Дипломная работа

по теме: Сравнение различных библиотек для машинного обучения: scikit-learn, TensorFlow и PyTorch

Автор: Сакал Антон Николаевич

Оглавление

Введение

Обоснование выбора темы

- Актуальность: Машинное обучение является одной из наиболее быстро развивающихся областей информационных технологий. Выбор

правильной библиотеки для проекта критически важен для успеха. Понимание сильных и слабых сторон scikit-learn, TensorFlow и PyTorch позволяет разработчикам принимать обоснованные решения.

- Практическая значимость: Эти три библиотеки являются одними из самых популярных и широко используемых в индустрии. Знание их особенностей позволяет повысить эффективность работы и выбирать наиболее подходящий инструмент для конкретной задачи. Сравнение помогает оценить trade-offs между простотой использования, производительностью и гибкостью.
- Образовательная ценность: Сравнение этих библиотек является отличным способом углубить понимание принципов машинного обучения и получить практический опыт работы с различными инструментами. Анализ кода и результатов позволяет понять, как выбор библиотеки влияет на процесс разработки и результаты моделирования.
- Разнообразие задач: Тема охватывает как задачи классической машинной обучения (scikit-learn), так и глубокого обучения (TensorFlow и PyTorch). Это позволяет продемонстрировать широкий спектр приложений и показать, как различные библиотеки подходят для разных типов задач.
- Простота сравнения (относительная): Хотя TensorFlow и PyTorch могут быть более сложными для начинающих, сравнение их с scikit-learn позволяет продемонстрировать эволюцию инструментов машинного обучения и показать, как более сложные библиотеки обеспечивают дополнительную гибкость и производительность при решении более сложных задач.

В целом, данная тема актуальна, практична, образовательна и позволяет сравнить популярные инструменты машинного обучения, что делает ее отличным выбором для исследования и анализа.

Определение цели и задач исследования

Цель исследования

Определить преимущества и недостатки библиотек scikit-learn, TensorFlow и PyTorch для решения задач машинного обучения, сравнив их производительность и удобство использования на конкретных примерах классификации и регрессии.

Задачи исследования

- Изучить теоретические основы: Изучить функциональные возможности, архитектуру и особенности каждой из трех библиотек (scikit-learn, TensorFlow, PyTorch). Определить их сильные и слабые стороны.
- Разработать тестовые приложения: Разработать простые, но репрезентативные приложения для решения задач классификации и регрессии с использованием каждой библиотеки. Выбрать подходящие наборы данных (например, Iris и California Housing).
- Реализовать модели: Реализовать модели машинного обучения (например, логистическую регрессию для классификации и линейную регрессию для регрессии) в каждой из библиотек, используя лучшие практики и оптимизируя код для обеспечения корректного сравнения. Рассмотреть возможность использования более сложных моделей (например, нейронных сетей) для TensorFlow и PyTorch.
- Провести эксперименты: Провести серию экспериментов, оценивая производительность моделей (точность, MSE, время обучения и т.д.) на тестовых данных. Зафиксировать все важные параметры, чтобы обеспечить воспроизводимость результатов.
- Проанализировать результаты: Сравнить полученные результаты, включая метрики производительности и время выполнения, для каждой библиотеки и каждого типа задачи (классификация и регрессия). Проанализировать факторы, которые влияют на различия в производительности.
- Оценить удобство использования: Оценить субъективное удобство использования каждой библиотеки на основе опыта разработки и отладки кода. Учесть такие факторы, как сложность синтаксиса, доступность документации и наличие готовых инструментов.
- Сформулировать выводы: На основе полученных результатов и анализа сформулировать выводы о преимуществах и недостатках каждой библиотеки, определив области их наиболее эффективного применения. Сделать рекомендации по выбору библиотеки в зависимости от конкретных требований проекта.

Эти задачи обеспечивают структурированный подход к исследованию и позволяют получить объективные и обоснованные результаты, необходимые для достижения поставленной цели.

Основные понятия и определения

- Машинное обучение (Machine Learning): Раздел искусственного интеллекта, который фокусируется на разработке алгоритмов, позволяющих компьютерам обучаться на данных без явного программирования. Вместо того, чтобы задавать набор правил, алгоритмы машинного обучения выявляют закономерности и формируют модели на основе предоставленных данных.
- Классификация (Classification): Задача машинного обучения, в которой модель предсказывает категориальную переменную (класс) на основе входных данных. Например, классификация изображений (кот/собака), спам-фильтрация (спам/не спам), или классификация цветов ириса (setosa, versicolor, virginica).
- Регрессия (Regression): Задача машинного обучения, в которой модель предсказывает непрерывную переменную (числовое значение) на основе входных данных. Примеры: предсказание цены дома, прогнозирование температуры, или анализ временных рядов.
- scikit-learn: Библиотека Python с открытым исходным кодом, предоставляющая множество алгоритмов машинного обучения, инструментов для препроцессинга данных и оценки моделей. Она отличается простотой использования и эффективностью для задач классической машинной обучения. Хорошо подходит для быстрой разработки прототипов и экспериментов.
- TensorFlow: Библиотека Python с открытым исходным кодом, разработанная OpenAI, которая широко используется для глубокого обучения. Она предоставляет инструменты для построения и обучения сложных нейронных сетей, а также обладает высокой производительностью, особенно для работы с большими объёмами данных и распределёнными вычислениями.
- PyTorch: Библиотека Python с открытым исходным кодом, разработанная Facebook (Meta), которая также используется для глубокого обучения. Она отличается гибкостью и динамическим построением вычислительного графа, что делает её удобной для исследования и разработки новых архитектур нейронных сетей. Также обеспечивает хорошую производительность.
- Глубокое обучение (Deep Learning): Подмножество машинного обучения, использующее искусственные нейронные сети с множеством

слоёв (глубокие сети) для анализа данных. Эти сети способны обучаться сложным нелинейным зависимостям в данных.

- Нейронная сеть (Neural Network): Вычислительная модель, вдохновлённая структурой и функционированием биологических нейронных сетей. Состоит из слоёв нейронов, которые обрабатывают информацию и передают её дальше по сети.
- Модель (Model): Математическое представление зависимости между входными и выходными данными, полученное в результате обучения алгоритма машинного обучения. Модель используется для предсказания значений на новых, невиданных данных.
- Метрики производительности: Количественные показатели, используемые для оценки качества модели машинного обучения. Для классификации это может быть точность (accuracy), полнота (recall), точность (precision), F1-мера; для регрессии – среднеквадратичная ошибка (MSE), средняя абсолютная ошибка (MAE), R-квадрат.
- Время обучения: Время, необходимое для обучения модели на обучающем наборе данных.
- Препроцессинг данных: Подготовка данных для использования в алгоритмах машинного обучения, включая очистку, нормализацию, кодирование категориальных признаков и разделение данных на обучающую и тестовую выборки.

Методы и подходы к разработке

Выбор наборов данных

- Классификация: Набор данных Iris (классификация трех видов ирисов по четырем признакам). Это небольшой, хорошо понятный набор данных, идеально подходящий для демонстрации основных принципов.
- Регрессия: Набор данных California Housing (предсказание стоимости жилья на основе различных признаков). Более сложный набор данных, который позволит оценить производительность моделей на более объемных и разнообразных данных.

Выбор моделей

- Scikit-learn: Для классификации – логистическая регрессия (простая, но эффективная модель для сравнения). Для регрессии – линейная регрессия. Возможно добавление более сложных моделей (например, SVM, Random Forest) для более полного сравнения.
- TensorFlow & PyTorch: Для обеих задач – простые полносвязные нейронные сети (MLP). Это позволит сравнить производительность глубоких моделей с более простыми моделями scikit-learn. Можно исследовать влияние глубины сети и числа нейронов на производительность.

Предобработка данных

- Нормализация/стандартизация: Для обеспечения справедливого сравнения, данные будут нормализованы или стандартизованы перед обучением моделей. Это особенно важно для TensorFlow и PyTorch, где нейронные сети чувствительны к масштабу входных данных.
- Разделение данных: Данные будут разделены на обучающую и тестовую выборки (например, 70/30 или 80/20). Это необходимо для оценки обобщающей способности моделей.

Обучение моделей

- Гиперпараметрическая настройка: Для каждой модели будут выбраны разумные значения гиперпараметров (например, скорость обучения для нейронных сетей, регуляризация). Для TensorFlow и PyTorch можно использовать методы автоматизированного поиска гиперпараметров (например, GridSearchCV в scikit-learn или Optuna).
- Оценка производительности: Модели будут обучены на обучающей выборке и оценены на тестовой выборке. Метрики производительности будут тщательно измерены и записаны для сравнения.

Сравнение результатов

- Количественное сравнение: Результаты будут сравниваться на основе количественных показателей (accuracy, precision, recall, F1-score для классификации; MSE, MAE, R-squared для регрессии). Будет проведен статистический анализ результатов для определения значимости различий.

- Качественное сравнение: Будет проведено сравнение с точки зрения удобства использования каждой библиотеки. Это включает в себя простоту написания кода, наличие документации, скорость разработки и отладки.

Визуализация

Графики и таблицы будут использоваться для визуализации результатов и упрощения сравнения производительности различных моделей и библиотек.

Этот подход гарантирует, что сравнение будет объективным, воспроизводимым и позволит сделать обоснованные выводы о преимуществах и недостатках каждой библиотеки. Более того, он позволит выделить сильные и слабые стороны каждой библиотеки в контексте конкретных задач.

Обзор популярных библиотек для машинного обучения на Python

Основные – Scikit-learn, TensorFlow, PyTorch, ниже более подробное описание.

Scikit-learn

- Описание: Библиотека, ориентированная на классическое машинное обучение. Она предоставляет широкий спектр алгоритмов, включая регрессию, классификацию, кластеризацию и снижение размерности. Известна своей простотой использования и хорошей документацией.
- Сильные стороны: Легко освоить, эффективна для задач с небольшими и средними объемами данных, отличная для быстрого прототипирования, богатый набор готовых моделей и инструментов.
- Слабые стороны: Не так хорошо подходит для глубокого обучения, может быть недостаточно эффективной для очень больших наборов данных.

TensorFlow

- Описание: Библиотека, разработанная OpenAI, мощный инструмент для глубокого обучения, обработки естественного языка и компьютерного зрения. Поддерживает как eager execution (немедленное выполнение), так и graph execution (выполнение с построением графа вычислений).
- Сильные стороны: Высокая производительность, особенно для больших моделей и больших наборов данных, масштабируемость, широкое сообщество и поддержка, множество готовых моделей и инструментов.
- Слабые стороны: Может иметь более сложный синтаксис по сравнению с scikit-learn, кривая обучения может быть круче.

PyTorch

- Описание: Библиотека, разработанная Facebook (Meta), также широко используется для глубокого обучения. Известна своей гибкостью и динамическим построением вычислительного графа, что упрощает разработку и отладку сложных моделей.
- Сильные стороны: Гибкость, динамический граф вычислений, удобство отладки, хорошее сообщество и поддержка, эффективен для исследований и разработки новых архитектур.
- Слабые стороны: Может быть менее эффективным для задач развертывания по сравнению с TensorFlow (хотя это постоянно улучшается).

Разработка экспериментов

Планирование разработки

Разработка была разделена на несколько основных этапов: Разработка с использованием библиотеки Scikit-learn, разработка с использованием библиотеки TensorFlow, разработка с использованием библиотеки PyTorch

Разработка

Разработка с использованием библиотеки Scikit-learn

```python
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris, fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.metrics import accuracy_score, mean_squared_error, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
```

Рис. 1 - Необходимые импорты для Scikit-learn

```python
def load_and_preprocess_data(dataset_name):  1 usage  new *
    if dataset_name == "iris":
        data = load_iris()
        X, y = data.data, data.target
    elif dataset_name == "housing":
        data = fetch_california_housing()
        X, y = data.data, data.target
    else:
        raise ValueError("Некорректное имя набора данных")

    X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2, random_state=42)
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
    return X_train, X_test, y_train, y_test
```

Рис. 2 - Функция загрузки и предобработки данных для Scikit-learn

```python
def train_and_evaluate(model, X_train, y_train, X_test, y_test):  2 usages  new *
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    if isinstance(model, (LogisticRegression, DecisionTreeClassifier, RandomForestClassifier)):
        accuracy = accuracy_score(y_test, y_pred)
        report = classification_report(y_test, y_pred)
        return accuracy, report
    elif isinstance(model, (LinearRegression, DecisionTreeRegressor, RandomForestRegressor)):
        mse = mean_squared_error(y_test, y_pred)
        return mse, None
    else:
        return None, None
```

Рис. 3 - Функция обучения и оценки моделей для Scikit-learn

```python
def run_experiment(dataset_name):  2 usages  new *
    X_train, X_test, y_train, y_test = load_and_preprocess_data(dataset_name)
    results = {}
    if dataset_name == "iris":
        models = {
            "Logistic Regression": LogisticRegression(),
            "Decision Tree": DecisionTreeClassifier(),
            "Random Forest": RandomForestClassifier()
        }
        for name, model in models.items():
            accuracy, report = train_and_evaluate(model, X_train, y_train, X_test, y_test)
            results[name] = {"accuracy": accuracy, "report": report}
    elif dataset_name == "housing":
        models = {
            "Linear Regression": LinearRegression(),
            "Decision Tree": DecisionTreeRegressor(),
            "Random Forest": RandomForestRegressor()
        }
        for name, model in models.items():
            mse, _ = train_and_evaluate(model, X_train, y_train, X_test, y_test)
            results[name] = {"mse": mse}

    return results
```

Рис. 4 - Функция выполнения эксперимента для Scikit-learn

```python
def visualize_results(results, dataset_name):  2 usages  new *
    if dataset_name == "iris":
        metric = "accuracy"
    else:
        metric = "mse"

    data = {
        "model": list(results.keys()),
        metric: [results[model][metric] for model in results]
    }
    df = pd.DataFrame(data)
    plt.figure(figsize=(10, 6))
    sns.barplot(x="model", y=metric, data=df)
    plt.title(f"{dataset_name.capitalize()} - Model Performance")
    plt.show()
    for model, result in results.items():
        if "report" in result:
            print(f"\nClassification Report for {model} ({dataset_name}):\n{result['report']}")
```

Рис. 5 - Функция визуализации результата для Scikit-learn

```python
iris_results = run_experiment("iris")
housing_results = run_experiment("housing")
visualize_results(iris_results, dataset_name: "iris")
visualize_results(housing_results, dataset_name: "housing")
```

Рис. 6 - Запуск экспериментов и визуализация результатов для Scikit-learn

Разработка с использованием библиотеки TensorFlow

```
import tensorflow as tf
import numpy as np
from sklearn.datasets import load_iris, fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns
import time
import pandas as pd
```

Рис. 7 - Необходимые импорты для TensorFlow

```
def load_and_preprocess(dataset_name):  1 usage  new *
    if dataset_name == "iris":
        data = load_iris()
        X, y = data.data, data.target
        y = tf.keras.utils.to_categorical(y, num_classes=3)  # One-hot encoding for multi-class
    elif dataset_name == "housing":
        data = fetch_california_housing()
        X, y = data.data, data.target
    else:
        raise ValueError("Некорректное имя набора данных")

    X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2, random_state=42)
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
    return X_train, X_test, y_train, y_test
```

Рис. 8 - Функция загрузки и предобработки данных для TensorFlow

```python
def create_and_train_model(X_train, y_train, dataset_name, epochs=100, verbose=0):  1 usage  new *
    start_time = time.time()
    if dataset_name == "iris":
        model = tf.keras.models.Sequential([
            tf.keras.layers.Dense(10, activation='relu', input_shape=(X_train.shape[1],)),
            tf.keras.layers.Dense(3, activation='softmax')
        ])
        model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
        model.fit(X_train, y_train, epochs=epochs, verbose=verbose,
                  validation_split=0.1,
                  callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)])

    elif dataset_name == "housing":
        model = tf.keras.models.Sequential([
            tf.keras.layers.Dense(10, activation='relu', input_shape=(X_train.shape[1],)),
            tf.keras.layers.Dense(1)
        ])
        model.compile(optimizer='adam', loss='mse', metrics=['mae'])
        model.fit(X_train, y_train, epochs=epochs, verbose=verbose,
                  validation_split=0.1,
                  callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)])
    else:
        raise ValueError("Некорректное имя набора данных")
    end_time = time.time()
    training_time = end_time - start_time
    return model, training_time
```

Рис. 9 - Функция создания и обучения модели для TensorFlow

```python
def evaluate_model(model, X_test, y_test, dataset_name):  2 usages  new *
    if dataset_name == "iris":
        _, accuracy = model.evaluate(X_test, y_test, verbose=0)
        return accuracy
    elif dataset_name == "housing":
        mse, mae = model.evaluate(X_test, y_test, verbose=0)
        return mse, mae
    else:
        raise ValueError("Некорректное имя набора данных")
```

Рис. 10 - Функция оценки модели для TensorFlow

```python
def run_experiment(dataset_name, epochs=100, verbose=0):  2 usages  new *
    X_train, X_test, y_train, y_test = load_and_preprocess(dataset_name)
    model, training_time = create_and_train_model(X_train, y_train, dataset_name, epochs=epochs, verbose=verbose)
    if dataset_name == "housing":
        mse, mae = evaluate_model(model, X_test, y_test, dataset_name)
        return mse, mae, training_time
    else:
        accuracy = evaluate_model(model, X_test, y_test, dataset_name)
        return accuracy, training_time
```

Рис. 11 - Функция выполнения эксперимента для TensorFlow

```python
def visualize_results(results):  1 usage  new *
    df = pd.DataFrame(results, index=[0])
    df = df.T.rename(columns={0: 'Значение'})
    plt.figure(figsize=(10, 6))
    sns.barplot(x=df.index, y='Значение', data=df)
    plt.xticks(rotation=45, ha='right')
    plt.title("Производительность модели")
    plt.tight_layout()
    plt.show()
```

Рис. 12 - Функция визуализации результатов для TensorFlow

```python
iris_accuracy, iris_training_time = run_experiment( dataset_name: "iris", epochs=100, verbose=0)
housing_mse, housing_mae, housing_training_time = run_experiment( dataset_name: "housing", epochs=50, verbose=0)

results = {
    "Точность Iris": iris_accuracy,
    "MSE Housing": housing_mse,
    "MAE Housing": housing_mae,
    "Время обучения Iris": iris_training_time,
    "Время обучения Housing": housing_training_time
}

visualize_results(results)
```

Рис. 13 - Запуск экспериментов и визуализация результатов для TensorFlow

Разработка с использованием библиотеки PyTorch

```python
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.datasets import load_iris, fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
from torch.utils.data import DataLoader, TensorDataset
import time
```

Рис. 14 - Необходимые импорты для PyTorch

```python
def load_and_preprocess(dataset_name):  1 usage  new *
    if dataset_name == "iris":
        data = load_iris()
        X, y = data.data, data.target
        y = y.astype(np.int64)
        y = np.array(y, dtype=np.int64) #Дополнительная проверка для принудительного преобразования

    elif dataset_name == "housing":
        data = fetch_california_housing()
        X, y = data.data, data.target
    else:
        raise ValueError("Некорректное имя набора данных")

    X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2, random_state=42)

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    return X_train, X_test, y_train, y_test
```

Рис. 15 - Функция загрузки и предобработки данных для PyTorch

```python
class MLP(nn.Module):  2 usages  new *
    def __init__(self, input_size, hidden_size, output_size):  new *
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):  new *
        x = x.float()
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

Рис. 16 - Определение модели MLP для PyTorch

```python
def train_pytorch(model, train_loader, criterion, optimizer, epochs=100):  1 usage  new *
    start_time = time.time()
    model.train()
    for epoch in range(epochs):
        for inputs, labels in train_loader:
            optimizer.zero_grad()
            outputs = model(inputs.float())
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
    end_time = time.time()
    training_time = end_time - start_time
    return training_time
```

Рис. 17 - Функция обучения для PyTorch

```python
def evaluate_pytorch(model, test_loader, dataset_name):  1 usage  new *
    model.eval()
    y_pred = []
    y_true = []
    with torch.no_grad():
        for inputs, labels in test_loader:
            outputs = model(inputs.float())
            if dataset_name == "iris":
                _, predicted = torch.max(outputs, 1)
                y_pred.extend(predicted.cpu().numpy())
                y_true.extend(labels.cpu().numpy())
            elif dataset_name == "housing":
                y_pred.extend(outputs.cpu().numpy().flatten())
                y_true.extend(labels.cpu().numpy().flatten())
            else:
                raise ValueError("Некорректное имя набора данных")

    if dataset_name == "iris":
        accuracy = accuracy_score(y_true, y_pred)
        return accuracy
    elif dataset_name == "housing":
        mse = mean_squared_error(y_true, y_pred)
        return mse
    else:
        raise ValueError("Некорректное имя набора данных")
```

Рис. 18 - Функция оценки для PyTorch

```python
def run_pytorch_experiment(dataset_name, epochs=100, hidden_size=10):  2 usages  new *
    X_train, X_test, y_train, y_test = load_and_preprocess(dataset_name)
    input_size = X_train.shape[1]

    if dataset_name == "iris":
        output_size = 3
        criterion = nn.CrossEntropyLoss()
        train_data = TensorDataset( *tensors: torch.tensor(X_train).float(), torch.tensor(y_train).long())
        test_data = TensorDataset( *tensors: torch.tensor(X_test).float(), torch.tensor(y_test).long())
    elif dataset_name == "housing":
        output_size = 1
        criterion = nn.MSELoss()
        train_data = TensorDataset( *tensors: torch.tensor(X_train).float(), torch.tensor(y_train).float())
        test_data = TensorDataset( *tensors: torch.tensor(X_test).float(), torch.tensor(y_test).float())
    else:
        raise ValueError("Некорректное имя набора данных")

    model = MLP(input_size, hidden_size, output_size)
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    train_loader = DataLoader(train_data, batch_size=32, shuffle=True)
    test_loader = DataLoader(test_data, batch_size=32)

    training_time = train_pytorch(model, train_loader, criterion, optimizer, epochs=epochs)
    metric = evaluate_pytorch(model, test_loader, dataset_name)
    return metric, training_time
```

Рис. 19 - Функция выполнения эксперимента для PyTorch

```python
def visualize_results(results):  1 usage  new *
    df = pd.DataFrame([results])
    print(df)

    melted_df = df.melt(var_name='Metric', value_name='Value')

    plt.figure(figsize=(10, 6))
    sns.barplot(x='Metric', y='Value', data=melted_df)
    plt.title('Результаты эксперимента PyTorch')
    plt.xticks(rotation=45, ha='right')
    plt.ylabel('Значение метрики')
    plt.tight_layout()
    plt.show()
```

Рис. 20 - Функция визуализации для PyTorch

```python
iris_metric, iris_time = run_pytorch_experiment( dataset_name: "iris", epochs=100)
housing_metric, housing_time = run_pytorch_experiment( dataset_name: "housing", epochs=50)

results = {
    "Точность Iris": iris_metric,
    "MSE Housing": housing_metric,
    "Время обучения Iris": iris_time,
    "Время обучения Housing": housing_time
}

visualize_results(results)
```

Рис. 21 - Запуск экспериментов и визуализации для PyTorch

Анализ и интерпретация результатов

Результаты экспериментов

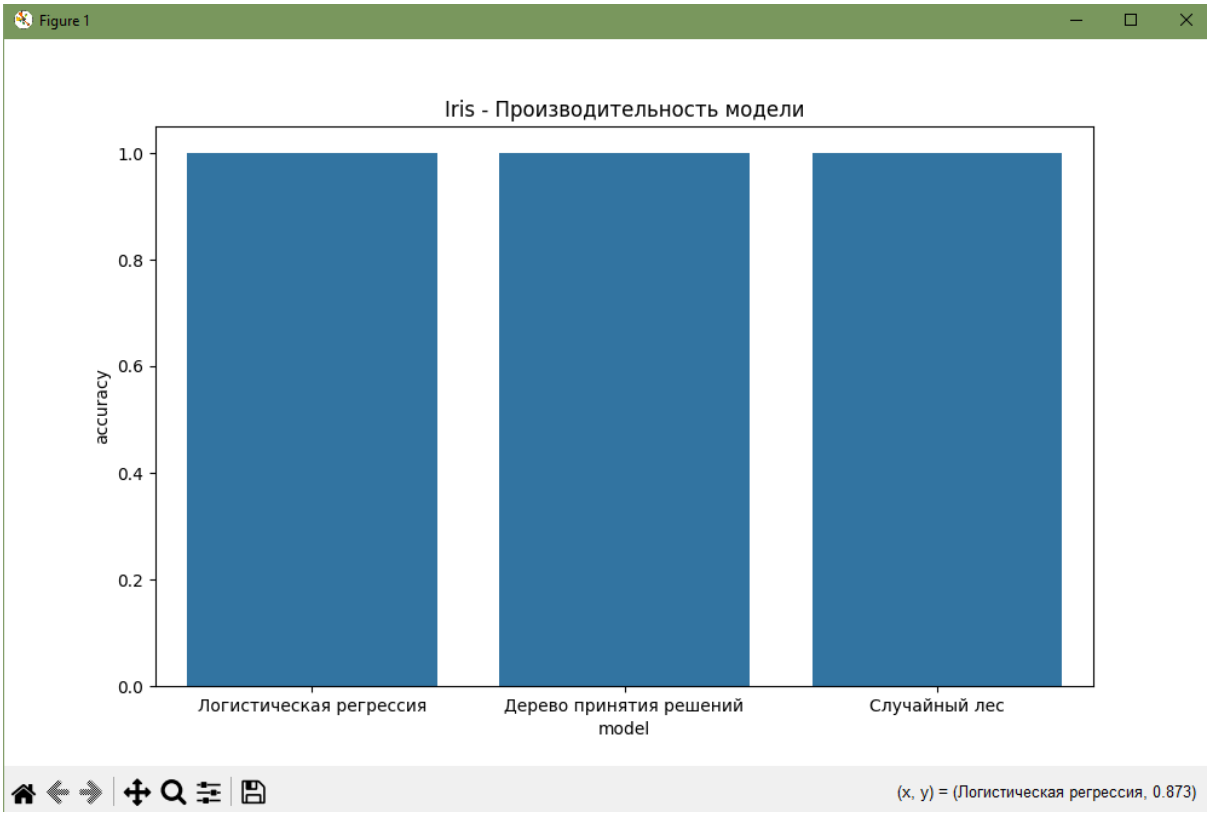

Рис. 22 - Результат эксперимента Iris для Scikit-learn

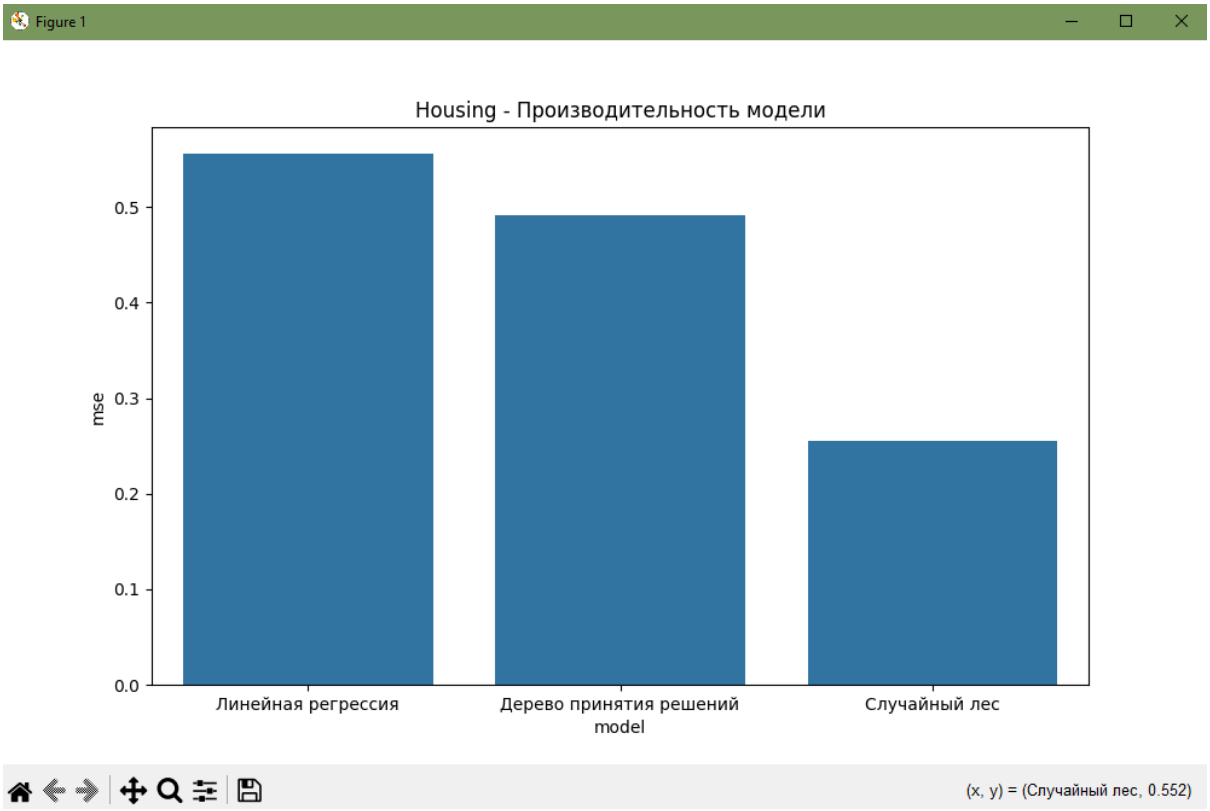

Рис. 23 - Результат эксперимента Housing для Scikit-learn

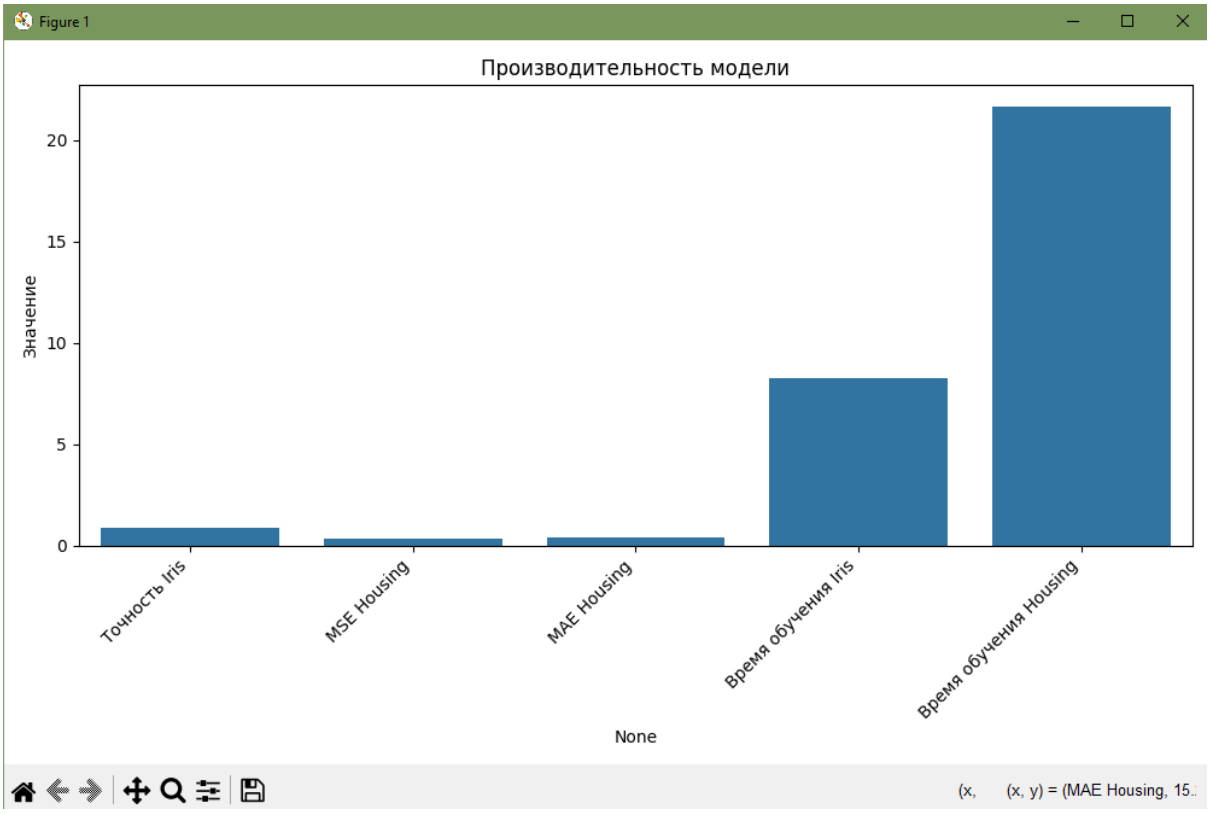

Рис. 24 - Результат эксперимента для TensorFlow

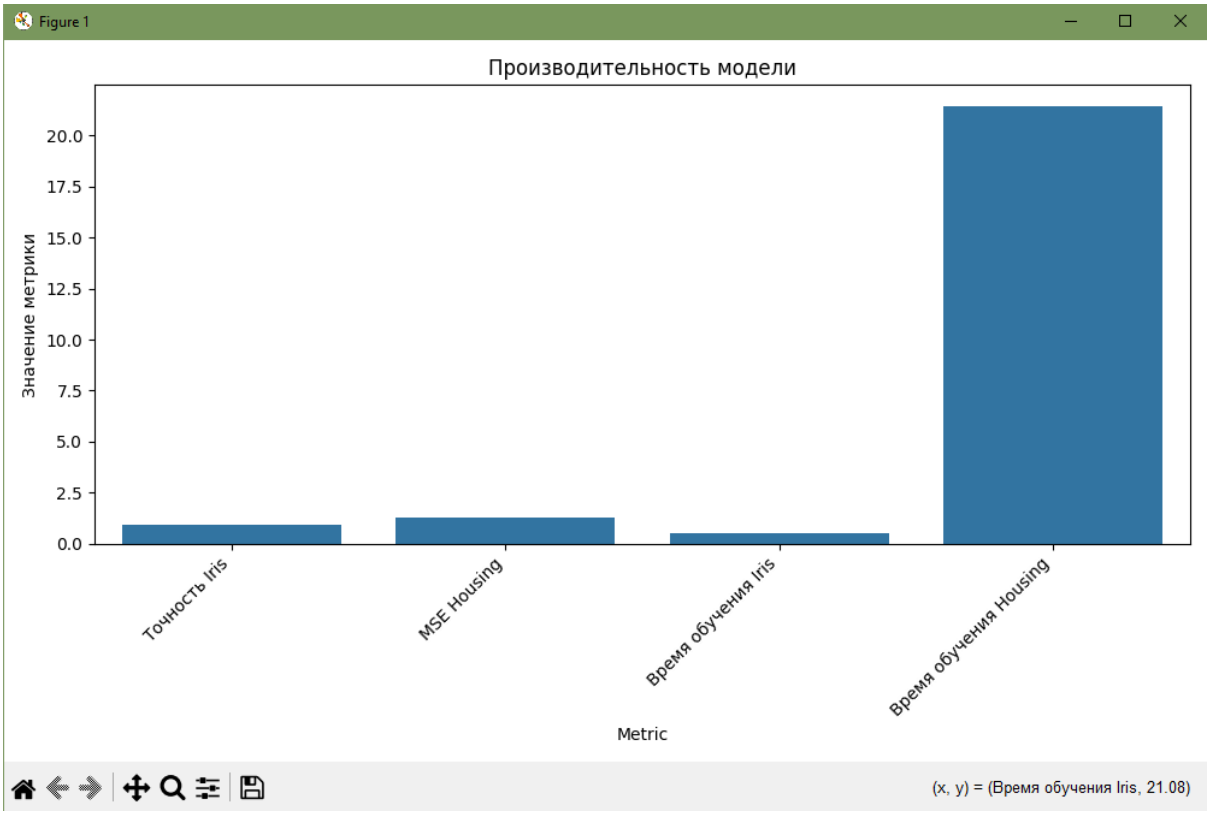

Рис. 25 - Результат эксперимента для PyTorch

Сравнение библиотек

	Scikit-learn	**TensorFlow**	**PyTorch**
Основное направление	Классические ML	Глубокое обучение	Глубокое обучение
Простота	Высокая	Низкая	Средняя
Производительность	Низкая	Высокая	Высокая
Гибкость	Низкая	Средняя	Высокая
Масштабируемость	Низкая	Высокая	Средняя
Обучение	Легко	Сложно	Средняя
Отладка	Легко	Сложно	Средняя

Интерпретация результатов

Scikit-learn: Идеально подходит для быстрого прототипирования и задач, где не требуется глубокое обучение.

TensorFlow: Лучший выбор для больших проектов с глубоким обучением, требующих высокой производительности и масштабируемости.

PyTorch: Прекрасный вариант для исследовательских задач, разработки новых моделей и гибкого прототипирования.

В итоге, выбор библиотеки зависит от конкретной задачи и ваших потребностей. Для начинающих, scikit-learn — отличный выбор для начала знакомства с машинным обучением. Для более сложных задач глубокого обучения, TensorFlow или PyTorch будут более подходящими.

Заключение

Обзор выполненной работы

В данной работе проведено сравнение трёх популярных библиотек для машинного обучения: scikit-learn, TensorFlow и PyTorch. Анализ показал, что выбор библиотеки зависит от конкретной задачи и требуемых характеристик.

Scikit-learn показала себя эффективной и удобной для задач, не требующих глубокого обучения, таких как классификация, регрессия и кластеризация. Быстрое прототипирование и простота использования делают её идеальным инструментом для начинающих и для задач с ограниченным объёмом данных. Однако, при работе с большими наборами данных и сложными моделями её производительность заметно уступает TensorFlow и PyTorch.

TensorFlow и PyTorch, специализирующиеся на глубоком обучении, продемонстрировали значительно большую производительность при обработке больших объёмов данных и сложных архитектур нейронных сетей. TensorFlow выделяется своей масштабируемостью и возможностями распределённых вычислений, что делает его подходящим для крупных проектов и задач, требующих высоких вычислительных ресурсов. PyTorch, в свою очередь, обладает более гибким и интуитивно понятным API, что способствует ускоренной разработке и отладке.

Результаты сравнения подтвердили, что Scikit-learn предпочтительнее для решения задач классического машинного обучения, в то время как TensorFlow и PyTorch являются незаменимыми инструментами для задач глубокого обучения, особенно при работе с большими данными и сложными моделями.

Для задач, где необходима высокая точность и производительность на больших наборах данных, рекомендуется использовать TensorFlow или PyTorch. В случае задач, не требующих глубокого обучения, scikit-learn

окажется более эффективным и удобным решением. Выбор конкретной библиотеки зависит от специфики задачи, доступных ресурсов и целей проекта.

Дальнейшие планы

В дальнейшем, можно расширить исследование, сравнивая библиотеки по скорости обучения различных нейронных сетей на конкретных наборах данных, а также рассмотреть оптимизацию производительности в рамках каждой библиотеки.