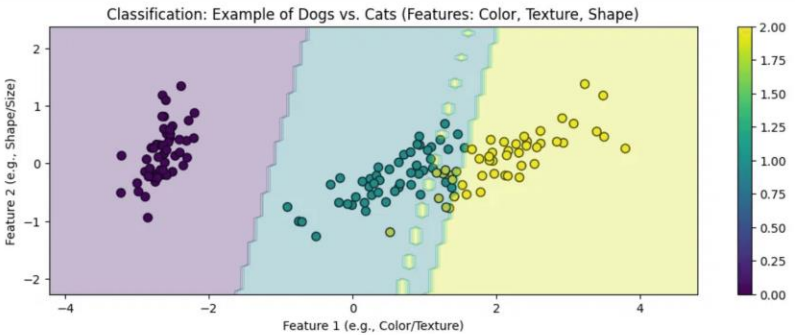
Классификация текстовых данных

Классификация: предсказание категориальной переменной



Постановка задачи

По выборке $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, где $y_i \in \{1, 2, \ldots, K\}$, построить модель $f: X \to Y$ для предсказания класса.

Популярные метрики

1. Accuracy

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$

Доля правильных предсказаний среди всех объектов.

2. F1-Score

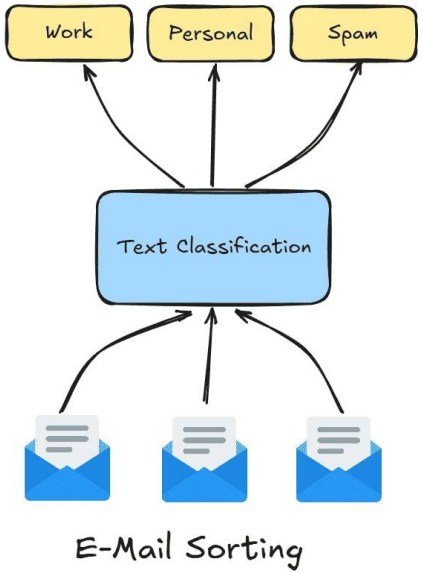
$$F1 = 2 \times (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})$$

$$\text{Precision} = TP/(TP+FP), \quad \text{Recall} = TP/(TP+FN)$$

Гармоническое среднее между точностью и полнотой.

Классификация текстов решает задачи автоматического распределения документов по категориям

Классификация текстов представляет собой фундаментальную задачу обработки естественного языка, которая заключается в автоматическом присвоении текстовым документам предопределенных категорий или меток.



Примеры задач классификации

Анализ тональности отзывов

Определение позитивных и негативных мнений клиентов

Категоризация новостей

Распределение статей по темам: спорт, политика, технологии

Обнаружение спама

Фильтрация нежелательных сообщений в электронной почте

Классификация обращений

Автоматическая сортировка запросов в службу поддержки

Text Data Preprocessing

Предобработка текста является критически важным этапом, который напрямую влияет на качество работы модели. Исследования показывают, что до 80% времени в проектах машинного обучения тратится именно на подготовку данных. Для русского языка этот этап имеет особенности, связанные с богатой морфологией и флективностью языка.

Этапы обработки

1. **Очистка текста**
   Удаление HTML-тегов, специальных символов, пунктуации
2. **Нормализация регистра**
   Приведение к нижнему регистру (Бежали –> бежали)
3. **Токенизация**
   Разбиение текста на отдельные слова или токены
4. **Лемматизация**
   Приведение слов к словарной форме (бежали → бежать)
5. **Удаление стоп-слов**
   Исключение частотных слов (и, в, на, с)

Пример кода с spaCy

```
import re
import spacy

nlp = spacy.load("ru_core_news_sm")

def preprocess_text(text):
 # Очистка от лишних символов
 text = re.sub(r'[^а-яА-Яа-zA-Z\s]', ' ',
 ; text = re.sub(r'\s+', ' ', text).strip()

 # Токенизация и лемматизация
 doc = nlp(text)
 tokens = [token.lemma_ for token in doc
 if token.is_alpha and
 not token.is_stop]

 return ' '.join(tokens)

# Пример
text = "Расскажи другу о своем любимом уроке!"
clean = preprocess_text(text)
print(clean) # рассказать друг любимый урок
```

EDA для текстовых данных

Исследовательский анализ данных (EDA) для текстов позволяет понять структуру датасета, выявить дисбаланс классов, определить наиболее частотные слова и обнаружить потенциальные проблемы. Этот этап помогает принять обоснованные решения о выборе модели и методов обработки данных.

Основные метрики EDA

1. Распределение длин текстов

Гистограмма количества слов/символов в документах

2. Частотный анализ

Топ-20 наиболее частых слов по классам

3. Баланс классов

Соотношение количества примеров в каждом классе

4. Облако слов

Визуализация наиболее частых терминов

5. Уникальность словаря

Размер словаря, покрытие редких слов

Пример кода для анализа

```
import pandas as pd
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from collections import Counter

# Анализ длин текстов
df['text_length'] = df['text'].apply(
 lambda x: len(x.split()))
df['text_length'].hist(bins=50)
plt.xlabel('Длина текста (слов)')
plt.show()

# Частотный анализ
all_words = ' '.join(df['text']).split()
word_freq = Counter(all_words)
print(word_freq.most_common(20))

# Облако слов
wordcloud = WordCloud(width=800,
 height=400,
 background_color='white'
).generate(' '.join(all_words))
plt.imshow(wordcloud)
plt.axis('off')
plt.show()

# Баланс классов
```

Кодирование текстов: Bag of Words (мешок слов)

Bag of Words (BoW) — классический подход к представлению текстов в виде векторов для машинного обучения. Метод игнорирует порядок слов и грамматику, фокусируясь только на частоте встречаемости каждого слова в документе. Несмотря на простоту, BoW остается эффективным базовым методом для многих задач классификации текстов.

Визуализация концепции

Document D1	*The child makes the dog happy* the: 2, dog: 1, makes: 1, child: 1, happy: 1
Document D2	*The dog makes the child happy* the: 2, child: 1, makes: 1, dog: 1, happy: 1

↓

	child	dog	happy	makes	the	BoW Vector representations
D1	1	1	1	1	2	[1,1,1,1,2]
D2	1	1	1	1	2	[1,1,1,1,2]

Этапы создания Bag of Words

1. Токенизация

Разбиение текста на отдельные слова (токены)

2. Создание словаря

Формирование списка уникальных слов из всех документов корпуса

3. Векторизация

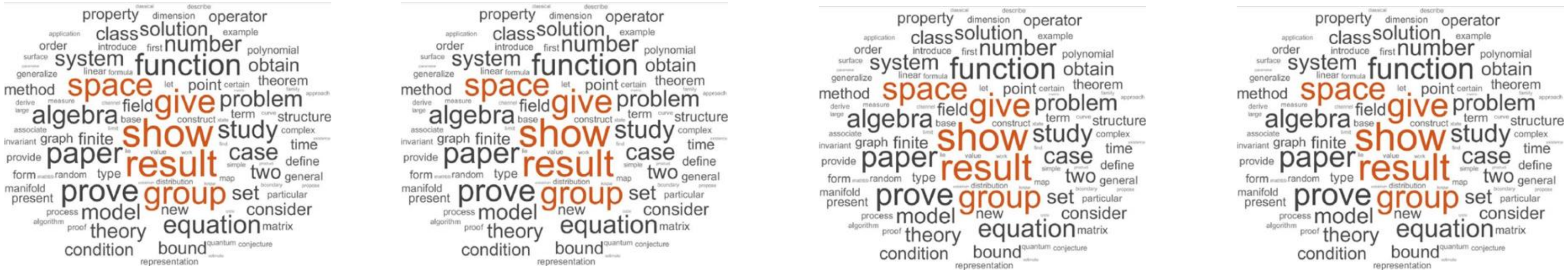
Подсчет частоты каждого слова из словаря в каждом документе

4. Представление

Каждый документ = вектор размерности |словарь|

Ключевые особенности

Игнорирование порядка: "кот ловит мышь" = "мышь ловит кот" • **Разреженность:** большинство элементов вектора равны нулю • **Размерность:** равна размеру словаря



TF-IDF взвешивает важность слов с учетом их частоты в документе и корпусе

TF-IDF (Term Frequency-Inverse Document Frequency) — это статистическая мера, используемая для оценки важности слова в документе, являющемся частью коллекции или корпуса. Она представляет собой значительное улучшение по сравнению с моделью Bag of Words (BoW).

Формула и Интуиция

TF-IDF = TF(t,d) × IDF(t)

TF (Term Frequency): Частота термина t в документе d.

IDF (Inverse Document Frequency): Мера редкости термина в корпусе.

IDF(t) = log(N / df),
где N — количество документов в корпусе,
df — количество документов, содержащих термин t.

Интуиция: Слова, которые часто встречаются в конкретном документе (высокий TF), но редко во всем корпусе (высокий IDF), получают наивысший вес.

Преимущества и Применение

Снижение веса стоп-слов: Общие слова (артикли, предлоги) получают низкий IDF, что эффективно снижает их важность.

Выделение уникальных терминов: Позволяет выделить слова, которые являются ключевыми для данного документа.

Применение: Использование векторов TF-IDF для задач классификации текстов (например, спам/не спам) и в системах информационного поиска (ранжирование документов по релевантности).

Пример кода (Scikit-learn, русский язык)

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Пример корпуса на русском языке
corpus = [
    'Кот сидит на окне и смотрит на улицу',
    'Собака бежит за котом по улице',
    'Простое предложение без кота и собаки'
]

# Инициализация с настройкой для русского языка
vectorizer = TfidfVectorizer(
    lowercase=True,
    stop_words=None# Стоп-слова лучше задать явно для русского
)

# Создание TF-IDF матрицы
tfidf_matrix = vectorizer.fit_transform(corpus)

# Вывод названий признаков (слов)
feature_names = vectorizer.get_feature_names_out()
# ['без', 'бежит', 'и', 'кот', 'котом', 'на', 'окне', 'по', 'предложение', 'простое', 'сидит', 'собака',
'смотрит', 'улице', 'улицу', 'за']# Вывод матрицы (фрагмент)# print(tfidf_matrix.toarray())
```

N-граммы

N-граммы — это последовательности из **N соседних слов или символов**. Они позволяют сохранить **локальный контекст** и улавливать устойчивые выражения, что является ключевым улучшением по сравнению с моделью Bag of Words.

Типы N-грамм

Типы: Униграммы (N=1), Биграммы (N=2), Триграммы (N=3).

Пример (фраза 'машинное обучение текстов'):

```
Униграммы (N=1): ['машинное', 'обучение', 'текстов']
Биграммы (N=2): ['машинное обучение', 'обучение текстов']
Триграммы (N=3): ['машинное обучение текстов']
```

Баланс: Преимущества и Недостатки

Преимущества:

- Сохранение локального контекста
- Улавливание устойчивых выражений

Недостатки:

- Экспоненциальный рост размерности
- Разреженность данных (Sparsity)

Пример кода (Scikit-learn)

```
from sklearn.feature_extraction.text import CountVectorizer# Создание биграмм и триграмм (N=2 и N=3)
vectorizer = CountVectorizer(ngram_range=(2, 3))

corpus = ['машинное обучение текстов', 'анализ данных']
X = vectorizer.fit_transform(corpus)

# Вывод полученных признаков
print(vectorizer.get_feature_names_out())
# ['анализ данных' 'машинное обучение' 'машинное обучение текстов' 'обучение текстов']
```

Модули **CountVectorizer** и **TfidfVectorizer** используют параметр **ngram_range=(min_n, max_n)** для генерации n-грамм.

Рекомендации по выбору N

Оптимальный диапазон: Обычно 1-3. Редко используется N > 3 из-за проблемы разреженности.

Символьные N-граммы: Полезны для русского языка (морфология) и для повышения устойчивости к опечаткам.

Кодирование текстовых данных: Эмбеддинги

Эмбеддинги представляют собой ключевую технологию современного NLP, которая позволяет преобразовать текст в числовые векторы фиксированной размерности.

В отличие от классических методов (Bag of Words, TF-IDF), эмбеддинги сохраняют семантическую близость слов: похожие по смыслу слова имеют близкие векторные представления.

Векторное пространство эмбеддингов обладает интересными свойствами: можно выполнять арифметические операции над векторами слов. Классический пример: вектор("король") - вектор("мужчина") + вектор("женщина") ≈ вектор("королева").

Типы эмбеддингов

Word Embeddings (Word2Vec, GloVe, FastText)

Статические векторы для каждого слова, не учитывают контекст предложения, быстрые в вычислении

Contextual Embeddings (BERT, ELMo)

Динамические векторы, зависящие от контекста. Одно слово может иметь разные векторы в разных предложениях

Пример Word2Vec

```
from gensim.models import Word2Vec

# Обучение модели
sentences = [
 ["кошка", "ловит", "мышь"],
 ["собака", "гоняется", "за", "кошкой"],
 ["кошка", "спит", "на", "диване"]
]

model = Word2Vec(
 sentences,
 vector_size=100,
 window=5,
 min_count=1,
 workers=4
)

# Получение вектора слова
vector = model.wv['кошка']
print(f"Размерность: {len(vector)}")

# Поиск похожих слов
similar = model.wv.most_similar('кошка', topn=3)
print(f"Похожие слова: {similar}")
```

Как измерить сходство текстов?

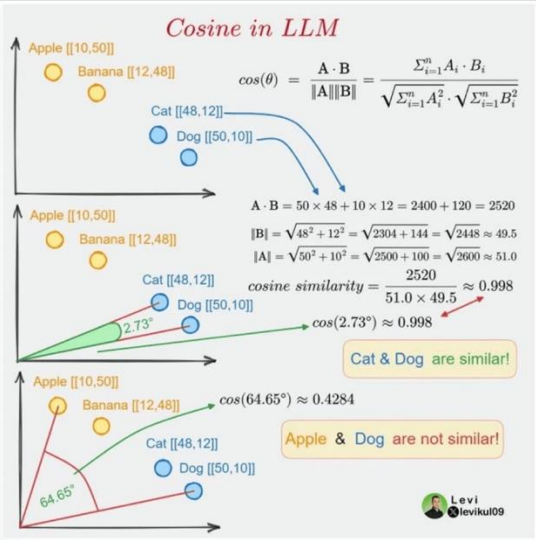
Как измерить сходство текстов?

Косинусное расстояние — одна из наиболее популярных метрик для измерения семантической близости текстов в векторном пространстве. В отличие от евклидова расстояния, косинусное сходство учитывает только направление векторов, игнорируя их длину, что делает его идеальным для сравнения текстов разной длины.

Формула

$$\cos(\theta) = (A \cdot B) / (||A|| \times ||B||)$$

A · B — скалярное произведение векторов
||A||, ||B|| — нормы (длины) векторов
θ — угол между векторами



Сравнение метрик и применение

Метрика	Формула	Применение
Косинусное	cos(θ) = A·B / (||A| |×| |B| |)	Тексты разной длины, семантика
Евклидово	√Σ(A_i - B_i)²	Числовые данные, координаты
Манхэттенское	Σ|A_i - B_i|	Высокоразмерные данные
Жаккара	|A∩B| / |A∪B|	Множества, бинарные данные

Пример кода

```
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer

texts = ["Отличный товар, очень доволен покупкой",
 "Превосходный продукт, рекомендую всем",
 "Ужасное качество, не советую"]

vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(texts)
cosine_sim = cosine_similarity(tfidf_matrix)

print("Сходство 1-2:", cosine_sim[0][1]) # ~0.65
print("Сходство 1-3:", cosine_sim[0][2]) # ~0.15
```

Современные модели классификации текстов

Attention

Архитектура Transformer, представленная в статье "Attention Is All You Need" (2017), произвела революцию в обработке естественного языка. В отличие от рекуррентных нейронных сетей, которые обрабатывали текст последовательно, трансформеры обрабатывают весь текст параллельно, что значительно ускорило обучение и позволило эффективно работать с длинными последовательностями.

Преимущества

Параллельная обработка текста

Трансформеры обрабатывают все слова одновременно, в отличие от последовательной обработки в RNN, что значительно ускоряет обучение модели

Работа с длинными последовательностями

Эффективная обработка длинных текстов без проблемы затухающего градиента, характерной для рекуррентных сетей

Прямые связи между словами

Каждое слово может напрямую взаимодействовать с любым другим словом в тексте, независимо от расстояния между ними

Механизм Self-Attention

Self-attention — это ключевой механизм, который позволяет модели определить, какие слова в предложении наиболее важны для понимания каждого конкретного слова.

При обработке слова модель "обращает внимание" на все остальные слова в тексте и вычисляет их важность (веса внимания). Это позволяет учитывать контекст и семантические связи между словами.

Пример работы механизма

В предложении "Банк реки был крутым" слово "банк" может означать финансовое учреждение или берег реки.

Self-attention анализирует окружающие слова ("реки", "крутым") и определяет, что здесь "банк" означает берег, а не финансовую организацию.

Transformer

Основные компоненты

Multi-Head Attention

Параллельное применение нескольких механизмов внимания для фокусировки на разных аспектах данных

Positional Encoding

Кодирование позиции слов, так как трансформер не имеет встроенного понимания порядка

Feed-Forward Networks

Полносвязные слои для преобразования представлений после механизма внимания

Layer Normalization

Нормализация для стабильности обучения и ускорения сходимости модели

Формула Self-Attention

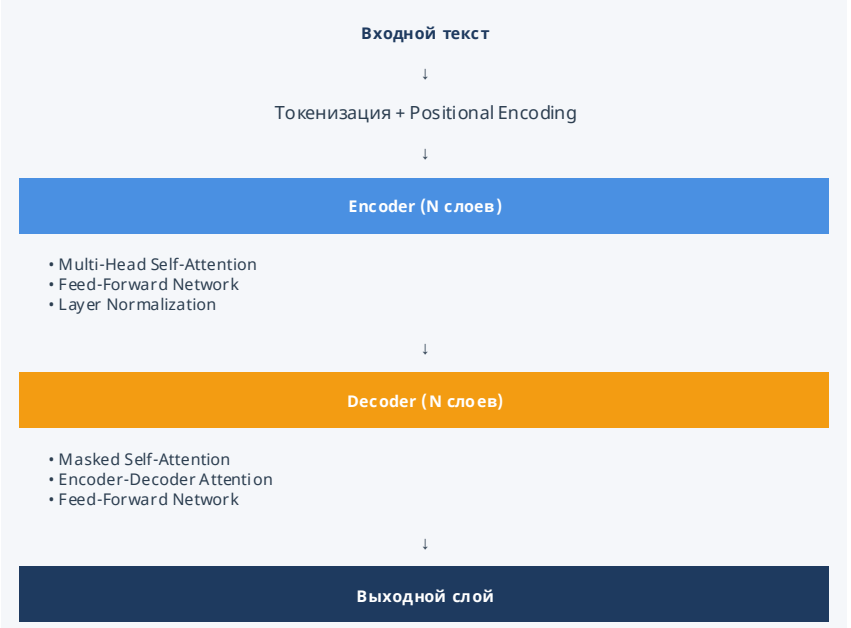
$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T / \sqrt{d_k}) \times V$$

Q (Query) — запросы, определяют что ищем

K (Key) — ключи, с чем сравниваем

V (Value) — значения, что извлекаем

d_k — размерность ключей для масштабирования



Ключевые особенности

- Encoder обрабатывает входной текст
- Decoder генерирует выходную последовательность
- Каждый слой включает механизм внимания и feed-forward сеть
- Residual connections и нормализация между слоями

Модель BERT

BERT (Bidirectional Encoder Representations from Transformers) представляет собой модель, которая предобучается на огромных объемах текста и затем может быть **адаптирована (fine-tuned) для конкретных задач**.

Ключевое отличие BERT — **двунаправленность**: модель анализирует контекст слова как слева, так и справа одновременно.

Предобучение BERT

Masked Language Modeling (MLM)

Случайно маскируется 15% токенов в тексте, и модель предсказывает их на основе окружающего контекста. Это позволяет модели научиться понимать двунаправленные зависимости между словами.

Next Sentence Prediction (NSP)

Модель определяет, следует ли второе предложение за первым в исходном тексте. Эта задача помогает модели понимать связи между предложениями, что важно для задач понимания текста.

Применение для классификации

Для адаптации BERT к задаче классификации текстов используется специальный токен [CLS], который добавляется в начало каждого текста. Выходное представление этого токена содержит агрегированную информацию обо всем тексте.

Этапы применения

- Добавляется токен [CLS] в начало текста
- Выход для [CLS] используется как представление всего текста
- Добавляется классификационный слой (Dense layer)
- Fine-tuning на размеченных данных

Эволюция трансформеров

После успеха BERT появилось множество улучшенных и специализированных моделей. RoBERTa устранила некоторые недостатки BERT, GPT специализируется на генерации текста, а DistilBERT предлагает баланс между качеством и скоростью. Выбор модели зависит от конкретной задачи: для классификации лучше подходят encoder-only модели (BERT, RoBERTa), для генерации — decoder-only модели (GPT).

Сравнение современных моделей

Модель	Архитектура	Размер	Особенности	Применение
BERT	Encoder-only	110M-340M	Двунаправленный контекст	Классификация, NER, Q&A
RoBERTa	Encoder-only	125M-355M	Улучшенное предобучение, без NSP (Next Sentence Prediction)	Классификация, понимание текста
GPT-3	Decoder-only	175B	Генеративная модель	Генерация текста, диалоги
DistilBERT	Encoder-only	66M	40% меньше BERT, 97% качества	Быстрая классификация
ALBERT	Encoder-only	12M-235M	Меньше параметров, разделение эмбеддингов	Эффективная классификация
T5	Encoder-Decoder	60M-11B	Text-to-Text подход	Универсальные задачи NLP

Ключевые различия: Encoder-only модели (BERT, RoBERTa, DistilBERT, ALBERT) оптимальны для понимания и классификации текста, decoder-only модели (GPT-3) — для генерации, а encoder-decoder модели (T5) — для универсальных задач преобразования текста.

Выбор модели

Практические рекомендации по выбору модели

Высокая точность

BERT, RoBERTa, ALBERT — идеальны для задач, где критична максимальная точность классификации. RoBERTa показывает лучшие результаты на большинстве бенчмарков благодаря улучшенному предобучению.

Скорость работы

DistilBERT, TinyBERT, rubert-tiny — облегченные версии, которые работают в 2-3 раза быстрее при сохранении 95-97% качества. Оптимальны для продакшен-систем с высокой нагрузкой.

Генерация текста

GPT-2, GPT-3 — специализированы на генерации связного текста, дополнении предложений, создании диалогов. Используют decoder-only архитектуру.

Универсальность

T5, BART — универсальные модели с encoder-decoder архитектурой, которые могут решать различные задачи NLP через text-to-text подход: классификацию, суммаризацию, перевод.

Русский язык

ruBERT, rubert-tiny, multilingual-BERT — специализированные модели для русского языка. ruBERT обучен на русских текстах и показывает лучшее качество для русскоязычных задач.

Примеры использования с pipeline API

```
from transformers import pipeline

# RoBERTa для классификации
clf = pipeline("sentiment-analysis",
 model="FacebookAI/roberta-base")
result = clf("This product is amazing!")
print(result)
# [{'label': 'POSITIVE', 'score': 0.9998}]

# DistilBERT для быстрой классификации
fast_clf = pipeline("sentiment-analysis",
 model="distilbert-base-uncased")
result = fast_clf("Great service!")
print(result)
# [{'label': 'POSITIVE', 'score': 0.9995}]

# Multilingual BERT для русского языка
ru_clf = pipeline("sentiment-analysis",
 model="bert-base-multilingual-cased")
result = ru_clf("Отличный товар, рекомендую!")
print(result)
# [{'label': 'POSITIVE', 'score': 0.9992}]

# Batch processing для нескольких текстов
texts = ["Great!", "Terrible!", "Not bad"]
results = clf(texts)
for text, res in zip(texts, results):
 print(f"{text}: {res['label']}")
```

NLP библиотеки в Python

Современная экосистема Python для обработки естественного языка включает библиотеки различного уровня: от базовых инструментов предобработки до фреймворков для работы с state-of-the-art моделями. Правильный выбор библиотеки зависит от задачи, требований к производительности и доступных вычислительных ресурсов.

Основные библиотеки

NLTK — базовая библиотека для обучения NLP

Токенизация, стемминг, лемматизация, работа с корпусами текстов

spaCy — высокопроизводительная библиотека

Быстрая обработка больших объемов, NER, dependency parsing, поддержка русского языка

Hugging Face Transformers — современные модели

10,000+ предобученных моделей, BERT, GPT, RoBERTa, простой API для fine-tuning

Gensim — эмбеддинги и тематическое моделирование

Word2Vec, FastText, Doc2Vec, LDA, эффективная работа с большими корпусами

scikit-learn — классическое машинное обучение

TF-IDF, CountVectorizer, классификаторы (Naive Bayes, SVM, Logistic Regression)

PyMorphy3 — морфологический анализ русского

Точная лемматизация русских слов, определение частей речи, склонение и спряжение

Сравнение производительности

Библиотека	Скорость	Точность	Русский язык
NLTK	★★★	★★	★★
spaCy	★★★★★	★★★★	★★★★
Transformers	★★★	★★★★★	★★★★★
Gensim	★★★★	★★★	★★★★
scikit-learn	★★★★★	★★★	★★★★★
PyMorphy3	★★★★	★★★★★	★★★★★

Рекомендации по выбору

Для продакшена: spaCy (предобработка) + Transformers (модели)
Для экспериментов: NLTK + Gensim
Для русского языка: PyMorphy3 + ruBERT
Для классического ML: scikit-learn + TF-IDF

Интерпретация трансформеров

README Apache-2.0 license

BertViz

Visualize Attention in NLP Models

Quick Tour • Getting Started • Colab Tutorial • Paper

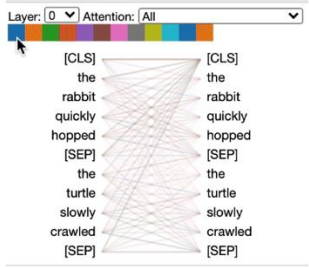
BertViz is an interactive tool for visualizing attention in Transformer language models such as BERT, GPT2, or T5. It can be run inside a Jupyter or Colab notebook through a simple Python API that supports most Huggingface models. BertViz extends the Tensor2Tensor visualization tool by Llion Jones, providing multiple views that each offer a unique lens into the attention mechanism.

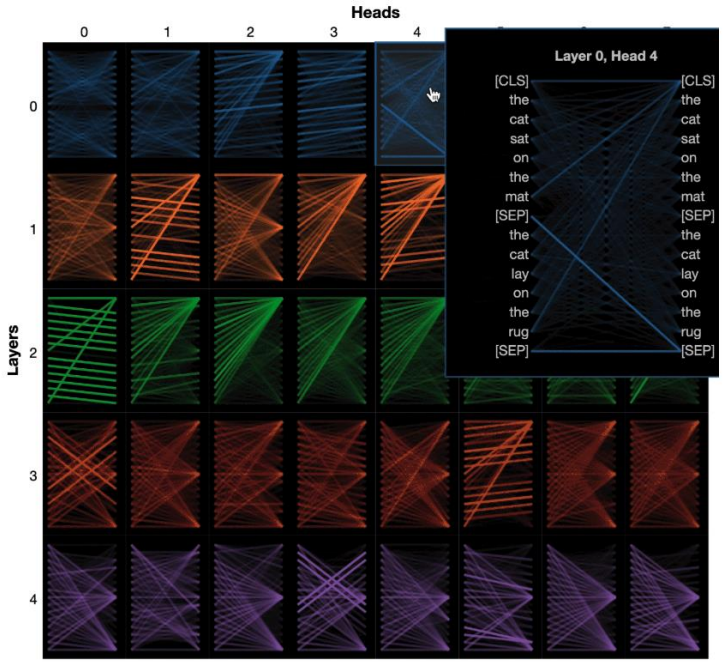
🚀 Quick Tour

Head View

The *head view* visualizes attention for one or more attention heads in the same layer. It is based on the excellent Tensor2Tensor visualization tool by Llion Jones.

Try out the head view in the Interactive Colab Tutorial (all visualizations pre-loaded).





https://github.com/jessevig/bertviz

Практика