст, в отличие от мешок слов или TF-IDF, которые основаны на частоте.

**Word2vec**

**Word2vec** (word to vector) — метод построения векторных представлений слов на основе контекста.

Основная идея: Смысл слов определяется контекстом. Задача — предсказать, являются ли слова соседями в "окне" (максимальное расстояние между словами).

Процесс:

Лемматизация текста.

Формирование n-грамм (например, пятиграмма для окна размера 5).

Создание пар слов-соседей (положительные примеры).

Генерация случайных пар (отрицательные примеры) для бинарной классификации: соседи (1) или нет (0).

Цель обучения: Подобрать векторы, отражающие соседство слов, сводя задачу к классификации.

**Embeddings для классификации**

Векторные представления применяются для задач классификации/регрессии текстов. Модель состоит из двух блоков:

**Модель перевода в векторы**: Преобразует текст в эмбеддинги.

**Модель классификации**: Прогнозирует на основе векторов (например, эмоциональную окраску: 0 — отрицательная, 1 — положительная).

**Предобработка**:

Токенизация: Разбиение текста на слова (токены).

Лемматизация: Приведение к начальной форме (не требуется для сложных моделей вроде BERT).

Очистка: Удаление стоп-слов и ненужных символов.

Добавление маркеров: Токены начала (101) и конца (102) предложения.

**Перевод в векторы**: Токены сопоставляются со словарем, формируя векторы фиксированной длины.

Пример: Классификация твитов по тональности.

**BERT**

**BERT** (Bidirectional Encoder Representations from Transformers) — нейронная сеть для создания языковых моделей. Разработана Google для понимания контекста запросов. Поддерживает 104 языка, включая русский.

Эволюция от word2vec. Другие модели: FastText, GloVe (Global Vectors for Word Representation), ELMO (Embeddings from Language Models), GPT (Generative Pre-Training Transformer). Самые точные: BERT и GPT-3 (GPT-3 недоступен открыто).

**Как работает**:

Учитывает контекст соседних и дальних слов.

Задачи обучения:

Угадывание маскированных слов (MASK) в предложении (например, "Красный клюв тупика [MASK] на голубом [MASK]").

Определение связи между предложениями.

Преимущество: Предобученные модели для построения эмбеддингов.

**RuBERT и предобработка**

**RuBERT** — версия BERT, предобученная на русском разговорном корпусе (из DeepPavlov). Используется для задач вроде классификации твитов по тональности.

Библиотеки: torch, transformers.

**Предобработка**:

**Токенизатор**: BertTokenizer(vocab\_file='vocab.txt') — разбивает текст на токены из словаря, без лемматизации.

**Encode**: tokenizer.encode(text, add\_special\_tokens=True) — преобразует текст в номера токенов, добавляя [101] (начало) и [102] (конец).

**Padding**: Дополнение векторов нулями до максимальной длины (n, например 280): padded = vector + [0] \* (n - len(vector)).

**Attention mask**: Маска для значимых токенов: attention\_mask = np.where(padded != 0, 1, 0).

**Эмбеддинги RuBERT**

**Конфигурация и модель**:

config = BertConfig.from\_json\_file('bert\_config.json')

model = BertModel.from\_pretrained('rubert\_model.bin', config=config)

**Преобразование в эмбеддинги** (батчами для экономии памяти):

batch\_size = 100

Используйте tqdm.notebook для прогресса.

Цикл:

python

embeddings = []

for i in notebook.tqdm(range(padded.shape[0] // batch\_size)):

batch = torch.LongTensor(padded[batch\_size\*i : batch\_size\*(i+1)])

attention\_mask\_batch = torch.LongTensor(attention\_mask[batch\_size\*i : batch\_size\*(i+1)])

with torch.no\_grad():

batch\_embeddings = model(batch, attention\_mask=attention\_mask\_batch)

embeddings.append(batch\_embeddings[0][:, 0, :].numpy())

features = np.concatenate(embeddings)

**no\_grad()**: Отключает градиенты для ускорения (модель не обучается).

Результат: Матрица признаков (features) для обучения классификатора (например, LogisticRegression).