2025-11-12

Библиотеки машинного обучения в экосистеме Python

scikit-learn

Документация: User Guide — scikit-learn 1.7.2 documentation

Официальный сайт: https://scikit-learn.org/

Страница библиотеки в PyPI: scikit-learn · PyPI

```
pip install scikit-learn
```

Исходный код: GitHub - scikit-learn/scikit-learn: scikit-learn: machine learning in Python

Наиболее популярная в отрыве от каких-либо отдельных направлений развития машинного обучения библиотека Python для соответствующих моделей.

Изначально появилась в 2007-м году как библиотека предиктивной аналитики. Само название «Scikit» появилось от сокращения «SciPy Toolkit» — т.е. задумана она была как расширения инструментов SciPy для предиктивной аналитики.

Соответственно, построена на NumPy, SciPy, matplotlib и легко интегрируется с ними.

Автор — французский исследователь, аналитик данных и разработчик **David Cournapeau** (Давид Курнапо). В дальнейшем разработку продолжили другие исследователи и программисты, например, Александр Грамфор (agramfort (Alexandre Gramfort) · GitHub)

Библиотека написана на Python, Cython, C++ и чистом C.

> **Tip**
>
> **Cython** — надмножество (суперсет) языка Python, которое позволяет писать Python-код, напрямую преобразуемый в C-структуры.
> Это компилируемый язык, который на этапе компиляции преобразует аннотированный Python-код в C, после чего автоматически предоставляет обертки (wrappers) для того, чтобы с бинарными исполняемыми файлами можно было продолжать работать при помощи Python.
> Одна из самых распространенных областей применения Cython — прекомпиляция модулей стандартной библиотеки CPython.

Основные возможности:

- содержит реализации большинства популярных алгоритмов регрессии, классификации, кластеризации
- включает также методы снижения размерности (например, метод главных компонент, PCA `Principal component analysis`), различные алгоритмы предварительной обработки данных (препроцессинга) — нормализация, детекция аномалий, очистка данных и т.п., интерфейс подбора моделей (т.е. тестирование нескольких моделей с возможностью выбора наилучшей для задачи)
- включает в себя множество классических датасетов для демонстрации и обучения (например, Ирисы Фишера и т.д.)
- пермиссивная (разрешающая) лицензия BSD и открытый исходный код — можно свободно использовать все модели и алгоритмы в любых (в том числе коммерческих) проектах
- полная совместимость с pandas 2.2 и NumPy 2.0
- поддержка разреженных датафреймов (актуально для текстовых данных)

Из минусов:

- не подходит для deep learning (глубокого обучения), нужно использовать более специализированные инструменты
- в последнее время выявились проблемы масштабирования (по-настоящему большие датасеты сложно обрабатывать, не хватает инструментария для эффективного скейлинга)
- нет возможности легко перевести данные из датафрейма pandas в вид, пригодный для модели из scikit-learn — придется использовать в качестве промежуточных структур ndarray из NumPy

Структура и основные модели и алгоритмы библиотеки

1. Обучение с учителем (Supervised learning)

2. Обучение без учителя (Unsupervised learning)
3. Выбор моделей и их оценка
4. Инспекция данных
5. Визуализация данных
6. Преобразования данных
7. Инструменты загрузки готовых датасетов

- **Обучение с учителем** (линейные модели, SVM, k-ближайших соседей, наивный Байес, деревья решений, ансамблевые методы)
- **Обучение без учителя** (кластерный анализ: алгоритм k-средних, алгоритм распространения близости, DBCSAN, OPTICS, спектральная кластеризация, метод Уорда для иерархической кластеризации)

Модели и алгоритмы обучения с учителем

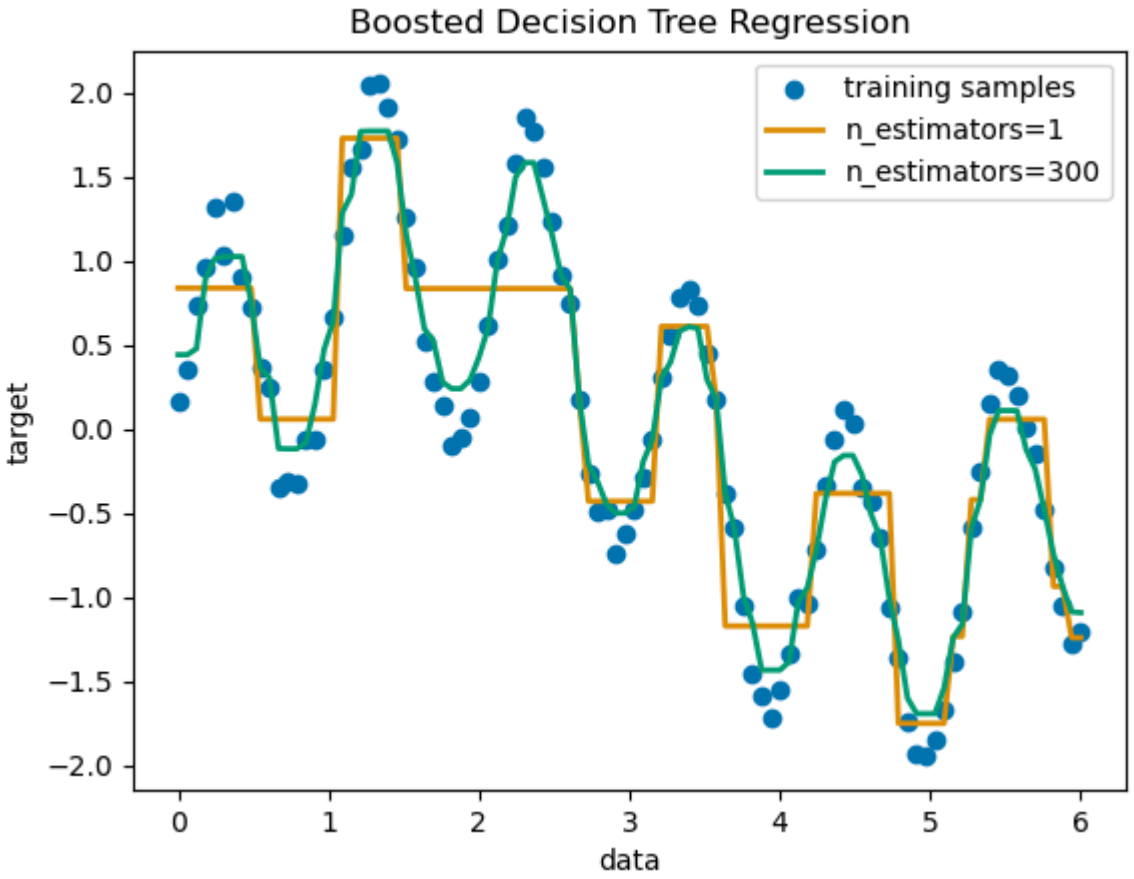
[1. Supervised learning — scikit-learn 1.7.2 documentation]

- Линейные модели, включая ElasticNet, логистическую регрессию (прогнозирует обычно вероятности), перцептроны, стохастический градиентный спуск (Stochastic Gradient Descent, SGD), гребневая регрессия (ридж-регрессия, ridge regression, регрессия Тихонова) — относится к методам снижения размерности
- Машины опорных векторов (Support Vector Machines, SVM) — модель для классификации
- Метод k-ближайших соседей — тоже классификация
- Наивный Байес (наивный байесовский классификатор) — классификация
- Деревья решений (decision tree) — регрессия или классификация
- Ансамблевые методы — случайный лес (random forest) — ансамбль на базе дерева решений; деревья с градиентным бустингом (дерево решений + градиентный бустинг), голосующий классификатор (Voting Classifier) — объединение нескольких моделей машинного обучения (можно разных) в один ансамбль, каждая модель производит свою классификацию, после чего производится голосование

> **Info**
>
> ИТМО — праймер по регрессиям Вариации регрессии — Викиконспекты

- Модуль нейронных сетей — вариации перцептронов

Пример реализации обучения с учителем — дерево решений + AdaBoost Decision Tree Regression with AdaBoost — scikit-learn 1.7.2 documentation



Больший плюс документации — есть возможность скачать примеры как в виде .py-файла (plot_adaboost_regression.py), так и в виде Jupyter-ноутбука (plot_adaboost_regression.ipynb). Весь пример реализации такого алгоритма, включая исходные данные, доступен для изучения и экспериментов.

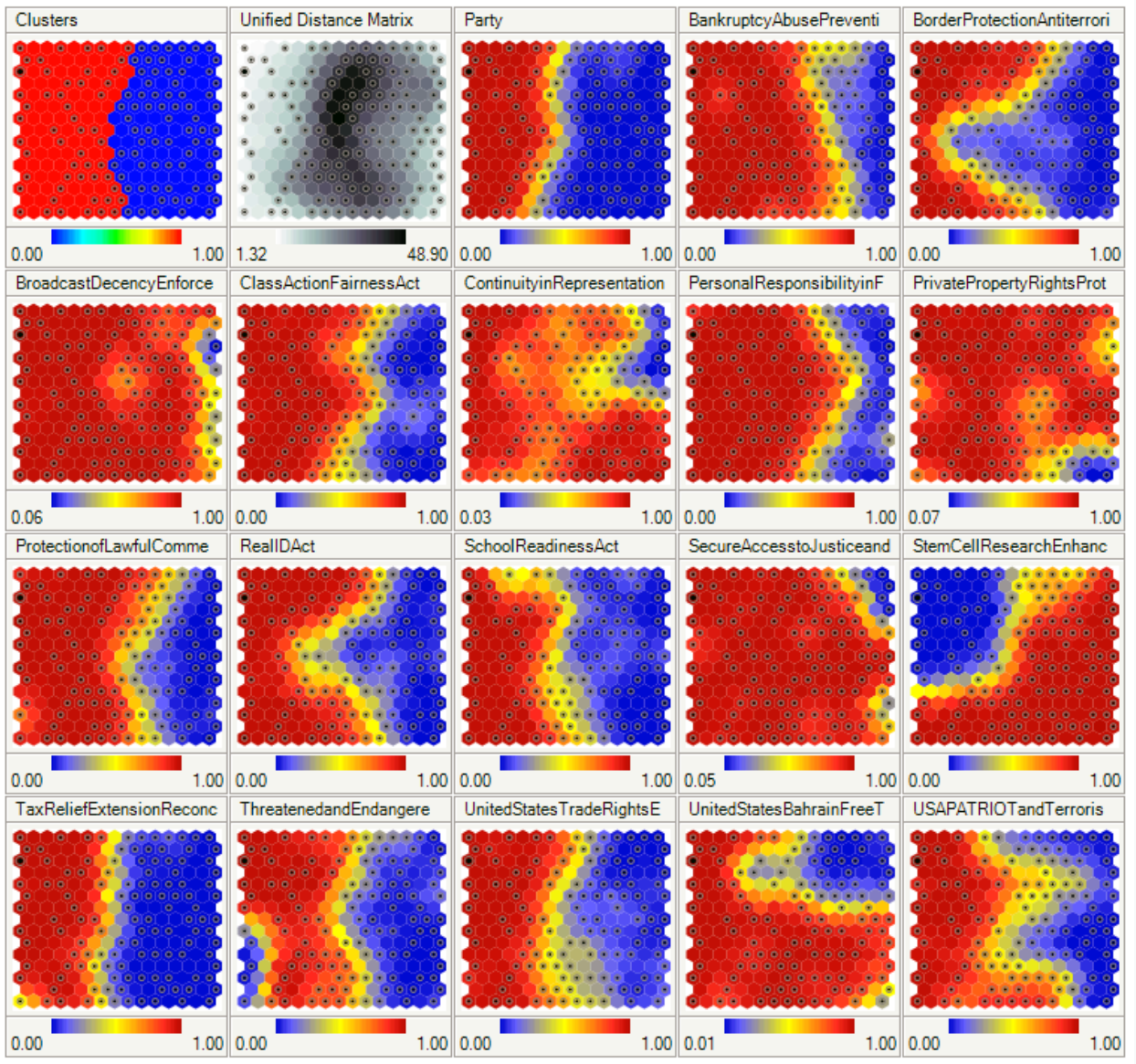
Модели и алгоритмы обучения без учителя

2. Unsupervised learning — scikit-learn 1.7.2 documentation

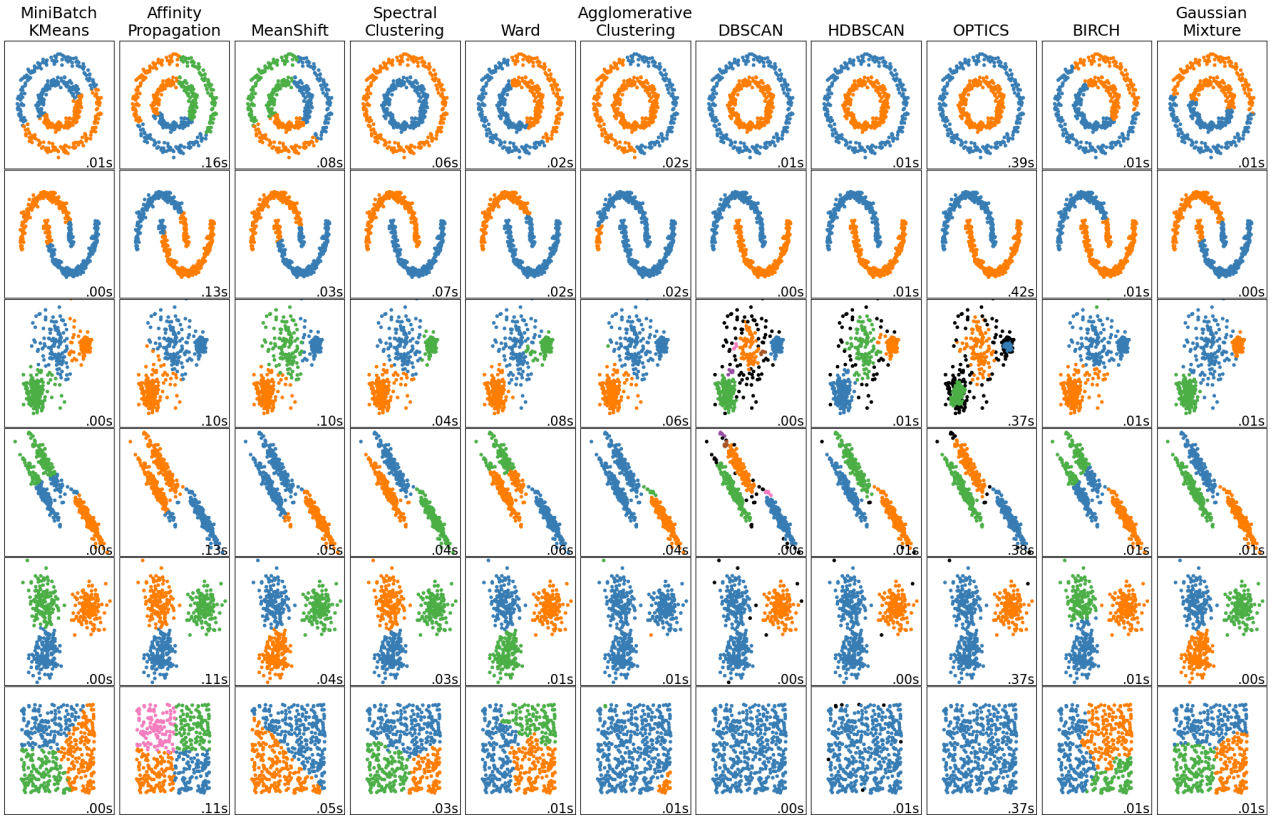
Обучение без учителя не предполагает какой-либо разметки данных до начала обучения модели.

> **Tip**
>
> Самоорганизующиеся карты Кохонена (SOM, Self-organizing maps, Kohonen map) — методика машинного обучения без учителя, которую использовали для двумерного представления данных высоких размерностей. Карты организуются таким образом, что семантически близкие (со схожим набором существенных признаков) образцы (объекты) оказываются ближе, чем семантически различные — и, соответственно, между ними может быть посчитано, например, Евклидово расстояние.
> Также карты могут рассматриваться как нелинейное обобщения метода главных компонент (PCA), т.к. позволяют снизить размерность данных, сохраняя

сущностные характеристики каждой записи (объекта, образца)

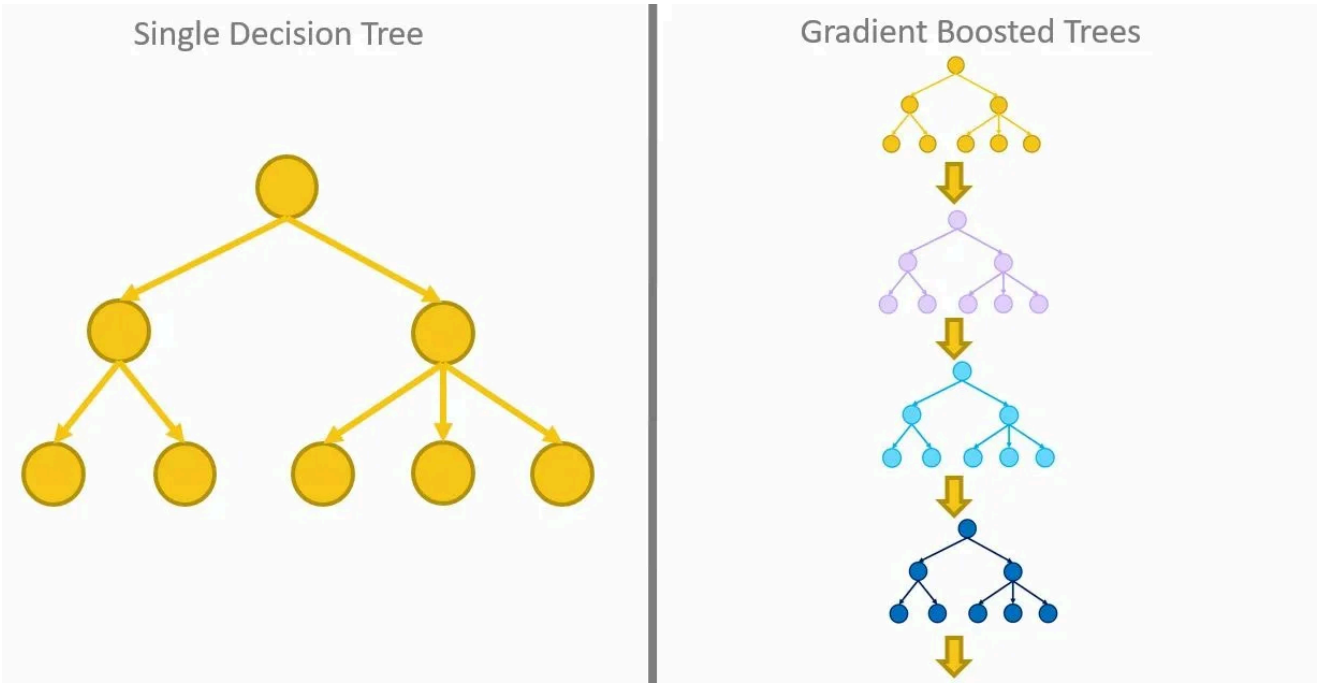


- Кластерный анализ — если классификация предполагает, что у нас есть заранее заданные классы (и мы знаем, сколько их и что мы вкладываем в каждый из классов — например, кошки, собаки, попугаи, капибары и т.п.), то кластерный анализ работает с массивом данных, который модель пытается разбить на некоторое количество групп, объединяя схожие по определенному алгоритму
- Практически все модели работают с векторным представлением данных и вычислением расстояний в N-мерном пространстве



XGBoost, LightGBM, CatBoost

Семейство библиотек для **градиентного бустинга (Gradient Boosting)** — одного из самых эффективных подходов для работы со структурированными/табличными данными.



XGBoost (eXtreme Gradient Boosting)

Официальный сайт: https://xgboost.readthedocs.io/

```
pip install xgboost
```

- использует параллельную обработку и оптимизированное использование памяти

- предотвращает переобучение с помощью L1 (Lasso) и L2 (Ridge) регуляризации
- автоматически обрабатывает пропуски в данных, выбирая оптимальный путь
- поддержка GPU
- градиенты второго порядка: использует не только ошибки, но и скорость их изменения для более информированного обучения

Применение: структурированные данные, задачи классификации и регрессии

LightGBM (Light Gradient Boosting Machine)

Официальный сайт: https://lightgbm.readthedocs.io/

```
pip install lightgbm
```

- значительно быстрее XGBoost за счет использования histogram-based алгоритмов
- Leaf-wise: в отличие от level-wise в XGBoost, выбирает для разбиения лист с наибольшей потерей (более глубокие и точные деревья)
- низкое использование памяти
- нативная обработка категорий без необходимости One-Hot Encoding
- GOSS (Gradient-based One-Side Sampling): умная выборка экземпляров с большими градиентами
- EFB (Exclusive Feature Bundling): объединение взаимоисключающих признаков

HO! Может переобучаться, иногда менее точный

Применение: большие датасеты, задачи, требующие быстрого обучения, работа с категориальными данными

CatBoost (Categorical Boosting)

Официальный сайт: https://catboost.ai/

```
pip install catboost
```

- использует уникальный алгоритм для прямой работы с категориальными данными без необходимости предварительного кодирования
- Ordered Boosting (специальная техника построения деревьев, предотвращающая переобучение)
- автоматически создает новые признаки из комбинаций существующих
    - высокая устойчивость к шуму в данных
- часто требует меньше тюнинга гиперпараметров
- поддержка GPU

Применение: данные с большим количеством категориальных признаков, задачи с минимальной предобработкой данных

PyTorch

Официальный сайт: https://pytorch.org/

```
pip install torch torchvision torchaudio
```

Исходный код: https://github.com/pytorch/pytorch

PyTorch — фреймворк глубокого обучения с открытым исходным кодом. Стал де-факто стандартом для исследований в области глубокого обучения.

- Интуитивный API, похожий на обычный Python-код
- Легкая отладка с использованием стандартных Python-инструментов (pdb, print)
- Бесшовная интеграция с экосистемой Python
- Динамические вычислительные графы (Dynamic Computation Graph) — возможность изменять структуру модели в процессе выполнения без необходимости перекомпиляции, граф строится во время выполнения кода; критически важно для моделей с динамическим потоком управления (например, RNN, трансформеры с переменной длиной последовательности)
- Поддержка GPU/TPU: NVIDIA CUDA, AMD ROCm, Apple Silicon

Основные модули

- **torch.nn** — построение и обучение нейронных сетей (слои, функции потерь, активации)
- **torch.optim** — оптимизаторы (SGD, Adam, AdamW и др.)
- **torch.autograd** — автоматическое дифференцирование
- **torchvision** — инструменты для компьютерного зрения (датасеты, модели, трансформации)
- **torchaudio** — обработка аудио
- **torchtext** — обработка текста

Экосистема PyTorch

- **PyTorch Lightning** — высокоуровневая обертка для упрощения кода обучения
- **fastai** — высокоуровневая библиотека для быстрого прототипирования
- **timm** (PyTorch Image Models) — современные архитектуры для компьютерного зрения

Применение: исследования в области глубокого обучения, NLP, компьютерное зрение, reinforcement learning, production ML systems

TensorFlow

Официальный сайт: https://www.tensorflow.org/

```
pip install tensorflow
```

Исходный код: https://github.com/tensorflow/tensorflow

TensorFlow — фреймворк машинного обучения с открытым исходным кодом, разработанный Google Brain. Один из самых зрелых и широко используемых фреймворков для production ML систем.

- Eager Execution: операции выполняются немедленно, возвращая конкретные значения вместо построения графа
- XLA (Accelerated Linear Algebra) — оптимизирующий компилятор для линейной алгебры, автоматическое слияние операций для повышения эффективности
- Keras — единый API для трех бэкендов: TensorFlow, JAX и PyTorch

Применение: production ML-системы, мобильные приложения (TensorFlow Lite/LiteRT), edge computing, web deployment (TensorFlow.js)

JAX

Официальный сайт: https://jax.readthedocs.io/

```
pip install jax jaxlib
```

Исходный код: https://github.com/google/jax

JAX — библиотека Python от Google для высокопроизводительных численных вычислений и машинного обучения, объединяющая автоматическое дифференцирование с компилятором XLA.

- Интерфейс практически идентичен NumPy (`jax.numpy` — drop-in замена для `numpy`)
- Функциональное программирование
- Унифицированный интерфейс для CPU, GPU, TPU
- XLA компиляция для оптимизации
- Эффективного вычисления производных высоких порядков (JAX поддерживает производные произвольного порядка за один проход, остальные фреймворки оптимизированы для производных первого порядка)

Применение: физические симуляции, квантовые вычисления, решение дифференциальных уравнений, масштабное обучение на TPU

Hugging Face Transformers

Официальный сайт: https://huggingface.co/docs/transformers

```
pip install transformers
```

Исходный код: https://github.com/huggingface/transformers

Флагманская библиотека для работы с предобученными трансформерами и large language models (LLM). Де-факто стандарт для задач по обработке естественного языка.

- Доступ к архитектурам моделей (BERT, GPT, T5, LLaMA, Mistral, Qwen и т.д.)
- Model Hub — репозиторий с предобученными моделями
- Модели для текста, изображений, аудио, видео (multimodal AI)

```
from transformers import pipeline

# Классификация или анализ тональности
classifier = pipeline("sentiment-analysis")
result = classifier("I love this product!")

# ЛЛМ
generator = pipeline("text-generation", model="gpt2")
text = generator("Once upon a time", max_length=50)

# Описание изображения
captioner = pipeline("image-to-text")
caption = captioner("path_to_image.jpg")
```

- Оптимизированная генерация для LLM и VLM (Vision Language Models)
- Поддержка streaming
- Множество стратегий декодирования (beam search, sampling, top-k, top-p)

Основные компоненты

Tokenizers

- Токенизация с поддержкой множества языков
- Byte-Pair Encoding (BPE), WordPiece, SentencePiece

Datasets Library

- Готовые датасеты (CoLA, MNLI, SQuAD)
- Эффективная загрузка и препроцессинг

Model Hub

- Централизованный репозиторий моделей
- Версионирование моделей

Применение: классификация текста, извлечение именованных сущности, вопросно-ответные системы, реферирование, перевод, компьютерное зрение, мультимодальные задачи

Ollama

Официальный сайт: https://ollama.com/

Ollama — инструмент для локального запуска больших языковых моделей (LLM) на собственном оборудовании. Docker-подобный подход к управлению моделями.

`pip install ollama` (требует предварительно установленного сервера с оф. сайта)

- простая в использовании (установка и запуск модели одной командой)
- широкая поддержка моделей (Llama, Qwen, Mistral, Gemma, gpt-oss, qwq)
- поддержка квантизации
- кроссплатформенность (Win/Linux/macOS; NVIDIA GPU через CUDA, AMD GPU через ROCm, Apple Silicon (Metal))
- совместимость с OpenAI API

```
# Установка Ollama
curl https://ollama.ai/install.sh | sh

# Запуск модели
ollama run llama3.2

# Запуск других популярных моделей
ollama run mistral
ollama run codellama
ollama run qwen
```

```
import requests

response = requests.post('http://localhost:11434/api/generate',
    json={
        'model': 'llama3.2',
        'prompt': 'Explain quantum computing in simple terms'
    })

print(response.json())
```

LangChain

Официальный сайт: https://www.langchain.com/

Документация: https://docs.langchain.com/

`pip install langchain`

Исходный код: https://github.com/langchain-ai/langchain

LangChain — фреймворк с открытым исходным кодом для разработки приложений на основе больших языковых моделей (LLM). Наиболее популярный инструмент для создания LLM-приложений.

Пять основных абстракций:

1. **Models** — интерфейсы к различным LLM-провайдерам
2. **Prompts** — управление промптами и их шаблонизация
3. **Chains** — последовательности вызовов (LLM → инструмент → LLM)
4. **Agents** — автономные агенты, которые принимают решения
5. **Memory** — сохранение контекста между взаимодействиями

Retrieval-Augmented Generation (RAG):

- Комбинирование LLM с внешними источниками данных
- Векторные базы данных (ChromaDB, Pinecone, Weaviate)
- Semantic search для релевантного контекста
- Критично для приложений, требующих актуальной или domain-specific информации

LangGraph

Официальный сайт: https://langchain.com/langgraph

Документация: https://docs.langchain.com/langgraph

```
pip install langgraph
```

Исходный код: https://github.com/langchain-ai/langgraph

LangGraph — библиотека для построения мультиагентных приложений на основе LLM с использованием графовой архитектуры. Входит в экосистему LangChain.

- Организация действий как узлов в направленном графе
- Рёбра определяют возможные пути и условия перехода
- Поддержка циклических графов — агенты могут возвращаться к предыдущим шагам

vLLM — высокопроизводительный inference

Официальный сайт: https://docs.vllm.ai/

```
pip install vllm
```

Исходный код: https://github.com/vllm-project/vllm

vLLM — высокопроизводительная библиотека для запуска больших языковых моделей с оптимизированным управлением памятью.

ONNX

Официальный сайт: https://onnx.ai/

`pip install onnx onnxruntime`

Исходный код: https://github.com/onnx/onnx

ONNX (Open Neural Network Exchange) — открытый стандарт для представления моделей машинного обучения, обеспечивающий взаимодействие между различными ML-фреймворками

```
# Экспорт PyTorch модели в ONNX
import torch
import torch.onnx

model = MyModel()
model.eval()

dummy_input = torch.randn(1, 3, 224, 224)

torch.onnx.export(model, dummy_input, "model.onnx")
```

```
import onnxruntime as ort

session = ort.InferenceSession("model.onnx")
inputs = {session.get_inputs()[^0].name: input_data}
outputs = session.run(None, inputs)
```

Лабораторная работа №5: Retrieval-Augmented Generation (RAG) для научного анализа

1. Реализовать полный пайплайн обработки документов (загрузка → chunking → embedding → индексирование)
2. Создать систему семантического поиска с использованием локальной LLM
3. Применить RAG-систему для поиска ответов на научные вопросы
4. Оценить качество работы системы через различные метрики
5. Провести анализ эффективности различных конфигураций

Задание 1: Подготовка датасета из Википедии

1. Загрузите статьи из Википедии на тему Пермского периода, используйте MediaWiki API
2. Загрузите не менее 5-10 статей (10+ тысяч слов)

3. Выведите статистику (сколько документов, символов, слов, средняя длина статьи и т.д.)

Задание 2: Обработка документов и создание эмбеддингов

1. Разбейте документы на чанки:

```
from langchain_community.document_loaders import DirectoryLoader,
TextLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter

class DocumentProcessor:
    def __init__(self, documents_dir: str, chunk_size: int = 1000,
chunk_overlap: int = 200):
        self.documents_dir = documents_dir
        self.chunk_size = chunk_size
        self.chunk_overlap = chunk_overlap
        self.documents = []
        self.chunks = []

    def load_and_chunk(self) -> dict:
        loader = DirectoryLoader(
            self.documents_dir,
            glob="**/*.txt",
            loader_cls=TextLoader
        )
        self.documents = loader.load()

        splitter = RecursiveCharacterTextSplitter(
            chunk_size=self.chunk_size,
            chunk_overlap=self.chunk_overlap,
            length_function=len,
            separators=["\n\n", "\n", ". ", " ", ""]
        )

        self.chunks = splitter.split_documents(self.documents)

        return {
            'num_documents': len(self.documents),
            'num_chunks': len(self.chunks),
            'total_chars': sum(len(doc.page_content) for doc in
self.documents),
            'avg_chunk_size': sum(len(chunk.page_content) for chunk in
self.chunks) / len(self.chunks)
        }
```

2. Создайте эмбеддинги через Ollama и поместите их в ChromaDB

```
from langchain_chroma import Chroma
from langchain_community.embeddings import OllamaEmbeddings
import shutil

class VectorStoreBuilder:
    def __init__(self, chunks, model: str = "nomic-embed-text",
persist_dir: str = "./chroma_db"):
        self.chunks = chunks
        self.model = model
        self.persist_dir = persist_dir
        self.vectorstore = None

    def build(self) -> Chroma:
        if os.path.exists(self.persist_dir):
            shutil.rmtree(self.persist_dir)

        embeddings = OllamaEmbeddings(
            model=self.model,
            base_url="http://localhost:11434",
            show_progress=True
        )

        self.vectorstore = Chroma.from_documents(
            documents=self.chunks,
            embedding=embeddings,
            persist_directory=self.persist_dir
        )

        return self.vectorstore
```

Задание 3: Построение RAG-системы и ответы на вопросы

1. Инициализируйте систему

```
from langchain_community.llms import Ollama
from langchain.chains import RetrievalQA
from langchain.prompts import PromptTemplate

class PermianRAGSystem:
    def __init__(self, vectorstore, llm_model: str = "llama3.2"):
        self.vectorstore = vectorstore

        self.llm = Ollama(
            model=llm_model,
            base_url="http://localhost:11434",
            temperature=0.3,
            top_p=0.9,
```

```
            num_predict=512
        )

        # Промпт
        self.prompt_template = """You are an expert in paleontology and the
Permian period.
Use ONLY the provided context from Wikipedia articles to answer the
question.
If the answer is not in the context, say "This information is not available
in the provided documents."

Context:
{context}

Question: {question}

Detailed Answer:"""

        self.prompt = PromptTemplate(
            template=self.prompt_template,
            input_variables=["context", "question"]
        )

        self.qa_chain = RetrievalQA.from_chain_type(
            llm=self.llm,
            chain_type="stuff",
            retriever=vectorstore.as_retriever(search_kwargs={"k": 4}),
            return_source_documents=True,
            chain_type_kwargs={"prompt": self.prompt},
            verbose=False
        )

    def answer_question(self, question: str) -> dict:
        result = self.qa_chain({"query": question})

        return {
            "question": question,
            "answer": result["result"],
            "sources": [
                {
                    "title": doc.metadata.get('source', 'Unknown'),
                    "content": doc.page_content[:300],
                    "metadata": doc.metadata
                }
                for doc in result["source_documents"]
            ]
        }
```

2. Задайте системе набор контрольных вопросов о Пермском периоде (например, сколько он продолжался и когда его дата начала; каким был климат; что вызвало вымирание; какие животные являлись доминирующими видами; как ПП повлиял на эволюцию жизни; какие были условия в морях; какие растения существовали; какая была атмосфера)

```
rag_system = PermianRAGSystem(vectorstore)
results = []

for q in questions:
    result = rag_system.answer_question(q["question"])
    result["category"] = q["category"]
    result["expected_keywords"] = q["expected_keywords"]
    results.append(result)

    print(f"\n{'='*80}")
    print(f"Q{result['question']}")
    print(f"{'—'*80}")
    print(result['answer'])
```

3. Оцените качество (например, задать наборы ключевых слов, которые должны быть в ответах, потом проверить косинусное сходство через scikit-learn)
4. Визуализируйте результаты (точность ответов с RAG и без RAG)

Документация:

- LangChain: https://docs.langchain.com/
- Ollama: https://github.com/ollama/ollama
- ChromaDB: https://docs.trychroma.com/

Статьи (англ и можно их рус. аналоги):

- https://en.wikipedia.org/wiki/Permian
- https://en.wikipedia.org/wiki/Paleozoic
- https://en.wikipedia.org/wiki/Permian%E2%80%93Triassic_extinction_event
- https://en.wikipedia.org/wiki/Siberian_Traps
- https://en.wikipedia.org/wiki/Therapsida