

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРАКТИЧЕСКИМ РАБОТАМ

«Алгоритмы обработки и анализа данных»

Направление подготовки 09.03.04 «Программная инженерия»
Направленность (профиль) «Инженерия сложных программных систем»

Квалификация выпускника: бакалавр

Ставрополь
2026

ВВЕДЕНИЕ

Настоящие методические указания предназначены для выполнения лабораторных работ по дисциплине «Алгоритмы обработки и анализа данных».

Цель дисциплины: сформировать у студентов инженерные компетенции в области применения, оценки, оптимизации и интеграции алгоритмов анализа данных в программные продукты, опираясь на знания, полученные в курсе «Системы искусственного интеллекта» (модели ML, нейронные сети, NLP, CV), с акцентом на практическую реализацию, интеграцию с веб-системами и подготовку данных.

Задачи дисциплины:

1. Систематизация и углубление знаний об алгоритмах анализа данных (кластеризация, классификация, регрессия, снижение размерности) на уровне выбора и настройки под реальные задачи.
2. Инженерная оценка качества моделей с использованием продвинутых метрик (PR-AUC, логистическая потеря, квантильные потери, бизнес-метрики).
3. Оптимизация гиперпараметров масштабируемыми методами (GridSearch, RandomizedSearch, Optuna, Bayesian Optimization).
4. Построение пайплайнов предобработки и обучения с использованием Pipeline и ColumnTransformer из scikit-learn для воспроизводимости.
5. Интеграция моделей в веб-приложения (FastAPI, Docker, Streamlit) с учётом требований к задержке, пропускной способности и версионированию.
6. Мониторинг и управление моделями в production (MLflow, дрейф данных и концептуальный дрейф, обнаружение аномалий в предсказаниях).
7. Интерпретация моделей с помощью LIME, SHAP, PDP (частичных зависимостей) для объяснимого ИИ (XAI).
8. A/B-тестирование и роллбэк моделей в production-среде.

Данные методические указания включают 14 лабораторных работ, которые последовательно проводят студента от базовых принципов построения воспроизводимых пайплайнов до создания полноценного MLOps-конвейера с оркестрацией в Apache Airflow, канареечным деплоем и мониторингом дрейфа. Каждая работа содержит чётко сформулированные цель и задачи, теоретическое обоснование, детальный пример практического выполнения, индивидуальные варианты заданий (15 вариантов), требования к отчёту и контрольные вопросы.

Лабораторная работа №1.

Построение воспроизводимого пайплайна с Pipeline

Цель работы: Сформировать практический навык построения единого воспроизводимого конвейера предобработки данных и обучения модели с использованием Pipeline и ColumnTransformer библиотеки scikit-learn. Научиться разделять типы признаков (числовые и категориальные), применять к ним соответствующие преобразования и сохранять готовый пайплайн для последующего использования в production-среде.

Задачи:

- Загрузить и проанализировать структуру исходного датасета (смешанные типы признаков).
- Разделить признаки на числовые и категориальные.
- Построить ColumnTransformer с применением StandardScaler для числовых и OneHotEncoder для категориальных признаков.
- Создать полный Pipeline из двух стадий: предобработка (трансформер) и классификатор/регрессор (на выбор).
- Обучить пайплайн на обучающей выборке и вычислить качество на тестовой.
- Сохранить обученный пайплайн в файл с помощью joblib.
- Загрузить сохранённый пайплайн и выполнить предсказание на новых данных, убедившись в воспроизводимости результата.

Теоретическая часть

1. Проблема утечки данных (data leakage)

При выполнении предобработки (масштабирование, кодирование категорий, заполнение пропусков) важно не применять глобальные параметры (например, среднее и стандартное отклонение, вычисленные по всей выборке). Правильный подход: вычислять параметры преобразований только на

обучающих данных, а затем применять те же параметры к тестовым и новым данным. Ручное выполнение этой процедуры чревато ошибками.

2. Pipeline

Pipeline из `sklearn.pipeline` последовательно связывает несколько этапов обработки. На вход подаётся список пар (имя_шага, объект_трансформера_или_модели). При вызове `fit()` каждый трансформер обучается на выходе предыдущего, а при `predict()` — только преобразует. Это гарантирует, что при кросс-валидации или получении новых данных преобразования будут выполняться корректно.

3. ColumnTransformer

Применяется, когда разные столбцы таблицы требуют различных преобразований. Например:

- числовые → `StandardScaler` (нормировка с нулевым средним и единичной дисперсией);
- категориальные → `OneHotEncoder` (бинарные фиктивные переменные).

4. Важность воспроизводимости

В production-среде модель и все шаги предобработки должны быть сохранены в одном объекте. Это позволяет избежать рассинхронизации между обучением и инференсом. `joblib` эффективнее `pickle` для объектов, содержащих большие массивы `numpy`.

5. Сохранение и загрузка пайплайна

```
python
import joblib
joblib.dump(pipeline, 'model_pipeline.joblib')
loaded_pipeline = joblib.load('model_pipeline.joblib').
```

Пример практического выполнения

Исходные данные:

Датасет titanic (выжившие пассажиры). Признаки: Age (числ.), Fare (числ.), Sex (катег.), Embarked (катег.). Целевая переменная: Survived (бинарная).

Код примера:

```
python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_auc_score
import joblib

# 1. Загрузка данных
df = pd.read_csv('titanic.csv')
X = df[['Age', 'Fare', 'Sex', 'Embarked']]
y = df['Survived']

# 2. Простая обработка пропусков (для примера)
X['Age'] = X['Age'].fillna(X['Age'].median())
X['Embarked'] = X['Embarked'].fillna('S')

# 3. Разделение
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 4. Определение типов признаков
numeric_features = ['Age', 'Fare']
```

```
categorical_features = ['Sex', 'Embarked']
```

```
# 5. Трансформеры
```

```
numeric_transformer = StandardScaler()
```

```
categorical_transformer = OneHotEncoder(drop='first',  
handle_unknown='ignore')
```

```
# 6. ColumnTransformer
```

```
preprocessor = ColumnTransformer(  
    transformers=[  
        ('num', numeric_transformer, numeric_features),  
        ('cat', categorical_transformer, categorical_features)  
    ]  
)
```

```
# 7. Полный пайплайн
```

```
pipeline = Pipeline(steps=[  
    ('preprocessor', preprocessor),  
    ('classifier', RandomForestClassifier(n_estimators=100,  
random_state=42))  
)
```

```
# 8. Обучение
```

```
pipeline.fit(X_train, y_train)
```

```
# 9. Оценка
```

```
y_pred = pipeline.predict(X_test)  
y_proba = pipeline.predict_proba(X_test)[:, 1]  
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")  
print(f"ROC-AUC: {roc_auc_score(y_test, y_proba):.4f}")
```

10. Сохранение

```
joblib.dump(pipeline, 'titanic_pipeline.joblib')
```

11. Загрузка и предсказание новых данных

```
loaded_pipeline = joblib.load('titanic_pipeline.joblib')
```

```
new_data = pd.DataFrame({
```

```
    'Age': [30, 50],
```

```
    'Fare': [50, 200],
```

```
    'Sex': ['male', 'female'],
```

```
    'Embarked': ['S', 'C']
```

```
})
```

```
predictions = loaded_pipeline.predict(new_data)
```

```
print("Предсказания для новых данных:", predictions)
```

Результат выполнения (пример вывода):

text

Accuracy: 0.8212

ROC-AUC: 0.8745

Предсказания для новых данных: [0 1]

Индивидуальные задания

Каждый вариант использует свой датасет (указан источник или описание) и собственный набор признаков. Студент должен:

1. Загрузить данные.
2. Выполнить минимальную обработку пропусков (заполнить медианой/модой).
3. Построить ColumnTransformer, применив StandardScaler к числовым признакам и OneHotEncoder к категориальным.

4. В качестве модели использовать:
 - для бинарной классификации – LogisticRegression или RandomForestClassifier;
 - для многоклассовой – RandomForestClassifier;
 - для регрессии – RandomForestRegressor.
5. Вычислить метрику: accuracy / ROC-AUC / R^2 (в зависимости от задачи).
6. Сохранить пайплайн и загрузить, проверив предсказание на 2–3 новых примерах.

Примечание:

- Для вариантов без явных категориальных признаков используйте только числовой трансформер.
- Для вариантов с многоклассовой классификацией используйте RandomForestClassifier и выведите classification_report.
- Датасеты, не входящие в sklearn, студент скачивает с Kaggle или по ссылке (ссылка указывается в задании).

Вариант	Датасет / описание	Числовые признаки	Категориальные признаки	Целевая переменная
1	<code>sklearn.datasets.load_breast_cancer</code>	mean radius, mean texture, mean perimeter	(нет)	malignant (0/1)
2	titanic (kaggle)	Age, Fare, SibSp, Parch	Sex, Embarked	Survived
3	housing (калифорнийское жильё)	MedInc, HouseAge, AveRooms, AveOccup	OceanProximity (текст)	MedHouseVal (регрессия)
4	adult (перепись населения)	Age, EducationNum, CapitalGain, HoursPerWeek	WorkClass, Sex, Race, NativeCountry	Income (>50K)

5	wine (классы вин)	Alcohol, MalicAcid, Ash, ColorIntensity	(нет)	target (3 класса)
6	churn (отток клиентов, телеком)	Tenure, MonthlyCharges, TotalCharges	Gender, PaymentMethod, Contract	Churn
7	sklearn.datasets.fetch_california_housing	MedInc, HouseAge, AveRooms, Population	(нет)	MedHouseVal
8	student_performance (GPA)	StudyTime, Absences, PreviousScores	Gender, ParentEducation, LunchType	GradeClass (0-4)
9	bank_marketing	Age, Balance, Duration, Campaign	Job, Marital, Education, Contact	Deposit
10	credit_risk (German Credit)	Duration, Amount, Age, InstallmentRate	Purpose, Sex, Housing, Job, Savings	Risk (good/bad)
11	flight_delays (авиа, малая выборка)	Distance, DepDelay, TaxiOut	Airline, Origin, Dest	ArrDelay (бинарно: >15 мин)
12	heart_disease (UCI)	Age, Chol, Thalach, Oldpeak, RestBP	Sex, ChestPainType, FastingBS, RestECG	Target (0/1)
13	auto_mpg (регрессия MPG)	Cylinders, Displacement, Horsepower, Weight	ModelYear (как категория), Origin	MPG
14	fraud_detection (синтетический)	TransactionAmount, OldBalance, NewBalance	MerchantCategory, DeviceType, Location	IsFraud
15	house_prices (Ames, малая версия)	LotArea, YearBuilt, TotalBsmntSF, GrLivArea	MSZoning, Street, SaleCondition	SalePrice (регрессия)

Контрольные вопросы

1. Почему нельзя сначала применить `StandardScaler` ко всей выборке, а потом разделить на train/test?
2. Какие преимущества даёт использование `Pipeline` по сравнению с ручной последовательной обработкой?
3. В чём отличие `fit` от `transform` и от `fit_transform`?

4. Как `ColumnTransformer` обрабатывает столбцы, не указанные ни в одном трансформере (параметр `remainder`)?
5. Какую проблему решает параметр `handle_unknown='ignore'` в `OneHotEncoder`?
6. Почему для сохранения моделей в `scikit-learn` рекомендуется использовать `joblib`, а не `pickle`?
7. Что произойдёт, если сохранить только обученный классификатор, а `ColumnTransformer` не сохранять?
8. Как можно добавить в пайплайн заполнение пропусков (импутацию) перед масштабированием? Приведите пример.
9. В чём разница между `Pipeline` и `make_pipeline`?
10. Как загрузить сохранённый пайплайн и получить доступ к отдельному шагу (например, к коэффициентам линейной модели)?

Требования к отчёту

Отчёт оформляется в текстовом редакторе (Microsoft Word, LibreOffice) в соответствии с ГОСТ 7.32-2017. Структура:

1. **Титульный лист** (название вуза, факультет, кафедра, дисциплина, тема работы, ФИО студента, номер группы, вариант, преподаватель, год).
2. **Цель работы** (формулировка из раздела 1).
3. **Задачи работы** (список из раздела 2).
4. **Теоретическое обоснование** (кратко своими словами, 1–2 страницы с основными понятиями).
5. **Практическая часть:**
 - Код программы (листинг с комментариями).
 - Скриншоты процесса выполнения (загрузка данных, этапы обработки, обучение, сохранение).
 - Выводы о качестве модели (метрики).

6. **Результаты выполнения** (таблицы, графики по желанию, значения метрик, предсказания на новых данных).
 7. **Анализ результатов** (что получилось, с какими трудностями столкнулись, как их преодолели).
 8. **Ответы на контрольные вопросы** (полные развёрнутые ответы).
 9. **Вывод** (освоенные навыки, достигнута ли цель, практическая значимость).
 10. **Список литературы** (не менее 3 источников, включая документацию scikit-learn).
1. **Примечание:** Отчёт сдаётся в электронном виде (PDF или DOCX) вместе с файлом пайплайна (.joblib) и исходным кодом (.py или .ipynb).

Критерии оценивания

Оценка	Критерии
Отлично	Работа выполнена полностью, без ошибок. Код соответствует стандартам PEP8, использованы все требуемые компоненты (Pipeline, ColumnTransformer, StandardScaler, OneHotEncoder, joblib). Отчёт структурирован, содержит скриншоты, анализ метрик и ответы на все контрольные вопросы. Реализована вариативная часть (например, добавлен импутер пропусков или разные модели сравнены внутри пайплайна). Защита: уверенные ответы на все вопросы преподавателя.
Хорошо	Работа выполнена полностью, но есть незначительные замечания по оформлению кода или отчёта (отсутствует анализ, нет одного скриншота). Основной функционал работает. Вариативная часть не выполнена. Защита: ответы на большинство вопросов.
Удовлетворительно	Работа выполнена частично: основной функционал работает с ошибками (например, не сохранён пайплайн, или некорректно обработаны категориальные признаки) или не в полном объёме (отсутствует ColumnTransformer, использован

	LabelEncoder вместо OneHotEncoder). Отчёт неполный (нет ответов на вопросы, нет анализа). Студент затрудняется при ответах.
Неудовлетворительно	Работа не представлена, или ключевой функционал не работает (пайплайн не обучен, предсказания неверны), или код скопирован из примера без адаптации под свой вариант. Отсутствует отчёт либо он не соответствует требованиям.

Лабораторная работа №2.

Сравнение 5 моделей на реальном датасете

Цель работы: Провести объективное сравнительное исследование пяти алгоритмов классификации (логистическая регрессия, дерево решений, случайный лес, XGBoost, LightGBM) на реальном датасете. Оценить качество моделей по метрикам ROC-AUC и PR-AUC, измерить время обучения и сформулировать рекомендации по выбору модели для заданной бизнес-задачи.

Задачи:

1. Загрузить и выполнить первичную предобработку реального датасета (обработка пропусков, кодирование категориальных признаков, масштабирование при необходимости).
2. Реализовать кросс-валидацию (не менее 5 фолдов) для каждой из пяти моделей.
3. Для каждой модели вычислить:
 - среднее значение ROC-AUC;
 - среднее значение PR-AUC (площадь под Precision-Recall кривой);
 - среднее время обучения одного фолда (в секундах).
4. Построить сравнительную таблицу и визуализировать результаты (ящики с усами или barplot).
5. Сформулировать вывод о том, какая модель показывает наилучшее качество и какая – оптимальна по соотношению качество/скорость.
6. Зафиксировать параметры моделей (гиперпараметры по умолчанию или подобранные заранее).

Теоретическая часть

1. Кросс-валидация (cross-validation)

Метод оценки обобщающей способности модели, при котором выборка многократно разбивается на обучающую и валидационную части. Стандартный подход – k-fold cross-validation (обычно $k=5$ или 10). Это снижает

риск переобучения и даёт более устойчивые оценки качества по сравнению с одним разбиением train/test.

2. ROC-AUC и PR-AUC

- ROC-AUC (Area Under the Receiver Operating Characteristic curve) – мера разделяющей способности бинарного классификатора. Значение 0.5 соответствует случайному угадыванию, 1.0 – идеальному разделению.

- PR-AUC (Area Under the Precision-Recall curve) – более информативная метрика для несбалансированных классов. Она показывает, насколько хорошо модель выделяет положительный класс при разных порогах.

Для дисбалансных датасетов (например, отток клиентов, мошенничество) PR-AUC предпочтительнее ROC-AUC.

3. Сравнимые модели

Модель	Тип	Преимущества	Недостатки
LogisticRegression	Линейная	Интерпретируемость, быстрота обучения, устойчивость к переобучению	Не учитывает нелинейности
DecisionTree	Дерево решений	Простота, интерпретируемость, не требует масштабирования	Склонность к переобучению
RandomForest	Ансамбль деревьев	Высокая точность, устойчивость, отбор признаков	Медленнее дерева, тяжелее интерпретировать
XGBoost	Градиентный бустинг	Очень высокая точность, регуляризация, обработка пропусков	Много гиперпараметров, медленное обучение на больших данных
LightGBM	Градиентный бустинг (листовой рост)	Быстрее XGBoost на больших данных, эффективная память	Требует настройки, чувствителен к выбросам

4. Метрика времени обучения

Время обучения (fit time) критично в production-сценариях, особенно при необходимости частого переобучения. Замеряется в секундах с помощью `time.perf_counter()` или `timeit`.

Пример практического выполнения

Датасет: bank-additional-full.csv (банковский маркетинг, цель – подписка на депозит).

Целевая переменная: `y` – "yes"/"no".

Исходные признаки: возраст, должность, семейное положение, образование, длительность последнего контакта и др.

```
python
```

```
import pandas as pd
```

```
import numpy as np
```

```
from sklearn.model_selection import cross_val_score, StratifiedKFold
```

```
from sklearn.preprocessing import LabelEncoder, StandardScaler
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
from xgboost import XGBClassifier
```

```
from lightgbm import LGBMClassifier
```

```
from sklearn.metrics import roc_auc_score, average_precision_score, make_scorer
```

```
import time
```

```
# 1. Загрузка данных
```

```
df = pd.read_csv('bank-additional-full.csv', sep=';')
```

```
X = df.drop('y', axis=1)
```

```
y = (df['y'] == 'yes').astype(int)
```

```
# 2. Простейшая предобработка категориальных признаков
```

```
categorical_cols = X.select_dtypes(include=['object']).columns
```

```
for col in categorical_cols:
```

```
    X[col] = LabelEncoder().fit_transform(X[col])
```

```
# Пропусков в данном датасете нет
```

```
# Масштабирование для логистической регрессии (для деревьев  
необязательно, но оставим)
```

```
scaler = StandardScaler()
```

```
X_scaled = scaler.fit_transform(X)
```

```
# 3. Определение моделей
```

```
models = {
```

```
    'LogisticRegression': LogisticRegression(max_iter=1000, random_state=42),
```

```
    'DecisionTree': DecisionTreeClassifier(random_state=42),
```

```
    'RandomForest': RandomForestClassifier(n_estimators=100, random_state=42),
```

```
    'XGBoost': XGBClassifier(eval_metric='logloss', random_state=42),
```

```
    'LightGBM': LGBMClassifier(verbose=-1, random_state=42)
```

```
}
```

4. Настройки кросс-валидации

```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

```
scoring_roc = make_scorer(roc_auc_score)
```

```
scoring_pr = make_scorer(average_precision_score)
```

```
results = []
```

5. Сравнение

```
for name, model in models.items():
```

```
    # ROC-AUC
```

```
    roc_scores = cross_val_score(model, X_scaled, y, cv=cv, scoring='roc_auc')
```

```
    # PR-AUC
```

```
    pr_scores = cross_val_score(model, X_scaled, y, cv=cv, scoring=scoring_pr)
```

```
    # Время обучения
```

```
    start = time.perf_counter()
```

```
    cross_val_score(model, X_scaled, y, cv=cv, scoring='roc_auc')
```

```
    end = time.perf_counter()
```

```
    train_time = (end - start) / cv.get_n_splits()
```

```
    results.append({
```

```
        'Model': name,
```

```
        'ROC-AUC mean': np.mean(roc_scores),
```

```

'ROC-AUC std': np.std(roc_scores),

'PR-AUC mean': np.mean(pr_scores),

'PR-AUC std': np.std(pr_scores),

'Train time (sec)': train_time

})

```

6. Результаты

```

df_results = pd.DataFrame(results)

print(df_results.round(4))

```

Результат выполнения (пример):

Model	ROC-AUC mean	PR-AUC mean	Train time (sec)
LogisticRegression	0.7892	0.5123	0.08
DecisionTree	0.7214	0.4210	0.02
RandomForest	0.8456	0.6125	0.95
XGBoost	0.8512	0.6234	1.20
LightGBM	0.8498	0.6201	0.31

Вывод по примеру: XGBoost и LightGBM показали наилучшее качество (ROC-AUC $\approx$ 0.85), при этом LightGBM обучился в 4 раза быстрее XGBoost. Логистическая регрессия уступила по качеству, но быстра. DecisionTree – самый быстрый, но низкое качество. Рекомендуемая модель для production: LightGBM.

Индивидуальные задания

Каждый вариант получает реальный датасет бинарной классификации (можно загрузить с Kaggle или UCI). Обязательно: провести кросс-валидацию (5 фолдов), сравнить 5 моделей (LogisticRegression, DecisionTree, RandomForest, XGBoost, LightGBM). Вычислить ROC-AUC, PR-AUC, время обучения. Построить таблицу и диаграммы.

Вариант	Датасет	Ссылка / источник	Целевая переменная	Особенность
1	Bank Marketing	UCI	subscribed (yes/no)	большой дисбаланс
2	Credit Card Fraud	Kaggle	Class (0/1)	сильнейший дисбаланс
3	Telco Churn	Kaggle	Churn (Yes/No)	смешанные типы
4	Heart Disease (UCI)	UCI	target (0/1)	малый размер (300)
5	German Credit	UCI	Credit (good/bad)	категориальные признаки
6	Pima Indians Diabetes	Kaggle	Outcome (0/1)	все числовые
7	Employee Attrition	IBM HR Analytics	Attrition (Yes/No)	много признаков

Вариант	Датасет	Ссылка / источник	Целевая переменная	Особенность
8	Default of Credit Card Clients	UCI	default (0/1)	несбалансированный
9	Spambase	UCI	spam (0/1)	много числовых
10	Adult Income	UCI	income (>50K)	дисбаланс, категориальные
11	Breast Cancer Wisconsin	UCI	malignant (0/1)	легко разделяемый
12	Click-Through Rate (CTR)	Criteo sample	clicked (0/1)	большие данные (>1M)
13	News Popularity	UCI	shares binary (порог 1400)	регрессия бинарная ->
14	Amazon Employee Access	Kaggle	ACTION (0/1)	категориальные признаки

Вариант	Датасет	Ссылка / источник	Целевая переменная	Особенность
15	Melanoma Detection	ISIC 2020	target (0/1)	медицинская, изображения → признаки

Примечание:

- Для варианта 15 необходимо предварительно извлечь признаки из изображений (например, с помощью предобученной CNN, но это необязательно – можно использовать готовый датасет метаданных).
- Для больших датасетов (12) допустимо использовать 3-фолдовую кросс-валидацию и уменьшить выборку до 100 000 строк.

Контрольные вопросы

1. Почему ROC-AUC не всегда является лучшей метрикой для несбалансированных классов?
2. Как интерпретировать значение PR-AUC = 0.3 при ROC-AUC = 0.85?
3. В чём отличие между кросс-валидацией и простым разбиением train/test?
4. Почему для кросс-валидации используется StratifiedKFold, а не обычный KFold?
5. Какая модель в эксперименте показала наименьшее время обучения? Почему так?
6. Какие гиперпараметры XGBoost и LightGBM наиболее сильно влияют на переобучение?
7. В каких сценариях следует предпочесть логистическую регрессию градиентному бустингу?

8. Как влияет масштабирование признаков на деревья решений и на логистическую регрессию?
9. Что такое PR-AUC и как она вычисляется по выборке?
10. По результатам вашего варианта: какая модель обеспечивает наилучший компромисс между качеством и скоростью и почему?

Требования к отчёту

Отчёт оформляется аналогично лабораторной работе №1 (ГОСТ 7.32-2017). Дополнительные требования:

- В практической части обязательно привести **таблицу сравнения** всех пяти моделей по трём метрикам (среднее $\pm$ стандартное отклонение).
- Приложить **визуализации** (например, boxplot ROC-AUC по фолдам или barplot средних значений).
- В анализе результатов указать, какая модель показала лучший ROC-AUC, какая – лучший PR-AUC, какая – минимальное время обучения, и предложить выбор модели в зависимости от задачи (высокая точность vs. быстрый инференс vs. интерпретируемость).
- Для вариативной части (оценка «отлично») необходимо выполнить подбор гиперпараметров для XGBoost и LightGBM с помощью GridSearchCV или Optuna и повторить сравнение.

Критерии оценивания

Оценка	Критерии
Отлично	Работа выполнена полностью. Корректно реализована кросс-валидация для 5 моделей, вычислены ROC-AUC, PR-AUC, время обучения. Результаты представлены в виде таблицы и графиков. Сделан обоснованный вывод. Код оптимизирован, использованы <code>StratifiedKFold</code> и <code>make_scorer</code> . Дополнительно выполнен подбор гиперпараметров хотя бы для одной модели бустинга (вариативная часть). Защита: глубокое понимание метрик.
Хорошо	Работа выполнена полностью, но отсутствует визуализация или не вычислен PR-AUC (только ROC-AUC). Либо не проведён статистический анализ (нет std). Вариативная часть не выполнена. Защита: ответы на большинство вопросов.
Удовлетворительно	Сравнение проведено только для 2–3 моделей, либо кросс-валидация заменена одним разбиением, либо метрики вычислены с ошибками (например, PR-AUC посчитана неверно). Отчёт неполный. Студент затрудняется объяснить разницу между ROC-AUC и PR-AUC.
Неудовлетворительно	Работа не представлена, или модели не обучены, или использованы не те алгоритмы, или полностью отсутствуют вычисления метрик. Отчёт не соответствует требованиям.

Лабораторная работа №3.

Оптимизация гиперпараметров с Optuna

Цель работы: Освоить метод байесовской оптимизации гиперпараметров с использованием библиотеки Optuna на примере градиентного бустинга (XGBoost). Научиться настраивать пространство поиска, управлять количеством итераций, анализировать результаты оптимизации с помощью визуализаций (parallel coordinate plot, history plot) и сравнивать эффективность Optuna с традиционным GridSearchCV.

Задачи:

- Загрузить и предобработать реальный датасет для бинарной классификации.
- Реализовать функцию `objective(trial)`, задающую пространство гиперпараметров XGBoost (или LightGBM) для оптимизации.
- Создать Study в Optuna, выполнить оптимизацию с заданным числом итераций (например, 50–100).
- Построить визуализации: историю оптимизации (optimization history), параллельные координаты (parallel coordinate plot), важность гиперпараметров.
- Выполнить GridSearchCV (с ограниченной сеткой) для той же модели и датасета, измерить время выполнения.
- Сравнить результаты Optuna и GridSearchCV по:
 - достигнутому значению целевой метрики (ROC-AUC / PR-AUC);
 - количеству итераций / комбинаций;
 - времени выполнения.
- Сделать вывод о преимуществах байесовской оптимизации.

Теоретическая часть

1. Проблема настройки гиперпараметров

Современные модели машинного обучения (особенно градиентный бустинг) имеют десятки гиперпараметров: `max_depth`, `learning_rate`, `n_estimators`, `subsample`, `colsample_bytree`, `reg_alpha`, `reg_lambda` и др. Ручной перебор трудоёмок, а `GridSearch`/ `RandomSearch` страдают от "проклятия размерности" и не учитывают результаты предыдущих итераций.

2. Байесовская оптимизация

Байесовская оптимизация строит вероятностную суррогатную модель (например, `Gaussian Process` или `TPE` – `Tree-structured Parzen Estimator`), которая предсказывает значение целевой функции для неизвестных комбинаций гиперпараметров. На каждом шаге выбирается следующая комбинация, максимизирующая критерий улучшения (`Expected Improvement`). Это позволяет находить хорошие конфигурации за значительно меньшее число итераций.

3. Optuna

`Optuna` – фреймворк для автоматической оптимизации гиперпараметров с поддержкой:

- Определения пространства поиска через методы `suggest_*` (`suggest_int`, `suggest_float`, `suggest_categorical`).
- `Pruning` (отсечение бесперспективных trial'ов) на основе промежуточных результатов.
- Визуализации (`plot_optimization_history`, `plot_parallel_coordinate`, `plot_param_importances`).
- Интеграции с `XGBoost`, `LightGBM`, `PyTorch` и др.

4. Сравнение с GridSearch

`GridSearchCV` перебирает все комбинации из заданной сетки. При 5 параметрах и 5 значениях каждого – 3125 комбинаций. `Optuna` за 50–100 итераций находит сравнимые или лучшие значения, экономя время.

Пример практического выполнения

Датасет: Breast Cancer Wisconsin (встроенный в sklearn).

Модель: XGBoost (бинарная классификация).

Целевая метрика: ROC-AUC (кросс-валидация 5 фолдов).

```
import optuna

from optuna.samplers import TPESampler

from optuna.visualization import plot_optimization_history, plot_parallel_co
ordinate, plot_param_importances

from sklearn.model_selection import cross_val_score, StratifiedKFold

from sklearn.datasets import load_breast_cancer

from xgboost import XGBClassifier

import numpy as np

import time

from sklearn.model_selection import GridSearchCV
```

1. Данные

```
data = load_breast_cancer()

X, y = data.data, data.target

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

2. Функция цели для Optuna

```
def objective(trial):

    params = {

        'n_estimators': trial.suggest_int('n_estimators', 50, 300),

        'max_depth': trial.suggest_int('max_depth', 3, 10),

        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.3, log=True),

        'subsample': trial.suggest_float('subsample', 0.6, 1.0),

        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.6, 1.0),

        'reg_alpha': trial.suggest_float('reg_alpha', 1e-3, 10.0, log=True),

        'reg_lambda': trial.suggest_float('reg_lambda', 1e-3, 10.0, log=True),

        'min_child_weight': trial.suggest_int('min_child_weight', 1, 10),
```

```

    'eval_metric': 'logloss',
    'use_label_encoder': False,
    'random_state': 42
}
model = XGBClassifier(**params)
score = cross_val_score(model, X, y, cv=cv, scoring='roc_auc').mean()
return score

```

3. Запуск Optuna

```

start_optuna = time.time()
study = optuna.create_study(direction='maximize', sampler=TPESampler(see
d=42))
study.optimize(objective, n_trials=50, show_progress_bar=True)
time_optuna = time.time() - start_optuna

print(f'Лучшее значение ROC-AUC: {study.best_value:.4f}')
print(f'Лучшие гиперпараметры: {study.best_params}')
print(f'Время Optuna: {time_optuna:.2f} сек")

```

4. Визуализации

```

plot_optimization_history(study).show()
plot_parallel_coordinate(study).show()
plot_param_importances(study).show()

```

5. Сравнение с GridSearchCV

```

param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [4, 6, 8],
    'learning_rate': [0.05, 0.1, 0.2],
}

```

```

xgb = XGBClassifier(eval_metric='logloss', use_label_encoder=False, random_state=42)

grid = GridSearchCV(xgb, param_grid, cv=cv, scoring='roc_auc', n_jobs=-1)
start_grid = time.time()
grid.fit(X, y)
time_grid = time.time() - start_grid

print(f'GridSearchCV best score: {grid.best_score_:.4f}')
print(f'GridSearchCV best params: {grid.best_params_}')
print(f'Время GridSearchCV: {time_grid:.2f} сек')
print(f'Количество комбинаций GridSearch: {len(param_grid['n_estimators']) * len(param_grid['max_depth']) * len(param_grid['learning_rate'])}')
print(f'Количество trials Optuna: 50')

```

Результат выполнения (пример):

text

Лучшее значение ROC-AUC: 0.9842

Лучшие гиперпараметры: {'n_estimators': 278, 'max_depth': 6, 'learning_rate': 0.087, 'subsample': 0.85, 'colsample_bytree': 0.92, 'reg_alpha': 0.023, 'reg_lambda': 1.45, 'min_child_weight': 3}

Время Optuna: 48.3 сек

GridSearchCV best score: 0.9789

GridSearchCV best params: {'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 200}

Время GridSearchCV: 24.5 сек

Количество комбинаций GridSearch: 18

Вывод по примеру: Optuna за 50 итераций нашел более качественную конфигурацию (ROC-AUC 0.9842 против 0.9789), но потребовал больше

времени (48.3 сек против 24.5 сек) из-за большего числа обученных моделей. Однако при расширении сетки GridSearch время выросло бы экспоненциально, в то время как Optuna масштабируется линейно.

Индивидуальные задания

Для каждого варианта необходимо выполнить оптимизацию гиперпараметров XGBoost (или LightGBM – для вариантов, отмеченных *) с помощью Optuna и сравнить с GridSearch (или RandomizedSearch). Использовать кросс-валидацию (5 фолдов) и метрику **ROC-AUC** (или PR-AUC для сильно несбалансированных датасетов).

Вариант	Датасет	Модель	Число trials Optuna	Особенность
1	Breast Cancer Wisconsin	XGBoost	30	лёгкий
2	Pima Indians Diabetes	XGBoost	50	дисбаланс 35%
3	Bank Marketing (binary)	XGBoost	80	сильный дисбаланс
4	Credit Card Fraud	LightGBM*	100	очень сильный дисбаланс
5	Telco Churn	XGBoost	50	категориальные признаки
6	Heart Disease (UCI)	XGBoost	30	малый датасет

Вариант	Датасет	Модель	Число trials Optuna	Особенность
7	Default of Credit Card	LightGBM*	60	дисбаланс
8	Spambase	XGBoost	40	все числовые
9	Adult Income	XGBoost	80	большой
10	German Credit	LightGBM*	50	категориальные
11	Employee Attrition	XGBoost	50	~20 признаков
12	News Popularity (бинаризация)	XGBoost	70	регрессия → классификация
13	Amazon Employee Access	LightGBM*	60	много категорий
14	Melanoma (метаданные)	XGBoost	50	медицинская
15	Click-Through Rate (sample 100k)	LightGBM*	100	большие данные

Дополнительные требования:

- Для LightGBM в функции `objective` использовать `lgbm_params = {'boosting_type': 'gbdt', 'verbose': -1, 'random_state': 42}` и

метод `suggest_*` для `num_leaves`, `max_depth`, `learning_rate`, `feature_fraction`, `bagging_fraction`, `reg_alpha`, `reg_lambda`.

- Построить визуализации: **history plot**, **parallel coordinate**, **param importances**.

Сравнить с `GridSearchCV` (минимум 15–20 комбинаций). Если `GridSearch` занимает более 10 минут, разрешено использовать `RandomizedSearchCV` с 20 итерациями.

Контрольные вопросы

1. В чём заключается основное отличие байесовской оптимизации от случайного поиска?
2. Как Optuna учитывает результаты предыдущих trial'ов при выборе следующей комбинации?
3. Что такое TPE-сэмплер (Tree-structured Parzen Estimator) и когда его следует использовать?
4. Как работает механизм pruning в Optuna? Приведите пример для XGBoost.
5. Какой график показывает, насколько каждый гиперпараметр влияет на целевую метрику?
6. Почему для `learning_rate` в Optuna часто используют `log=True`?
7. В каких случаях Optuna может оказаться медленнее `GridSearchCV`?
8. Как задать в Optuna условие, чтобы при `max_depth < 0` не использовать `min_child_weight`?
9. Как сохранить и восстановить study Optuna для продолжения оптимизации?
10. По результатам вашего варианта: какая комбинация гиперпараметров оказалась лучшей и на сколько процентов она превзошла baseline (значения по умолчанию)?

Требования к отчёту

Отчёт оформляется по ГОСТ 7.32-2017. Содержит:

1. Титульный лист.
2. Цель и задачи работы.
3. Теоретическое обоснование (байесовская оптимизация, Optuna, сравнение с GridSearch).
4. Практическая часть:
 - код функции `objective` с комментариями;
 - создание `study` и запуск оптимизации;
 - код для `GridSearchCV` (или `RandomizedSearchCV`);
 - скриншоты визуализаций Optuna.
5. Результаты выполнения:
 - таблица сравнения (лучшая метрика, время, количество итераций/комбинаций).
6. Анализ результатов (почему Optuna нашёл лучше/хуже, какие гиперпараметры оказались важными).
7. Ответы на контрольные вопросы.
8. Вывод (освоенные навыки).
9. Список литературы.

Вариативная часть для оценки «отлично»:

- Добавить `pruning` с помощью `optuna.integration.XGBoostPruningCallback`.
- Провести оптимизацию сразу для двух метрик (ROC-AUC и `logloss`) с многокритериальной оптимизацией (MOO).
- Сравнить три метода: Optuna, RandomSearch, GridSearch.

Критерии оценивания

Оценка	Критерии
Отлично	Работа выполнена полностью. Правильно определена функция <code>objective</code> для XGBoost/LightGBM. Проведена оптимизация Optuna (не менее 50 trials). Построены все три обязательные визуализации. Выполнено сравнение с GridSearchCV (или RandomizedSearchCV) с корректными замерами времени. Сделан обоснованный вывод о преимуществах Optuna. Реализована вариативная часть (pruning или многокритериальная оптимизация). Защита: свободное владение материалом.
Хорошо	Работа выполнена полностью, но отсутствует одна из визуализаций, или сравнение выполнено только с baseline (без GridSearch), или количество trials < 30. Вариативная часть не выполнена. Защита: ответы на большинство вопросов.
Удовлетворительно	Optuna запущена, но пространство поиска задано некорректно (только 1–2 параметра), или результат не улучшил baseline. Отсутствует сравнение с другим методом. Отчёт неполный. Студент путает байесовскую оптимизацию со случайным поиском.
Неудовлетворительно	Работа не представлена, или Optuna не использована (например, ручной перебор), или код неработоспособен, или визуализации отсутствуют. Отчёт не соответствует требованиям.

Лабораторная работа №4.

Создание Feature Store из витрины данных

Цель работы: Освоить практические навыки создания Feature Store на базе фреймворка Feast для централизованного управления признаками. Научиться интегрировать данные из инжиниринговой витрины (PostgreSQL) в feature store, регистрировать feature views, выполнять материализацию признаков в онлайн-хранилище (Redis) и использовать их для обучения модели и инференса.

Задачи:

- Развернуть инфраструктуру: PostgreSQL (офлайн-хранилище признаков) и Redis (онлайн-хранилище) с использованием Docker Compose.
- Создать витрину данных в PostgreSQL и наполнить её синтетическими данными.
- Инициализировать репозиторий Feast, настроить подключение к PostgreSQL и Redis через `feature_store.yaml`.
- Определить сущность (Entity), источник данных (PostgreSQLSource) и feature view.
- Зарегистрировать объекты Feast командой `feast apply`.
- Выполнить материализацию признаков из офлайн-хранилища в онлайн-хранилище Redis.
- С помощью Python SDK получить признаки из онлайн-хранилища для заданных идентификаторов сущностей.
- Обучить модель на исторических признаках (offline) и выполнить предсказание с использованием онлайн-признаков.

Теоретическая часть

1. Проблема train/serve skew в машинном обучении

В традиционном ML-пайплайне признаки для обучения модели и для инференса часто вычисляются по-разному: при обучении используются сложные SQL-запросы к DWH, а при инференсе — упрощённые функции на языке приложения. Это приводит к расхождению (skew) между тем, на чём модель обучалась, и тем, что подаётся на вход в production. Feature store решает эту проблему, определяя признаки один раз и используя одни и те же определения как для обучения, так и для инференса.

2. Архитектура Feature Store (Feast)

Feast (Feature Store) — open-source платформа для централизованного управления признаками. Ключевые компоненты:

- Offline Store — хранилище исторических данных, используемых для обучения моделей. Поддерживаются: PostgreSQL, BigQuery, Snowflake, Redshift и другие.
- Online Store — низколатентное хранилище для получения актуальных значений признаков в реальном времени. Поддерживаются: Redis, DynamoDB, Datastore, SQLite.
- Registry — хранилище метаданных (определения сущностей, feature views, источников). Может быть файловым или SQL-ориентированным.
- Materialization — процесс переноса признаков из offline store в online store.

3. Основные объекты Feast

- Entity — бизнес-объект, для которого вычисляются признаки (customer_id, user_id, driver_id и т.д.).
- Source — источник сырых данных (таблица в PostgreSQL, паркетный файл, топик Kafka).
- Feature View — логическая группа признаков, объединённых общим источником и сущностью. Определяет схему признаков, TTL (время жизни), а также содержит ссылку на source.
- Feature Service — группа feature views, объединённая под одним именем для удобного доступа при обучении и инференсе.

4. Процесс материализации

Материализация загружает признак из offline store в online store за указанный временной интервал. В Redis все feature views, относящиеся к одной сущности, хранятся в одном Redis hash, что позволяет выполнять чтение признаков из нескольких feature views за один pipeline.

5. Преимущества Feature Store

1. Устранение train/serve skew — единое определение признаков.
2. Возможность временных срезов (time travel) — обучение модели на исторических данных с точным указанием временного состояния признаков на момент события.
3. Низкая латентность инференса — за счёт in-memory хранилища (Redis).
4. Повторное использование признаков — между командами и проектами.

Пример практического выполнения

Сценарий: Система кредитного скоринга. Витрина данных в PostgreSQL содержит признаки клиентов. Цель: зарегистрировать признаки, материализовать их в Redis и использовать для предсказания вероятности дефолта.

1. Инфраструктура (docker-compose.yaml)

yaml

```
version: '3.8'
```

```
services:
```

```
  postgres-offline:
```

```
    image: postgres:15
```

```
    container_name: postgres-offline
```

```
    environment:
```

```
      POSTGRES_USER: feast
```

POSTGRES_PASSWORD: feast123

POSTGRES_DB: feature_store

ports:

- "5432:5432"

volumes:

- postgres_data:/var/lib/postgresql/data

redis-online:

image: redis:7-alpine

container_name: redis-online

ports:

- "6379:6379"

command: redis-server --appendonly yes

feast-registry:

image: postgres:15

container_name: feast-registry

environment:

POSTGRES_USER: registry

POSTGRES_PASSWORD: registry123

POSTGRES_DB: feast_registry

ports:

- "5433:5432"

volumes:

postgres_data:

Запуск инфраструктуры: docker-compose up -d

2. Создание витрины данных в PostgreSQL

sql

```
-- Подключение к postgres-offline: psql -h localhost -p 5432 -U feast -d  
feature_store
```

```
CREATE SCHEMA IF NOT EXISTS retail;
```

```
CREATE TABLE retail.customer_features (  
    customer_id INTEGER,  
    total_purchase_amount NUMERIC(10,2),  
    purchase_frequency INTEGER,  
    avg_transaction_value NUMERIC(10,2),  
    days_since_last_purchase INTEGER,  
    credit_score INTEGER,  
    event_timestamp TIMESTAMP  
);
```

```
INSERT INTO retail.customer_features VALUES  
(101, 12500.00, 45, 277.78, 12, 720, '2024-01-15 10:00:00'),  
(101, 13200.00, 48, 275.00, 5, 725, '2024-02-15 10:00:00'),  
(102, 8500.00, 28, 303.57, 20, 680, '2024-01-15 10:00:00'),  
(102, 9200.00, 30, 306.67, 8, 690, '2024-02-15 10:00:00'),  
(103, 4500.00, 15, 300.00, 45, 580, '2024-01-15 10:00:00'),  
(103, 4800.00, 16, 300.00, 30, 590, '2024-02-15 10:00:00');
```

3. Инициализация Feast и настройка

```
bash  
pip install feast 'feast[postgres]' redis psycpg2-binary  
feast init credit_feast_repo  
cd credit_feast_repo  
credit_feast_repo/feature_store.yaml:  
yaml  
project: credit_scoring
```

```
registry: postgresql://registry:registry123@localhost:5433/feast_registry
provider: local
online_store:
  type: redis
  connection_string: "localhost:6379"
offline_store:
  type: postgres
  host: localhost
  port: 5432
  database: feature_store
  db_schema: retail
  user: feast
  password: feast123
entity_key_serialization_version: 2
```

4. Определение сущности и feature view

```
credit_feast_repo/features.py:
python
from datetime import timedelta
from feast import Entity, FeatureView, Field, PostgreSQLSource
from feast.types import Int64, Float32
```

1. Определение сущности

```
customer = Entity(
    name="customer_id",
    join_keys=["customer_id"],
    description="Уникальный идентификатор клиента"
)
```

2. Источник данных из PostgreSQL

```
customer_source = PostgreSQLSource(
```

```
name="customer_features_source",
query="SELECT customer_id, total_purchase_amount, purchase_frequency,
avg_transaction_value, days_since_last_purchase, credit_score, event_timestamp
FROM retail.customer_features",
timestamp_field="event_timestamp",
)
```

3. Feature view

```
customer_fv = FeatureView(
    name="customer_credit_features",
    entities=[customer],
    ttl=timedelta(days=30),
    schema=[
        Field(name="total_purchase_amount", dtype=Float32),
        Field(name="purchase_frequency", dtype=Int64),
        Field(name="avg_transaction_value", dtype=Float32),
        Field(name="days_since_last_purchase", dtype=Int64),
        Field(name="credit_score", dtype=Int64),
    ],
    source=customer_source,
    online=True,
)
```

5. Регистрация объектов Feast

```
bash
cd credit_feast_repo
feast apply
```

6. Материализация признаков в Redis

```
bash
# Материализация за последние 30 дней
feast materialize 2024-01-01T00:00:00 2024-03-01T00:00:00
```

7. Получение признаков из онлайн-хранилища

python

```
from feast import FeatureStore
```

```
import pandas as pd
```

```
fs = FeatureStore(repo_path="credit_feast_repo/feature_repo")
```

```
features = [  
    "customer_credit_features:total_purchase_amount",  
    "customer_credit_features:purchase_frequency",  
    "customer_credit_features:credit_score"  
]
```

```
entity_rows = [  
    {"customer_id": 101},  
    {"customer_id": 102}  
]
```

```
online_features = fs.get_online_features(  
    features=features,  
    entity_rows=entity_rows  
)to_dict()
```

```
print(online_features)
```

8. Обучение модели и инференс

python

1. Получение исторических признаков для обучения (offline)

```
train_df = fs.get_historical_features(  
    entity_df=pd.DataFrame({  
        "customer_id": [101, 102, 103],
```

```

        "event_timestamp": pd.to_datetime(["2024-02-15", "2024-02-15", "2024-02-15"])
    }),
    features=features
).to_df()

```

2. Обучение модели (пример)

```

from sklearn.ensemble import RandomForestClassifier
import numpy as np

```

Допустим, целевая переменная известна

```

y_train = np.array([0, 1, 1]) # 0 - не дефолт, 1 - дефолт
X_train = train_df[["total_purchase_amount", "purchase_frequency",
"credit_score"]]

```

```

model = RandomForestClassifier()
model.fit(X_train, y_train)

```

3. Инференс с использованием онлайн-признаков

```

new_entity_rows = [{"customer_id": 104}]
online_features_new = fs.get_online_features(features=features,
entity_rows=new_entity_rows).to_dict()
X_infer = pd.DataFrame(online_features_new)
prediction = model.predict(X_infer)
print(prediction)

```

Результаты выполнения:

- Регистрация feature view выполнена успешно.
- Материализация завершена (синхронизация PostgreSQL → Redis).
- Онлайн-признаки получены с латентностью <10 мс.

- Модель обучена и выполнила инференс с использованием актуальных признаков из Redis.

Индивидуальные задания

Для каждого варианта необходимо:

1. Развернуть PostgreSQL + Redis через Docker Compose.
2. Создать витрину данных в соответствии с вариантом.
3. Настроить Feast (feature_store.yaml, определение entity, source, feature view).
4. Зарегистрировать объекты.
5. Выполнить материализацию.
6. Получить онлайн-признаки.
7. Обучить простую модель (LogisticRegression / RandomForest) и выполнить предсказание.

Вариант	Домен (витрина данных)	Таблица / признаки	Сущность
1	Электронная коммерция	user_features: user_id, total_spent, avg_order_value, order_frequency, days_since_last_order, preferred_category	user_id
2	Телеком (отток)	customer_features: customer_id, monthly_charges, tenure_months, total_calls, avg_call_duration, complaint_count, payment_delay_days	customer_id
3	Кредитный скоринг (расширенный)	borrower_features: borrower_id, annual_income, debt_to_income_ratio, num_open_accounts,	borrower_id

Вариант	Домен (витрина данных)	Таблица / признаки	Сущность
		credit_utilization_rate, late_payments_30d, inquiry_count	
4	Гейминг	player_features: player_id, avg_session_duration, total_playtime_hours, level, daily_active_days, in_app_purchases_total	player_id
5	Логистика (доставка)	driver_features: driver_id, total_deliveries, avg_delivery_time_min, rating_avg, distance_traveled_km, active_hours_per_week	driver_id
6	Стreaming-сервис	user_features: user_id, watch_time_hours, content_genre_preference, avg_rating_given, num_devices, subscription_type_encoded	user_id
7	Финансовый скоринг (транзакции)	account_features: account_id, avg_balance_last_30d, transaction_volume, cash_withdrawal_frequency, overdraft_count, risk_score	account_id
8	Социальная сеть	user_features: user_id, num_friends, posts_per_week, likes_received_avg, engagement_rate, account_age_days	user_id

Вариант	Домен (витрина данных)	Таблица / признаки	Сущность
9	Мобильное приложение (фитнес)	athlete_features: athlete_id, avg_daily_steps, weekly_workouts, calories_burned_total, sleep_hours_avg, active_days_per_week	athlete_id
10	Е-health (медицина)	patient_features: patient_id, age_group_encoded, bmi, blood_pressure_systolic, cholesterol_level, num_visits_last_year	patient_id
11	Прогнозирование спроса	product_features: product_id, avg_daily_sales, stockout_frequency, price_elasticity, competitor_price_diff, promotion_impact	product_id
12	Кибербезопасность	session_features: session_id, login_attempts, ip_reputation_score, device_fingerprint_risk, geolocation_risk, time_of_day_encoded	session_id
13	Автомобильная телематика	vehicle_features: vehicle_id, total_mileage, avg_speed, harsh_braking_count, acceleration_events, fuel_efficiency	vehicle_id
14	HR (аналитика персонала)	employee_features: employee_id, performance_score, days_since_last_promotion,	employee_id

Вариант	Домен (витрина данных)	Таблица / признаки	Сущность
		training_hours, overtime_hours, satisfaction_survey_score	
15	IoT (умный дом)	device_features: device_id, avg_temperature_setting, energy_consumption_kwh, connectivity_stability_score, firmware_version_encoded	device_id

Дополнительные указания:

- Каждый студент самостоятельно генерирует не менее 10 строк данных для своего варианта.
- Всегда включать колонку `event_timestamp` (TIMESTAMP).

Для сущности, не являющейся целочисленной (например, `session_id` текстовый), использовать `ValueType.STRING`.

Контрольные вопросы

1. В чём заключается основное преимущество feature store перед ad-hoc вычислением признаков в коде?
2. Как Feast решает проблему train/serve skew?
3. В чём разница между offline store и online store в Feast? Приведите примеры.
4. Для чего используется механизм materialization в Feast?
5. Почему Redis обеспечивает низкую латентность при получении признаков по сравнению с PostgreSQL?
6. Как в Redis организовано хранение признаков для одной сущности?
7. Что такое TTL в контексте online store и зачем он нужен?

8. В каком случае файловый registry не подходит и нужен SQL registry?
9. Как с помощью Python SDK получить признаки из online store?
10. По результатам вашего варианта: как изменилась бы латентность инференса, если бы вы получали признаки напрямую из PostgreSQL, а не из Redis?

Требования к отчёту

Отчёт оформляется по ГОСТ 7.32-2017. Содержит:

1. Титульный лист.
2. Цель и задачи работы.
3. Теоретическое обоснование (понятие feature store, архитектура Feast).
4. Практическая часть:
 - Файл `docker-compose.yaml` и инструкция по запуску.
 - SQL-скрипты создания витрины и наполнения данными.
 - `feature_store.yaml` для вашего варианта.
 - Код определения Entity, PostgreSQLSource, FeatureView.
 - Скриншоты выполнения `feast apply`, `feast materialize`.
 - Код получения онлайн-признаков и обучения модели.
5. Результаты выполнения (таблица полученных признаков, время материализации).
6. Анализ результатов (сравнение offline vs online доступа к признакам).
7. Ответы на контрольные вопросы.
8. Вывод.
9. Список литературы.

Вариативная часть для оценки «отлично»:

- Настроить SQL registry вместо файлового.

- Добавить второй feature view и объединить их в FeatureService.
- Реализовать автоматическую материализацию по расписанию с использованием Apache Airflow.

Критерии оценивания

Оценка	Критерии
Отлично	Работа выполнена полностью. Инфраструктура развёрнута через Docker Compose. Настроен PostgreSQL offline store и Redis online store. Все объекты Feast (Entity, Source, Feature View) определены корректно. Материализация выполнена успешно. Получение онлайн-признаков работает. Обучение модели и инференс выполнены. Реализована вариативная часть. Отчёт содержит полные листинги и анализ. Защита: глубокое понимание архитектуры feature store.
Хорошо	Работа выполнена полностью, но отсутствует вариативная часть, или использован файловый офлайн-стор (Parquet) вместо PostgreSQL, или модель обучена, но инференс не показан. Защита: ответы на большинство вопросов.
Удовлетворительно	Feast инициализирован, но настройка онлайн-стора не завершена, или feast materialize не выполнен, или признаки получены только из офлайн-стора. Отчёт неполный. Студент не объясняет разницу между offline и online store.
Неудовлетворительно	Работа не представлена, или Feast не установлен/не настроен, или отсутствует подключение к Redis, или материализация не выполнена. Отчёт не соответствует требованиям.

Лабораторная работа №5.

Асинхронный инференс через FastAPI + Redis

Цель работы: Разработать асинхронный микросервис для инференса моделей машинного обучения с использованием очереди задач на базе Redis. Научиться разделять приём запросов (FastAPI) и тяжёлые вычисления (worker), а также реализовывать механизм получения результатов по асинхронному идентификатору (task_id).

Задачи:

- Развернуть Redis (контейнер Docker) для использования в качестве брокера очередей и хранилища результатов.
- Реализовать REST API на FastAPI с двумя эндпоинтами:
 - POST /predict – принимает входные данные, ставит задачу в очередь, возвращает task_id.
 - GET /result/{task_id} – возвращает статус задачи и результат предсказания (если готов).
- Реализовать worker-процесс, который:
 - забирает задачи из очереди Redis;
 - загружает предобученную модель (например, пайплайн из лабораторной №1);
 - выполняет предсказание;
 - сохраняет результат в Redis по ключу result:{task_id}.
- Обеспечить обработку ошибок (некорректные входные данные, отсутствие результата).
- Измерить пропускную способность системы (RPS) и латентность получения результата при асинхронном режиме.
- Продемонстрировать работу микросервиса с помощью клиента (например, requests или curl).

Теоретическая часть

1. Синхронный vs асинхронный инференс

Характеристика	Синхронный (HTTP 200 000)	Асинхронный (очередь)
Время ответа	~ время предсказания (сотни мс – секунды)	мгновенный ответ с task_id
Обработка пиковых нагрузок	ограничена количеством воркеров и потоков	очередь буферизует запросы
Сквозная латентность	минимальная	выше (время ожидания в очереди + инференс)
Применение	real-time с быстрыми моделями	пакетная обработка, тяжёлые модели, долгий инференс

При асинхронном подходе FastAPI не блокируется на время работы модели. Запрос немедленно принимается, задача ставится в очередь, а клиент может опрашивать результат.

2. Redis как очередь и хранилище результатов

Redis поддерживает структуры данных **List** (**LPUSH** / **RPOP**), которые позволяют реализовать очередь FIFO. Типовой паттерн:

- Producer (FastAPI) добавляет задачу в конец списка: **R PUSH task_queue "{task_id}:{input_json}"**.
- Worker в цикле извлекает задачу из головы: **L POP task_queue** (или использует блокирующую **BLPOP** для ожидания).
- После выполнения результат сохраняется: **SET result:{task_id} "{prediction_json}" EX 3600** (с таймаутом).

Альтернатива: использование библиотеки `rq` (Redis Queue), которая предоставляет более высокоуровневый API, но в данной работе рекомендуется ручная реализация для понимания принципов.

3. Генерация `task_id`

`task_id` должен быть уникальным и не предсказуемым. Обычно используется `uuid.uuid4().hex`. Клиент получает его сразу в ответе на `POST /predict`.

4. Статусы задачи

Рекомендуется ввести три статуса:

- `"pending"` – задача в очереди, ещё не взята воркером.
- `"processing"` – задача выполняется.
- `"completed"` – результат готов.
- `"failed"` – произошла ошибка.

Статус может храниться в отдельном ключе или быть частью результата.

5. Масштабирование воркеров

Архитектура позволяет запустить несколько экземпляров воркера (в разных процессах или контейнерах). Redis гарантирует, что каждая задача будет отдана только одному воркеру (при использовании атомарных операций).

Пример практического выполнения

Сценарий: Микросервис для предсказания вероятности оттока клиента (telco churn). Модель – пайплайн из лабораторной №1 (RandomForest). Redis запущен локально.

1. Запуск Redis

```
bash
```

```
docker run -d --name redis-inference -p 6379:6379 redis:7-alpine
```

2. Файл `model_loader.py` (загрузка модели)

```
python
```

```
import joblib
```

```
def load_model():  
    return joblib.load('churn_pipeline.joblib')  
  
# Глобальный объект для worker'a (инициализируется один раз)  
_pipeline = None
```

```
def get_pipeline():  
    global _pipeline  
    if _pipeline is None:  
        _pipeline = load_model()  
    return _pipeline
```

3. FastAPI приложение (main.py)

python

```
import asyncio  
import json  
import uuid  
import redis.asyncio as redis_async  
from fastapi import FastAPI, HTTPException, BackgroundTasks  
from pydantic import BaseModel  
from typing import Dict, Any, Optional
```

```
app = FastAPI(title="Async Inference Service")
```

```
# Подключение к Redis (асинхронный клиент)
```

```
redis_client = redis_async.from_url("redis://localhost:6379", decode_responses=True)
```

```
# Названия ключей
```

```
QUEUE_KEY = "inference_queue"
```

```
RESULT_PREFIX = "result:"
```

```

class PredictRequest(BaseModel):
    features: Dict[str, Any] # ключи: age, tenure, monthly_charges, ...

class PredictResponse(BaseModel):
    task_id: str
    status: str = "pending"

class ResultResponse(BaseModel):
    status: str
    prediction: Optional[float] = None
    error: Optional[str] = None

@app.post("/predict", response_model=PredictResponse)
async def predict(request: PredictRequest):
    # Генерация уникального ID
    task_id = uuid.uuid4().hex

    # Сохраняем статус "pending"
    await redis_client.set(f'{RESULT_PREFIX}{task_id}:status', "pending")

    # Помещаем задачу в очередь: формат "task_id|features_json"
    task_data = f'{task_id}|{json.dumps(request.features)}'
    await redis_client.rpush(QUEUE_KEY, task_data)

    return PredictResponse(task_id=task_id, status="pending")

@app.get("/result/{task_id}", response_model=ResultResponse)
async def get_result(task_id: str):
    # Проверяем статус

```

```

status = await redis_client.get(f'{RESULT_PREFIX}{task_id}:status')
if status is None:
    raise HTTPException(status_code=404, detail="Task not found")

if status == "completed":
    prediction = await redis_client.get(f'{RESULT_PREFIX}{task_id}:prediction')
    return ResultResponse(status="completed", prediction=float(prediction))
)

elif status == "failed":
    error = await redis_client.get(f'{RESULT_PREFIX}{task_id}:error')
    return ResultResponse(status="failed", error=error)
else:
    return ResultResponse(status=status, prediction=None)

```

Эндпоинт для health check

```

@app.get("/health")
async def health():
    return {"status": "ok"}

```

4. Worker (worker.py)

python

```

import redis
import json
import time
from model_loader import get_pipeline

```

Синхронное подключение Redis (для worker'a)

```

redis_client = redis.Redis(host='localhost', port=6379, decode_responses=True)

```

```
QUEUE_KEY = "inference_queue"
```

```
RESULT_PREFIX = "result:"
```

```
def process_task(task_data: str):
```

```
    try:
```

```
        # Разбор задачи
```

```
        task_id, features_json = task_data.split('|', 1)
```

```
        features = json.loads(features_json)
```

```
        # Статус "processing"
```

```
        redis_client.set(f'{RESULT_PREFIX}{task_id}:status', "processing")
```

```
        # Загрузка модели и предсказание
```

```
        pipeline = get_pipeline()
```

```
        # Преобразование словаря в DataFrame (упрощённо)
```

```
        import pandas as pd
```

```
        df = pd.DataFrame([features])
```

```
        prediction = pipeline.predict_proba(df)[0, 1] # вероятность положительного класса
```

```
        # Сохраняем результат
```

```
        redis_client.set(f'{RESULT_PREFIX}{task_id}:prediction', prediction
```

```
)
```

```
        redis_client.set(f'{RESULT_PREFIX}{task_id}:status', "completed")
```

```
        # Устанавливаем TTL 1 час
```

```
        redis_client.expire(f'{RESULT_PREFIX}{task_id}:prediction', 3600)
```

```
        redis_client.expire(f'{RESULT_PREFIX}{task_id}:status', 3600)
```

```
        print(f"Task {task_id} completed: {prediction}")
```

```
    except Exception as e:
```

```
redis_client.set(f"{RESULT_PREFIX} {task_id}:status", "failed")
redis_client.set(f"{RESULT_PREFIX} {task_id}:error", str(e))
print(f"Task {task_id} failed: {e}")
```

```
def main():
    print("Worker started. Waiting for tasks...")
    while True:
        # Блокирующая операция: ожидание задачи до 5 секунд
        task_data = redis_client.blpop(QUEUE_KEY, timeout=5)
        if task_data is None:
            # Если нет задач, немного спим (или продолжаем)
            continue
        # task_data[0] - ключ очереди, task_data[1] - значение
        process_task(task_data[1])

if __name__ == "__main__":
    main()
```

5. Запуск

bash

Терминал 1: запуск worker (синхронный процесс)

python worker.py

Терминал 2: запуск FastAPI

uvicorn main:app --reload --port 8000

6. Тестирование

python

```
import requests
```

```
import time
```

1. Отправка запроса

```

resp = requests.post("http://localhost:8000/predict", json={
    "features": {"age": 45, "tenure": 24, "monthly_charges": 89.5}
})
task_id = resp.json()["task_id"]
print(f"Task ID: {task_id}")

# 2. Опрос результата
while True:
    result = requests.get(f"http://localhost:8000/result/{task_id}").json()
    if result["status"] == "completed":
        print(f"Prediction: {result['prediction']}")
        break
    elif result["status"] == "failed":
        print(f"Error: {result['error']}")
        break
    else:
        print(f"Status: {result['status']}, waiting...")
        time.sleep(1)

```

Результат выполнения:

- POST /predict возвращает task_id мгновенно (<10 мс).
- Worker выполняет инференс (занято ~50 мс).
- Клиент получает результат после опроса.

Индивидуальные задания

Каждый вариант требует:

1. Использовать свою модель (пайплайн, обученную в лабораторной №1 или №2).
2. Реализовать асинхронный сервис аналогично примеру, но с учётом специфики задачи.

3. Настроить запуск через Docker Compose (FastAPI + Redis + worker).
4. Написать клиент для тестирования (многопоточный/асинхронный).
5. Измерить RPS и среднее время получения результата.

Вариант	Домен	Входные признаки (через POST)	Выход	Особенность
1	Кредитный скоринг	{age, income, emp_length, debt_ratio, credit_lines}	risk_score (0..1)	Модель XGBoost
2	Отток телеком	{tenure, monthly_charges, total_charges, contract_type, payment_method }	churn_prob	Категориальное кодирование в воркере
3	Прогноз цены квартиры	{sqft, bedrooms, bathrooms, location_score, year_built}	price (регрессия)	Нормализация в модели
4	Диагностика заболевания	{age, bmi, blood_pressure, glucose, insulin}	diabetes_prob	Модель LightGBM
5	Мошенничество в платежах	{amount, merchant_category, device_risk, hour_of_day}	fraud_prob	Имитация долгого инференса (time.sleep)

Вариант	Домен	Входные признаки (через POST)	Выход	Особенность
6	Классификация изображений	{image_base64} (строка)	class_label	Преобразование base64 в numpy в воркере
7	Рекомендательная система	{user_id, item_id, context}	rating (1-5)	Использование предвычисленных эмбеддингов
8	Обнаружение аномалий в логах	{log_length, error_count, time_since_last}	anomaly_score	Модель IsolationForest
9	Прогноз спроса на товар	{item_id, day_of_week, promotion_flag, price}	demand	Модель RandomForestRegressor
10	NLP: тональность текста	{text} (строка)	sentiment (positive/negative)	Использование TF-IDF + модель в пайплайне
11	Сегментация клиентов (вероятность принадлежности к сегменту)	{age, income, purchases_freq, categories}	segment_prob (вектор из 3 классов)	Модель softmax
12	Прогноз времени доставки	{distance, traffic_score, weather_score,	delivery_time_min	Модель LightGBM

Вариант	Домен	Входные признаки (через POST)	Выход	Особенность
		<code>driver_experience</code> <code>}</code>		
13	Распознавание атак в сети	<code>{packet_size,</code> <code>protocol, port,</code> <code>flags}</code>	<code>attack_prob</code>	Модель RandomForest
14	Оценка удовлетворённости сотрудника	<code>{satisfaction,</code> <code>years_at_company,</code> <code>overtime,</code> <code>performance_rating}</code>	<code>turnover_prob</code>	Модель XGBoost с дисбалансом
15	Прогноз возврата кредита (мультикласс)	<code>{loan_amount,</code> <code>interest_rate,</code> <code>grade,</code> <code>annual_income}</code>	<code>loan_status</code> (0/1/2)	Обработка категорий

Контрольные вопросы

1. В чем основное отличие синхронного инференса от асинхронного с очередью?
2. Почему для реализации очереди используется Redis List, а не Pub/Sub?
3. Как гарантировать, что задача не будет потеряна при падении воркера?
4. Какую структуру данных Redis использует для блокирующего извлечения задачи (BLPOP)?

5. Как можно масштабировать воркеры горизонтально (несколько экземпляров)?
6. Какие проблемы возникают при использовании `redis.asyncio` в FastAPI для `BLPOP`?
7. Как долго хранить результаты в Redis и почему нужно устанавливать TTL?
8. Как можно реализовать WebSocket-уведомление клиента о готовности результата вместо опроса?
9. Как повлияет на пропускную способность добавление второго воркера?
10. Как изменится архитектура, если модель требует GPU и инференс занимает 2 секунды?

Требования к отчёту

Отчёт оформляется по ГОСТ 7.32-2017. Содержит:

1. Титульный лист.
2. Цель и задачи.
3. Теоретическое обоснование (синхронный/асинхронный инференс, очередь Redis).
4. Практическая часть:
 - `docker-compose.yml` для FastAPI, Redis, worker (или отдельные инструкции).
 - Листинг `main.py` и `worker.py` с комментариями.
 - Пример входного запроса и ответов.
 - Код клиента (тестирование).
 - Результаты измерения времени обработки.
5. Результаты выполнения (скриншоты логов, ответы API).
6. Анализ результатов (сравнение синхронного режима с асинхронным, RPS).
7. Ответы на контрольные вопросы.

8. Вывод.
9. Список литературы.

Вариативная часть для оценки «отлично»:

- Реализовать WebSocket-эндпоинт `/ws/{task_id}` для автоматической отдачи результата.
- Использовать `rq` библиотеку вместо ручной работы с Redis List.
- Добавить эндпоинт `GET /queue/status` с длиной очереди.

Критерии оценивания

Оценка	Критерии
Отлично	Работа выполнена полностью. FastAPI, worker и Redis взаимодействуют корректно. Задачи ставятся в очередь, воркер выполняет, результат сохраняется. Клиент получает <code>task_id</code> и успешно забирает результат. Измерены RPS и латентность. Реализована вариативная часть. Отчёт содержит анализ пропускной способности. Защита: понимание всех компонентов.
Хорошо	Работа выполнена полностью, но отсутствует измерение производительности, или не реализована обработка ошибок (<code>failed</code> статус). Вариативная часть не выполнена. Защита: ответы на большинство вопросов.
Удовлетворительно	Основной функционал работает, но очередь реализована через простой список (без блокирующих операций), или worker не отдельный процесс, а <code>BackgroundTasks</code> в FastAPI (синхронный режим). Отчёт неполный. Студент не объясняет преимущества асинхронного подхода.

Оценка	Критерии
Неудовлетворительно	Работа не представлена, или Redis не используется, или endpoint /predict блокируется на время инференса (синхронный код). Отсутствует сохранение результатов. Отчёт не соответствует требованиям.

Лабораторная работа №6.

Пакетная обработка через Kafka и Spark Structured Streaming

Цель работы: Построить стриминговый пайплайн предсказаний на основе Apache Kafka и Spark Structured Streaming. Научиться генерировать поток синтетических данных, отправлять их в Kafka, обрабатывать с помощью Spark, применять модель машинного обучения (UDF) и записывать результаты в аналитическую базу данных ClickHouse.

Задачи:

- Развернуть инфраструктуру с использованием Docker Compose: Zookeeper, Kafka (брокер), Spark (master + worker), ClickHouse.
- Реализовать симулятор данных (producer) на Python, который отправляет JSON-сообщения в топик Kafka с заданной периодичностью.
- Разработать Spark Structured Streaming приложение (PySpark), которое:
 - читает поток данных из топика Kafka;
 - десериализует JSON, извлекает признаки;
 - применяет предобученную модель (или пайплайн) через User Defined Function (UDF);
 - добавляет колонку с предсказанием;
 - записывает результат (исходные признаки + предсказание) в таблицу ClickHouse.
- Обеспечить обработку ошибок (пропуск некорректных записей).
- Визуализировать загрузженность потока (процесс чтения/записи) и проверить корректность записи в ClickHouse.
- Сравнить производительность (задержка, throughput) в зависимости от параметров окна и размера микробатча.

Теоретическая часть

1. Стриминговая обработка в ML-пайплайнах

Во многих реальных сценариях (логи мобильных приложений, IoT-данные,

события с веб-сайтов) данные поступают непрерывно. Для принятия оперативных решений необходимо обрабатывать их по мере поступления, не дожидаясь накопления всего объёма. Стриминговый инференс позволяет применять модель к каждому сообщению практически в реальном времени.

2. **Apache** **Kafka**

Распределённая платформа для потоковой передачи сообщений с низкой задержкой. Основные понятия:

- **Topic** – категория/поток сообщений.
- **Producer** – приложение, отправляющее сообщения в топик.
- **Consumer** – приложение, читающее сообщения.
- **Partition** – единица параллелизма.

В данной работе Kafka используется как буфер между генератором данных и Spark.

3. **Spark** **Structured** **Streaming**

Модуль Apache Spark для обработки потоков данных, построенный на той же модели, что и пакетная обработка (DataFrames/Datasets). Ключевые особенности:

- **Микробатчинг** (по умолчанию) – поток разбивается на маленькие пакеты по интервалу (например, 2 секунды).
- **Continuous processing** (экспериментальный) – для сверхнизких задержек.
- **Fault tolerance** – возможность возобновления с места сбоя (checkpointing).
- **Source/Sink** – чтение из Kafka, файлов, сокетов; запись в Kafka, файлы, базы данных.

4. **User** **Defined** **Function** **(UDF)**

Поскольку стандартные функции Spark не поддерживают вызов сложных ML-моделей (например, пайплайн scikit-learn, XGBoost), необходимо обернуть загрузку модели и предсказание в пользовательскую функцию, которая применяется к каждой строке DataFrame. UDF работает на каждом

исполнителя (executor) и может кэшировать модель (broadcast) для повышения эффективности.

5. ClickHouse как аналитический sink

ClickHouse – колоночная OLAP-база данных, оптимизированная для высокоскоростной вставки и аналитических запросов. Подходит для накопления большого количества предсказаний с последующим анализом (агрегации, дашборды). Запись производится через JDBC (или родной коннектор) в режиме `append`.

6. Типовой пайплайн

text

Producer (Python) → Kafka (topic) → Spark (readStream) → десериализация → UDF (модель) → writeStream → ClickHouse.

Методика выполнения работы

Сценарий: Прогнозирование оттока клиентов в реальном времени. Данные (логи активности) генерируются с частотой 10 сообщений/сек. Модель – RandomForestClassifier (пайплайн из лабораторной №1). Результат сохраняется в ClickHouse.

1. Инфраструктура (docker-compose.yml)

yaml

version: '3.8'

services:

zookeeper:

image: confluentinc/cp-zookeeper:7.4.0

container_name: zookeeper

environment:

ZOOKEEPER_CLIENT_PORT: 2181

ports:

- "2181:2181"

kafka:

image: confluentinc/cp-kafka:7.4.0

container_name: kafka

depends_on:

- zookeeper

environment:

KAFKA_BROKER_ID: 1

KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181

KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092

KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1

ports:

- "9092:9092"

clickhouse:

image: clickhouse/clickhouse-server:23.8

container_name: clickhouse

ports:

- "8123:8123" # HTTP

- "9000:9000" # Native

volumes:

- clickhouse-data:/var/lib/clickhouse

spark-master:

image: bitnami/spark:3.4.1

container_name: spark-master

environment:

- SPARK_MODE=master

ports:

- "7077:7077"

- "8080:8080"

volumes:

- ./spark-apps:/opt/spark-apps

spark-worker:

image: bitnami/spark:3.4.1

container_name: spark-worker

depends_on:

- spark-master

environment:

- SPARK_MODE=worker

- SPARK_MASTER_URL=spark://spark-master:7077

volumes:

- ./spark-apps:/opt/spark-apps

volumes:

clickhouse-data:

Запуск: `docker-compose up -d`

2. Создание топика Kafka

bash

`docker exec -it kafka kafka-topics --create --topic user_events --bootstrap-server localhost:9092 --partitions 3 --replication-factor 1`

3. Producer (симулятор данных) – `producer.py`

python

`import json`

`import time`

`import random`

`from kafka import KafkaProducer`

`# Настройки`

```
TOPIC = "user_events"
```

```
KAFKA_SERVER = "localhost:9092"
```

```
producer = KafkaProducer(  
    bootstrap_servers=KAFKA_SERVER,  
    value_serializer=lambda v: json.dumps(v).encode('utf-8')  
)
```

```
# Функция генерации случайных признаков клиента
```

```
def generate_event():
```

```
    return {
```

```
        "user_id": random.randint(1, 1000),
```

```
        "age": random.randint(18, 70),
```

```
        "tenure": random.randint(1, 72),
```

```
        "monthly_charges": round(random.uniform(20, 150), 2),
```

```
        "total_charges": round(random.uniform(100, 5000), 2),
```

```
        "contract_type": random.choice(["month-to-month", "one_year", "two_  
year"]),
```

```
        "payment_method": random.choice(["electronic_check", "mailed_check  
", "bank_transfer"]),
```

```
        "timestamp": time.time()
```

```
    }
```

```
# Отправка сообщений
```

```
try:
```

```
    while True:
```

```
        event = generate_event()
```

```
        producer.send(TOPIC, value=event)
```

```
        print(f"Sent: {event}")
```

```
        time.sleep(0.1) # 10 сообщений в секунду
```

```
except KeyboardInterrupt:
```

```
    producer.close()
```

4. Spark Streaming приложение – `streaming_predict.py`

```
python
```

```
from pyspark.sql import SparkSession
```

```
from pyspark.sql.functions import from_json, col, udf, struct
```

```
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, FloatType, DoubleType
```

```
from pyspark.sql.types import TimestampType
```

```
import joblib
```

```
import pandas as pd
```

```
import os
```

```
# -----
```

```
# Загрузка модели (пайплайн) и broadcast
```

```
# -----
```

```
def load_model():
```

```
    # Патч для загрузки модели с диска (на каждом executor'е)
```

```
    # В production лучше передавать через broadcast переменную
```

```
    return joblib.load("/opt/spark-apps/churn_pipeline.joblib")
```

```
_model_broadcast = None
```

```
def predict_udf(age, tenure, monthly_charges, total_charges, contract_type, payment_method):
```

```
    # Преобразование входных значений в DataFrame
```

```
    input_dict = {
```

```
        "age": age,
```

```
        "tenure": tenure,
```

```
        "monthly_charges": monthly_charges,
```

```

        "total_charges": total_charges,
        "contract_type": contract_type,
        "payment_method": payment_method
    }
    df_input = pd.DataFrame([input_dict])
    # Предсказание вероятности положительного класса (churn=1)
    if _model_broadcast is not None:
        model = _model_broadcast.value
    else:
        model = load_model()
    proba = model.predict_proba(df_input)[0, 1]
    return float(proba)

# Создание UDF
predict_udf_spark = udf(predict_udf, DoubleType())

# -----
# Инициализация Spark
# -----

spark = SparkSession.builder \
    .appName("KafkaStreamingInference") \
    .config("spark.sql.streaming.checkpointLocation", "/tmp/spark-checkpoint
s") \
    .getOrCreate()

# Broadcast модели (для эффективности)
model = load_model()
broadcast_model = spark.sparkContext.broadcast(model)
# Замыкание для использования broadcast в UDF

```

```

def predict_with_broadcast(age, tenure, monthly_charges, total_charges, contract_type, payment_method):
    df = pd.DataFrame([[age, tenure, monthly_charges, total_charges, contract_type, payment_method]],
                       columns=["age", "tenure", "monthly_charges", "total_charges", "contract_type", "payment_method"])
    prob = broadcast_model.value.predict_proba(df)[0, 1]
    return float(prob)

```

```

predict_bc_udf = udf(predict_with_broadcast, DoubleType())

```

```

# -----
# Чтение из Kafka
# -----

```

```

schema = StructType([
    StructField("user_id", IntegerType()),
    StructField("age", IntegerType()),
    StructField("tenure", IntegerType()),
    StructField("monthly_charges", FloatType()),
    StructField("total_charges", FloatType()),
    StructField("contract_type", StringType()),
    StructField("payment_method", StringType()),
    StructField("timestamp", DoubleType())
])

```

```

# JSON из value

```

```

kafka_df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "user_events") \

```

```
.option("startingOffsets", "latest") \
.load()
```

```
# Десериализация JSON
```

```
parsed_df = kafka_df \
    .select(from_json(col("value").cast("string"), schema).alias("data")) \
    .select("data.*")
```

```
# Добавление предсказания
```

```
pred_df = parsed_df.withColumn(
    "churn_probability",
    predict_bc_udf(
        col("age"),
        col("tenure"),
        col("monthly_charges"),
        col("total_charges"),
        col("contract_type"),
        col("payment_method")
    )
)
```

```
# -----
```

```
# Запись в ClickHouse
```

```
# -----
```

```
# Необходимо добавить коннектор ClickHouse (например, через JDBC)
```

```
# В реальности требуется добавить библиотеку в classpath
```

```
# Способ 1: foreachBatch с JDBC
```

```
def write_to_clickhouse(df, epoch_id):
    df.write \
```

```
.mode("append") \
.format("jdbc") \
.option("url", "jdbc:clickhouse://localhost:8123/default") \
.option("driver", "com.clickhouse.jdbc.ClickHouseDriver") \
.option("dbtable", "churn_predictions") \
.option("user", "default") \
.option("password", "") \
.save()
```

Способ 2 (упрощённый для демонстрации): вывод в консоль

```
query = pred_df.writeStream \
    .outputMode("append") \
    .foreachBatch(write_to_clickhouse) \
    .trigger(processingTime='2 seconds') \
    .start()
```

```
query.awaitTermination()
```

5. Создание таблицы в ClickHouse

sql

```
CREATE TABLE default.churn_predictions (
    user_id Int32,
    age Int32,
    tenure Int32,
    monthly_charges Float32,
    total_charges Float32,
    contract_type String,
    payment_method String,
    event_timestamp Float64,
    churn_probability Float64,
    processed_at DateTime DEFAULT now()
```

```
) ENGINE = MergeTree()
```

```
ORDER BY (user_id, processed_at);
```

6. Запуск

```
bash
```

```
# Терминал 1: producer
```

```
python producer.py
```

```
# Терминал 2: Spark приложение (передаём jar для JDBC)
```

```
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.4.1,com.clickhouse:clickhouse-jdbc:0.4.6 streaming_predict.py
```

Результат: Данные из Kafka поступают в Spark, модель делает предсказания (вероятность оттока), и строки вставляются в ClickHouse.

Индивидуальные задания

Каждый вариант: реализовать полный конвейер (producer → Kafka → Spark Structured Streaming → ClickHouse). Модель использовать из предыдущих работ (или обучить простую). Варианты отличаются типом данных, схемой сообщений и задачей предсказания.

Вариант	Тип данных	Поля в Kafka (JSON)	Целевая переменная / модель	ClickHouse таблица
1	Логи веб-сервера	{ip, status_code, bytes_sent, user_agent, timestamp}	Класс аномалии (0/1) – модель на основе частоты ошибок	http_logs
2	IoT: датчики температуры	{device_id, temperature, humidity,	Прогноз выхода из строя (0/1)	sensor_predictions

Вариант	Тип данных	Поля в Kafka (JSON)	Целевая переменная / модель	ClickHouse таблица
		pressure, timestamp}		
3	Транзакции банка	{tx_id, amount, merchant, card_type, location_risk_score, timestamp}	Вероятность мошенничества	fraud_scores
4	Клики в мобильном приложении	{user_id, screen_name, time_spent_sec, is_conversion, session_id}	Прогноз конверсии	click_predictions
5	Показания счётчиков	{meter_id, consumption, temperature_outside, weekday, hour}	Аномалия потребления	meter_anomalies
6	Системные логи	{cpu_usage, mem_usage, disk_io, error_count, timestamp}	Критическая ошибка (1) / нет (0)	system_health
7	Текстовые сообщения (user feedback)	{user_id, text, sentiment_placeholder, timestamp}	Тональность (полож./отриц.)	feedback_sentiment

Вариант	Тип данных	Поля в Kafka (JSON)	Целевая переменная / модель	ClickHouse таблица
8	Автомобильная телематика	{car_id, speed, rpm, engine_temp, gps_lat, gps_lon}	Стиль вождения (агрессивный/нормальный)	driving_style
9	Энергопотребление здания	{building_id, energy_kwh, hvac_status, occupancy, timestamp}	Прогноз пикового потребления (регрессия)	energy_forecast
10	Данные с биржи (тикеры)	{symbol, price, volume, bid, ask, timestamp}	Класс движения цены (вверх/вниз)	price_predictions
11	Медицинские показатели	{patient_id, heart_rate, spo2, steps, sleep_hours, timestamp}	Риск гипертонии (0/1)	health_risk
12	Отзывы на товары (партишн)	{product_id, rating, review_length, helpful_votes, timestamp}	Рекомендовать (0/1)	product_feedback
13	Заказы в доставке	{order_id, distance_km, traffic_score, timestamp}	Ожидаемое время доставки (регрессия)	delivery_eta

Вариант	Тип данных	Поля в Kafka (JSON)	Целевая переменная / модель	ClickHouse таблица
		<code>weather_score,</code> <code>driver_rating</code>		
14	Трафик в сети (NetFlow)	<code>{src_ip, dst_ip,</code> <code>bytes, packets,</code> <code>protocol,</code> <code>timestamp}</code>	Атака (DoS) / норма	<code>netflow_anomaly</code>
15	Данные с производственной линии	<code>{machine_id,</code> <code>vibration,</code> <code>temperature,</code> <code>power_consumption,</code> <code>product_quality}</code>	Дефект продукта (0/1)	<code>production_defects</code>

Дополнительные требования:

- Для вариантов с текстом (7) использовать простые признаки (длина, количество символов).
- Все модели должны быть предварительно обучены и сериализованы (joblib/ pickle).

Поток данных – не менее 1000 сообщений для тестирования.

Контрольные вопросы

1. Почему для стримингового инференса используется микробатчинг (Spark Structured Streaming), а не обработка каждого сообщения по отдельности?
2. Как Kafka гарантирует доставку сообщений (at-least-once / exactly-once)?

3. Что произойдёт с приложением Spark, если ClickHouse станет недоступен?
4. Как в Spark Structured Streaming обеспечить ровно-однократную (exactly-once) запись в ClickHouse?
5. В чем преимущество передачи модели через broadcast-переменную по сравнению с загрузкой с диска в каждой UDF?
6. Какие параметры `spark.sql.streaming` влияют на пропускную способность и задержку?
7. Как можно масштабировать Kafka-топик (увеличить число партиций) для увеличения параллелизма в Spark?
8. Как бы вы организовали дедупликацию сообщений при возможной повторной доставке из Kafka?
9. Какой режим вывода (outputMode) используется в данном пайплайне и почему?
10. Как изменится архитектура, если модель должна работать с батчем из 100 сообщений для получения одного предсказания?

Требования к отчёту

Отчёт оформляется по ГОСТ 7.32-2017. Содержит:

1. Титульный лист.
2. Цель и задачи.
3. Теоретическое обоснование (Kafka, Spark Structured Streaming, ClickHouse, UDF).
4. Практическая часть:
 - Файл `docker-compose.yml` и команды запуска.
 - Код producer (генератор данных).
 - Код Spark Streaming приложения (с комментариями).
 - Скрипт создания таблицы в ClickHouse.
 - Скриншоты логов (Kafka, Spark, ClickHouse).

5. Результаты выполнения (количество обработанных записей, метрики задержки, throughput).
6. Анализ результатов (влияние размера микробатча на задержку).
7. Ответы на контрольные вопросы.
8. Вывод.
9. Список литературы.

Вариативная часть для оценки «отлично»:

- Настроить checkpointing для восстановления после сбоя.
- Добавить водяные знаки (watermark) и оконную агрегацию (например, скользящее среднее предсказаний).
- Визуализировать поток в реальном времени с помощью Grafana + ClickHouse.

Критерии оценивания

Оценка	Критерии
Отлично	Работа выполнена полностью. Развёрнута полная инфраструктура (Kafka, Spark, ClickHouse). Producer генерирует данные, Spark читает из Kafka, применяет UDF с моделью, записывает в ClickHouse. Проверена целостность (данные в ClickHouse соответствуют отправленным + предсказания). Измерены задержки и throughput. Реализована вариативная часть. Защита: глубокое понимание стриминговой архитектуры.
Хорошо	Работа выполнена полностью, но отсутствует анализ производительности, или запись в ClickHouse не через JDBC (например, только консоль). Вариативная часть не выполнена. Защита: ответы на большинство вопросов.

Оценка	Критерии
Удовлетворительно	Пайплайн работает частично (например, Spark читает из Kafka, но UDF не работает или модель не загружается), или ClickHouse не используется. Отчёт неполный. Студент не различает микробатчинг и потоковую обработку.
Неудовлетворительно	Работа не представлена, или отсутствуют ключевые компоненты (нет Kafka, нет Spark), или код неработоспособен. Отчёт не соответствует требованиям.

Лабораторная работа №7.

Мониторинг дрейфа данных с Evidently AI

Цель работы: Освоить инструментарий библиотеки Evidently AI для обнаружения и анализа дрейфа данных (Data Drift) в производственных ML-системах. Научиться генерировать отчёты о дрейфе, интерпретировать статистические метрики (PSI, Jensen–Shannon, Wasserstein) и настраивать автоматическое оповещение при обнаружении значимых изменений распределения признаков.

Задачи:

- Загрузить исторические данные (reference), служащие эталоном (например, обучающую выборку), и данные нового периода (current), потенциально содержащие дрейф.
- С использованием библиотеки Evidently AI сгенерировать отчёт о дрейфе данных (DataDriftPreset).
- Проанализировать результаты: выявить признаки, в которых обнаружен дрейф, интерпретировать значения PSI, Jensen–Shannon, p-value.
- Настроить программную отправку уведомлений (алертов) при обнаружении дрейфа (например, в Telegram или Slack).
- Сравнить чувствительность различных методов детекции дрейфа (KS-тест, PSI, Wasserstein) на примере искусственно модифицированных данных.
- Сформулировать рекомендации по порогам срабатывания алертов в зависимости от бизнес-контекста.

Теоретическая часть

1. Типы дрейфа в ML-системах

В производственной среде качество модели может деградировать по нескольким причинам:

- **Дрейф данных (Data Drift)** – изменение распределения входных признаков. Модель обучалась на одних данных, а в продукционной среде получает данные с иной статистикой.
- **Концептуальный дрейф (Concept Drift)** – изменение взаимосвязи между признаками и целевой переменной. Зависимость, выученная моделью, перестаёт соответствовать реальности.
- **Дрейф предсказаний (Prediction Drift)** – изменение распределения выходов модели, которое может сигнализировать о скрытых проблемах ещё до получения истинных меток.

2. Метрики обнаружения дрейфа

Evidently AI автоматически подбирает метод детекции в зависимости от типа признака и объёма данных. Для табличных данных действуют следующие правила:

Для малых данных (≤ 1000 наблюдений в reference):

Тип признака	Метод	Условие дрейфа
Числовой ($n_unique > 5$)	Двухвыборочный критерий Колмогорова–Смирнова (KS-тест)	$p\text{-value} \leq 0,05$
Категориальный или числовой с $n_unique \leq 5$	Критерий хи-квадрат (χ^2)	$p\text{-value} \leq 0,05$
Бинарный ($n_unique \leq 2$)	Z-тест разности пропорций	$p\text{-value} \leq 0,05$

Для больших данных (> 1000 наблюдений в reference):

Тип признака	Метод	Порог дрейфа
Числовой ($n_unique > 5$)	Расстояние Вассерштейна (Wasserstein)	$> 0,1$

Тип признака	Метод	Порог дрейфа
Категориальный или числовой с $n_unique \leq 5$	Дивергенция Йенсена–Шеннона (Jensen–Shannon)	$> 0,1$

Важно: Порог 0,1 по Wasserstein и Jensen–Shannon является значением по умолчанию и может быть скорректирован в зависимости от чувствительности, требуемой бизнес-задачей.

Population Stability Index (PSI) – ещё одна распространённая метрика для мониторинга дрейфа, особенно в финансовой сфере. $PSI > 0,2$ интерпретируется как сильный дрейф.

3. Подходы к детекции дрейфа в разных типах данных

- **Числовые признаки:** сравнение распределений через KS-тест (малые данные) или расстояние Вассерштейна (большие данные).
- **Категориальные признаки:** сравнение частот категорий через χ^2 -тест или дивергенцию Йенсена–Шеннона.
- **Текстовые данные:** обучение бинарного классификатора (domain classifier), различающего reference и current выборки. Дрейф фиксируется, если ROC-AUC классификатора превышает заданный порог (по умолчанию 0,55 для больших данных).

4. Evidently AI: отчёты, тесты и алерты

Библиотека предоставляет два основных интерфейса:

- **Report** – для визуального анализа и исследования. Генерирует HTML-отчёты с интерактивными графиками.

Test Suite – для автоматизированной проверки в пайплайнах. Возвращает JSON с результатами прохождения/непрохождения тестов и может быть интегрирован с системами оповещения.

Методика выполнения работы

Пример практического выполнения

Сценарий: Система кредитного скоринга. Исторические данные (reference) – данные клиентов за 2023 год. Текущие данные (current) – данные за 2024 год, в которых искусственно изменено распределение признака `credit_score` (смещение в сторону более низких значений). Задача: обнаружить дрейф и отправить уведомление в Telegram.

1. Установка и импорт

```
bash
```

```
pip install evidently pandas scikit-learn
```

2. Подготовка синтетических данных

```
python
```

```
import pandas as pd
```

```
import numpy as np
```

```
from sklearn.datasets import make_classification
```

```
# Параметры датасета
```

```
n_features = 5
```

```
# Исторические данные (reference) – baseline
```

```
X_ref, y_ref = make_classification(
```

```
    n_samples=5000, n_features=n_features, random_state=42
```

```
)
```

```
ref_df = pd.DataFrame(X_ref, columns=[f'feature_{i}' for i in range(n_features)])
```

```
ref_df['target'] = y_ref
```

```
ref_df['credit_score'] = np.random.normal(700, 50, 5000) # Средний балл 700
```

```
# Текущие данные (current) – с дрейфом признака credit_score
```

```

X_cur, y_cur = make_classification(
    n_samples=5000, n_features=n_features, random_state=43
)
cur_df = pd.DataFrame(X_cur, columns=[f'feature_{i}' for i in range(n_features)])
cur_df['target'] = y_cur
cur_df['credit_score'] = np.random.normal(620, 60, 5000) # Сдвиг среднего

print(ref_df['credit_score'].describe())
print(cur_df['credit_score'].describe())

```

3. Генерация отчёта о дрейфе данных

python

```

from evidently.report import Report
from evidently.metric_preset import DataDriftPreset

# Создание отчёта
data_drift_report = Report(metrics=[DataDriftPreset()])

# Выполнение расчёта
data_drift_report.run(reference_data=ref_df, current_data=cur_df)

# Сохранение HTML-отчёта
data_drift_report.save_html("data_drift_report.html")

# Вывод сводки в консоль (JSON)
drift_result = data_drift_report.as_dict()
n_drifted = drift_result['metrics'][0]['result']['number_of_drifted_columns']
print(f"Количество признаков с дрейфом: {n_drifted}")

```

4. Программная проверка дрейфа с Test Suite

python

```

from evidently.test_suite import TestSuite
from evidently.tests import TestColumnDrift

# Создание тестового набора для целевого признака
test_suite = TestSuite(tests=[
    TestColumnDrift(column_name='credit_score')
])

test_suite.run(reference_data=ref_df, current_data=cur_df)

# Извлечение результата
test_result = test_suite.as_dict()
drift_detected = test_result['tests'][0]['result']['drift_detected']
p_value = test_result['tests'][0]['result']['p_value']
print(f"Дрейф обнаружен: {drift_detected}, p-value = {p_value:.4f}")

```

5. Настройка алерта в Telegram

python

```
import requests
```

```
import json
```

```
TELEGRAM_BOT_TOKEN = "YOUR_BOT_TOKEN"
```

```
TELEGRAM_CHAT_ID = "YOUR_CHAT_ID"
```

```
def send_telegram_alert(message):
```

```
    url = f"https://api.telegram.org/bot{TELEGRAM_BOT_TOKEN}/sendMessage"
```

```
    payload = {
```

```
        "chat_id": TELEGRAM_CHAT_ID,
```

```
        "text": message,
```

```
        "parse_mode": "HTML"
```

```

    }

    response = requests.post(url, data=payload)

    return response.ok

# Интеграция с результатом дрейфа
if drift_detected:
    alert_msg = (
        f"🔔 <b>Data Drift Alert!</b>\n"
        f"Признак: credit_score\n"
        f"p-value: {p_value:.4f}\n"
        f"Действие: требуется переобучение модели"
    )
    send_telegram_alert(alert_msg)
    print("Алерт отправлен в Telegram")
else:
    print("Дрейф не обнаружен, алерт не отправлен")

```

6. Результаты выполнения

- Отчёт `data_drift_report.html` показывает, что `credit_score` является дрейфующим признаком (Wasserstein distance > 0,1).
- Другие признаки (`feature_0`–`feature_4`) дрейфа не обнаружили. Алерт в Telegram успешно отправлен, уведомление содержит значение p-value.

Индивидуальные задания

Каждый вариант получает доменную область и типы признаков. Необходимо:

1. Сгенерировать или загрузить reference и current датасеты.
2. Внести контролируемые изменения в распределение как минимум двух признаков.

3. Выполнить детекцию дрейфа с `DataDriftPreset`.
4. Проанализировать, какие методы детекции были применены к каждому типу признаков.
5. Настроить алерт (Telegram, Slack или email) при обнаружении дрейфа.
6. В отчёте сравнить чувствительность KS-теста, PSI и Wasserstein на ваших данных.

Вариант	Домен	Reference данные	Current данные (искусственный дрейф)	Ключевые признаки
1	Кредитный скоринг	Данные за 2023 г.	Снижение среднего <code>credit_score</code> на 15%	<code>credit_score</code> , <code>income</code> , <code>debt_ratio</code>
2	Отток клиентов (телеком)	Данные за 1-й квартал	Увеличение <code>monthly_charges</code> на 20%	<code>monthly_charges</code> , <code>tenure</code> , <code>contract_type</code>
3	Обнаружение мошенничества	Транзакции 2023 г.	Рост доли ночных транзакций с 10% до 35%	<code>amount</code> , <code>hour_of_day</code> , <code>merchant_category</code>
4	Прогноз доставки	Летние месяцы	Зимние месяцы (рост времени доставки)	<code>distance_km</code> , <code>traffic_score</code> , <code>weather_score</code>

Вариант	Домен	Reference данные	Current данные (искусственный дрейф)	Ключевые признаки
5	Медицинская диагностика	Пациенты 2022 г.	Пациенты 2024 г. (сдвиг распределения возраста)	age, bmi, blood_pressure
6	Рекомендательная система	Поведение в будни	Поведение в выходные (изменение времени сессий)	session_duration, click_rate, category_preference
7	Промышленный IoT	Работа оборудования летом	Работа оборудования зимой (вибрация выше)	vibration, temperature, power_consumption
8	Образование (успеваемость)	Осенний семестр	Весенний семестр (снижение attendance)	attendance_rate, homework_score, test_score
9	E-commerce (конверсия)	Данные до распродажи	Данные во время распродажи (рост трафика)	page_views, cart_adds, device_type

Вариант	Домен	Reference данные	Current данные (искусственный дрейф)	Ключевые признаки
10	Текстовые отзывы (NLP)	Отзывы 2023 г.	Отзывы 2024 г. (сдвиг тональности в негатив)	review_text, sentiment (целевая), length
11	Энергопотребление	Потребление в 2023 г.	Потребление в 2024 г. (рост среднего)	energy_kwh, temperature, hour_of_day
12	Кибербезопасность	Сетевой трафик в дневное время	Трафик в ночное время (иной порт)	bytes, packets, protocol, port
13	Автомобильная телематика	Водители-мужчины (усреднённые стили)	Водители-женщины (иное распределение скорости)	speed, rpm, engine_temp, throttle
14	Социальная сеть	Активность пользователей 2023 г.	Активность после внедрения нового алгоритма	posts_per_week, likes_received,

Вариант	Домен	Reference данные	Current данные (искусственный дрейф)	Ключевые признаки
				engagement_rate
15	Финансовые транзакции	Транзакции в рублях	Транзакции после валютной волатильности (суммы↑)	amount, currency_type, merchant_id

Примечание: Для варианта 10 (текстовые данные) необходимо указать `column_mapping` с типом `text` для колонки `review_text`.

Контрольные вопросы

1. Каковы принципиальные различия между дрейфом данных, концептуальным дрейфом и дрейфом предсказаний?
2. Какие статистические тесты Evidently применяет для числовых признаков в зависимости от размера reference-выборки?
3. Что означают $p\text{-value} \leq 0,05$ и Wasserstein distance $> 0,1$? Как интерпретировать эти пороги?
4. Как библиотека Evidently обрабатывает пустые значения при расчёте дрейфа?
5. В чём преимущество Jensen–Shannon дивергенции перед KL-дивергенцией для категориальных признаков?
6. Как можно настроить автоматическое переобучение модели при обнаружении дрейфа?
7. Какие метрики следует использовать в условиях сильного дисбаланса классов?

8. Почему для текстовых данных Evidently обучает domain classifier, а не применяет прямое сравнение распределений?

9. Какие риски возникают при слишком низком пороге срабатывания алерта (например, 0,01 вместо 0,05)?

10. По результатам вашего варианта: какие признаки дрейфуют наиболее сильно и какие бизнес-действия следует предпринять?

7. Требования к отчёту

Отчёт оформляется по ГОСТ 7.32-2017. Содержит:

1. Титульный лист.
2. Цель и задачи работы.
3. Теоретическое обоснование (типы дрейфа, PSI, Jensen–Shannon, пороговые значения).
4. Практическая часть:
 - код генерации reference и current датасетов;
 - код создания отчёта Evidently (DataDriftPreset);
 - скриншот HTML-отчёта с выделенными дрейфующими признаками;
 - код интеграции с Telegram/Slack для отправки алерта.
5. Результаты выполнения (таблица дрейфующих признаков с метриками).
6. Анализ результатов (сравнение чувствительности KS-теста и PSI).
7. Ответы на контрольные вопросы.
8. Вывод.
9. Список литературы.

Вариативная часть для оценки «отлично»:

- Настроить custom_threshold для PSI или Wasserstein.
- Реализовать мониторинг на потоке данных: каждую неделю проверять новую выборку и отправлять агрегированный отчёт в MLflow.

Критерии оценивания

Оценка	Критерии
Отлично	Работа выполнена полностью. Reference и current датасеты корректно сформированы, дрейф искусственно внедрён. Evidently отчёт сгенерирован, проанализирован, дрейфующие признаки идентифицированы. Алерт в Telegram/Slack настроен и работает. Реализована вариативная часть (настройка порогов или потоковый мониторинг). Защита: глубокое понимание статистических методов детекции дрейфа.
Хорошо	Работа выполнена полностью, но отсутствует вариативная часть, либо алерт реализован только в консоль, либо не выполнен анализ сравнительной чувствительности методов. Защита: ответы на большинство вопросов.
Удовлетворительно	Дрейф обнаружен, но отчёт сгенерирован с ошибками (не указаны типы признаков), либо алерт не настроен, либо анализ результатов отсутствует. Студент путает типы дрейфа.
Неудовлетворительно	Работа не представлена, или Evidently не использован, или данные не подготовлены, или отчёт не соответствует требованиям.

Лабораторная работа №8.

Интерпретация модели с SHAP на веб-дашборде

Цель работы: Освоить методы объяснимого искусственного интеллекта (XAI) с использованием библиотеки SHAP (SHapley Additive exPlanations). Научиться вычислять вклады признаков для предсказаний модели, визуализировать глобальную и локальную интерпретируемость, а также создать интерактивный веб-дашборд на базе Streamlit для демонстрации результатов интерпретации.

Задачи:

- Обучить модель классификации или регрессии на реальном датасете.
- Вычислить SHAP-значения (Shapley values) для обученной модели с помощью библиотеки `shap`.
- Построить глобальные графики интерпретации:
 - `summary_plot` (глобальная важность признаков с распределением вкладов);
 - `bar_plot` (усреднённые абсолютные SHAP-значения).
- Построить локальные объяснения: `force_plot` или `waterfall_plot` для отдельного наблюдения.
- Создать веб-приложение на Streamlit, которое позволяет:
 - загружать модель и SHAP-эксплейнер;
 - выбирать наблюдение из тестовой выборки (или вводить свои данные);
 - отображать `force_plot` (интерактивный) и `summary_plot`;
 - выводить значения признаков и предсказание модели.
- Проанализировать, какие признаки наиболее влияют на предсказания модели, и дать бизнес-интерпретацию.

Теоретическая часть

1. Необходимость интерпретируемости в ML

В производственных системах, особенно в высокорисковых областях (кредитный скоринг, медицина, кадровые решения), требуется не только точность модели, но и понимание причин, по которым принято то или иное решение. Это необходимо для:

- соблюдения регуляторных требований (GDPR, право на объяснение);
- отладки модели и обнаружения нежелательных смещений (bias);
- повышения доверия со стороны бизнес-пользователей.

2. SHAP (SHapley Additive exPlanations)

SHAP — унифицированный фреймворк для интерпретации предсказаний, основанный на теории игр (значения Шепли). Основная идея: предсказание модели раскладывается в сумму вкладов каждого признака плюс базовое значение (среднее предсказание по датасету).

Формально:

Пусть $f(x)$ — предсказание модели для экземпляра x , а $\phi_0 = E[f(x)]$ — среднее предсказание. Тогда

$$f(x) = \phi_0 + \sum_{j=1}^M \phi_j f(x) = \phi_0 + \sum_{j=1}^M \phi_j$$

где ϕ_j — вклад (Shapley value) j -го признака. Значения Шепли обладают свойствами: эффективность, симметричность, линейность, нулевой вклад для невливающих признаков.

Преимущества SHAP:

- согласованность с человеческим восприятием;
- возможность как глобальной, так и локальной интерпретации;
- поддержка любых моделей (древовидные, линейные, глубокие нейронные сети).

Типы визуализаций:

- `summary_plot` — показывает для каждого признака распределение SHAP-значений по всей выборке (цвет указывает на величину признака). Позволяет увидеть не только важность, но и направление влияния.

- `bar_plot` — средние абсолютные SHAP-значения (глобальная важность).
- `force_plot` — визуализирует вклад признаков для одного наблюдения в виде «силы», толкающей предсказание от базового значения к финальному.
- `waterfall_plot` — альтернатива `force_plot`, более детальная.
- `dependence_plot` — показывает зависимость SHAP-значения признака от его значения (может выявлять нелинейные эффекты).

3. Streamlit для интерактивных дашбордов

Streamlit — фреймворк на Python для быстрого создания веб-приложений для анализа данных и ML. Встроенные компоненты позволяют создавать интерактивные слайдеры, загрузчики файлов и отображать графики SHAP (HTML-виджеты, возвращаемые `shap.plots.force`). Это делает Streamlit идеальным инструментом для демонстрации интерпретации моделей.

Пример практического выполнения

Сценарий: Датасет «Heart Disease» (UCI). Обучается случайный лес (RandomForestClassifier) для предсказания наличия заболевания. Строится Streamlit-приложение для интерпретации.

1. Установка библиотек

```
bash
```

```
pip install shap streamlit pandas scikit-learn matplotlib
```

2. Обучение модели и вычисление SHAP (ноутбук / скрипт)

```
python
```

```
import pandas as pd
```

```
import numpy as np
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.model_selection import train_test_split
```

```
import shap
```

```
# Загрузка данных

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/heart-diseas
e/processed.cleveland.data"

columns = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'o
ldpeak', 'slope', 'ca', 'thal', 'target']

df = pd.read_csv(url, names=columns, na_values='?')
df.dropna(inplace=True)
df['target'] = (df['target'] > 0).astype(int) # бинарная классификация

X = df.drop('target', axis=1)
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random
_state=42)

# Обучение модели
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Вычисление SHAP (TreeExplainer для деревьев)
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test) # возвращает список [shap д
ля класса 0, shap для класса 1]

# Для бинарной классификации обычно берём shap_values[1]
shap_values_class1 = shap_values[1]

# Сохранение модели и explainer (через joblib)
import joblib
joblib.dump(model, 'heart_model.joblib')
```

```
joblib.dump(explainer, 'heart_explainer.joblib')
```

3. Streamlit-приложение (app.py)

```
python
```

```
import streamlit as st
```

```
import pandas as pd
```

```
import numpy as np
```

```
import joblib
```

```
import shap
```

```
import matplotlib.pyplot as plt
```

```
st.set_page_config(page_title="SHAP Model Interpreter", layout="wide")
```

```
st.title("🔍 Интерпретация модели с SHAP")
```

```
st.markdown("Визуализация глобальной и локальной важности признаков")
```

```
# Загрузка модели и explainer
```

```
@st.cache_resource
```

```
def load_model():
```

```
    model = joblib.load('heart_model.joblib')
```

```
    explainer = joblib.load('heart_explainer.joblib')
```

```
    return model, explainer
```

```
model, explainer = load_model()
```

```
# Загрузка тестового датасета (можно загрузить X_test из файла)
```

```
@st.cache_data
```

```
def load_test_data():
```

```
    # Здесь для простоты сгенерируем синтетический X_test (в реальности  
    и загрузить сохранённый)
```

```
    # В демо-целях используем датасет из sklearn
```

```

from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
X_test = pd.DataFrame(data.data, columns=data.feature_names)[:100]
return X_test

X_test = load_test_data()

# ----- Глобальная интерпретация -----
st.header("🌐 Глобальная интерпретация (вся тестовая выборка)")

# Для демонстрации используем первые 100 объектов
sample_X = X_test.iloc[:100]
shap_values_sample = explainer.shap_values(sample_X)
# Если explainer возвращает список, берём нужный класс
if isinstance(shap_values_sample, list):
    shap_values_sample = shap_values_sample[1] # для бинарной классифи
кации

# Summary plot
st.subheader("Summary plot (важность признаков с направлением влияния)")
fig, ax = plt.subplots(figsize=(10, 6))
shap.summary_plot(shap_values_sample, sample_X, show=False)
st.pyplot(fig)

# Bar plot
st.subheader("Bar plot (средние абсолютные SHAP-значения)")
fig2, ax2 = plt.subplots(figsize=(8, 5))
shap.summary_plot(shap_values_sample, sample_X, plot_type="bar", show=False)

```

```

st.pyplot(fig2)

# ----- Локальная интерпретация -----

st.header("👁 Локальная интерпретация (отдельное наблюдение)")

# Выбор наблюдения: либо ввод индекса, либо ручной ввод
option = st.radio("Выберите способ:", ("По индексу из тестовой выборки", "Ввести свои данные"))

if option == "По индексу из тестовой выборки":
    idx = st.number_input("Индекс наблюдения (0..{})".format(len(sample_X)-1), min_value=0, max_value=len(sample_X)-1, value=0)
    instance = sample_X.iloc[[idx]]
    st.write("Значения признаков:")
    st.dataframe(instance.T.style.format("{:.2f}"))
else:
    st.markdown("Введите значения признаков (в том же порядке, что и в датасете):")

    # Здесь упрощённо: для heart disease нужно 13 полей. Сгенерируем пример.
    input_dict = {}
    for col in sample_X.columns:
        input_dict[col] = st.number_input(f"{col}", value=float(sample_X[col].median()))
    instance = pd.DataFrame([input_dict])

# Предсказание модели
pred_proba = model.predict_proba(instance)[0][1]
pred_class = model.predict(instance)[0]

```

```
st.metric("Предсказание модели", f"Класс {pred_class} (вероятность пол  
ожительного: {pred_proba:.3f})")
```

```
# SHAP force plot для этого экземпляра
```

```
shap_instance = explainer.shap_values(instance)
```

```
if isinstance(shap_instance, list):
```

```
    shap_instance = shap_instance[1]
```

```
st.subheader("Force plot (вклад признаков в предсказание)")
```

```
# Для отображения в Streamlit используем HTML-виджет
```

```
force_plot_html = shap.force_plot(explainer.expected_value[1] if isinstance(  
explainer.expected_value, list) else explainer.expected_value,  
                                shap_instance,  
                                instance,  
                                matplotlib=False,  
                                show=False)
```

```
st.components.v1.html(force_plot_html.html_content, height=400)
```

```
# Альтернативно: waterfall plot
```

```
st.subheader("Waterfall plot (детализация вкладов)")
```

```
fig3, ax3 = plt.subplots(figsize=(10, 6))
```

```
shap.waterfall_plot(shap.Explanation(values=shap_instance[0],  
                                   base_values=explainer.expected_value[1] if isinstance(  
explainer.expected_value, list) else explainer.expected_value,  
                                   data=instance.iloc[0],  
                                   feature_names=instance.columns.tolist()),  
                   show=False)
```

```
st.pyplot(fig3)
```

```
st.markdown("---")
```

`st.caption("SHAP (SHapley Additive exPlanations) позволяет объяснить, как каждый признак повлиял на предсказание.")`

4. Запуск приложения

```
bash
```

```
streamlit run app.py
```

Результат:

Открывается веб-интерфейс, показывающий глобальную важность признаков (`summary_plot`, `bar_plot`) и локальное объяснение для выбранного пользователем наблюдения (`force_plot`, `waterfall_plot`). Пользователь может видеть, например, что высокое значение `thalach` (максимальная ЧСС) увеличивает предсказание в сторону «нет болезни», а `oldpeak` (депрессия ST) — в сторону «есть болезнь».

Индивидуальные задания

Каждый вариант получает датасет, модель и тип задачи.

Необходимо:

1. Обучить указанную модель (или разрешено использовать любую подходящую).
2. Вычислить SHAP-значения с использованием соответствующего объяснителя (`TreeExplainer` для деревьев, `KernelExplainer` для любых моделей, `DeepExplainer` для нейросетей).
3. Построить все базовые графики: `summary_plot` (с цветом), `bar_plot`, `dependence_plot` для двух самых важных признаков.
4. Создать Streamlit-приложение, позволяющее:
 - загружать модель (или обучать её при запуске);
 - выбирать наблюдение для локального `force_plot`;
 - отображать глобальную важность.
5. В отчёте дать интерпретацию: какие признаки наиболее значимы и почему (с точки зрения предметной области).

Вариант	Датасет / источник	Целевая переменная	Модель (рекомендуемая)
1	Boston Housing (регрессия)	medv (цена дома)	RandomForestRegressor
2	Breast Cancer Wisconsin	malignant (0/1)	XGBoostClassifier
3	Diabetes (Pima Indians)	Outcome (0/1)	LightGBM
4	Telco Churn	Churn (Yes/No)	RandomForestClassifier
5	Credit Card Fraud (sample)	Class (0/1)	XGBoost (с балансировкой)
6	California Housing	MedHouseVal	GradientBoostingRegressor
7	Student Performance (GPA)	G3 (числовая)	RandomForestRegressor
8	Titanic	Survived (0/1)	XGBoostClassifier
9	Wine Quality (red)	quality (3-8, классификация)	RandomForestClassifier
10	Adult Income	income (>50K)	LightGBM
11	Bike Sharing (count)	count (регрессия)	RandomForestRegressor
12	Air Quality (UCI)	CO(GT) (регрессия)	GradientBoostingRegressor
13	Default of Credit Card Clients	default (0/1)	XGBoostClassifier

Вариант	Датасет / источник	Целевая переменная	Модель (рекомендуемая)
14	MNIST (уменьшенный, 10 классов)	digit (0-9)	LogisticRegression (или MLP)
15	Text Classification (20 newsgroups subset)	topic (2 класса)	LogisticRegression с TF-IDF

Примечания:

- Для варианта 14 и 15 потребуется использовать `KernelExplainer` (медленнее) или `DeepExplainer` (для нейросетей).

- Для регрессии `shap_values` обычно одномерные, для классификации – выбираем `shap` для положительного класса.

Streamlit-приложение должно корректно обрабатывать тип задачи (регрессия или классификация).

Контрольные вопросы

- Что такое значение Шепли (Shapley value) и какие аксиомы лежат в его основе?
- В чём отличие `TreeExplainer` от `KernelExplainer`? Когда какой следует использовать?
- Как интерпретировать положительное и отрицательное SHAP-значение для признака?
- Что показывает `summary_plot` с цветовой шкалой? Почему красные точки справа означают, что высокое значение признака увеличивает предсказание?

5. Для чего используется `dependence_plot`? Как он помогает выявить нелинейные эффекты?
6. Как в `force_plot` интерпретировать базовое значение (base value) и итоговое предсказание?
7. Почему SHAP-значения могут быть одинаковыми для разных моделей, дающих разные предсказания?
8. Какие ограничения есть у SHAP? (вычислительная сложность, коррелированные признаки)
9. Как в Streamlit отобразить интерактивный `force_plot` (HTML-виджет)?
10. По результатам вашего варианта: какие два признака сильнее всего влияют на предсказание и каково это влияние (положительное/отрицательное)?

Требования к отчёту

Отчёт оформляется по ГОСТ 7.32-2017. Содержит:

1. Титульный лист.
2. Цель и задачи.
3. Теоретическое обоснование (SHAP, Шепли, виды графиков).
4. Практическая часть:
 - код обучения модели и вычисления SHAP (можно в виде ноутбука);
 - код Streamlit-приложения;
 - скриншоты дашборда (`summary_plot`, `bar_plot`, `force_plot` для выбранного наблюдения).
5. Результаты выполнения: таблица важности признаков с SHAP-значениями.
6. Анализ результатов (какие признаки оказались важными, почему это логично/нелогично с точки зрения предметной области).
7. Ответы на контрольные вопросы.

8. Вывод.
9. Список литературы.

Вариативная часть для оценки «отлично»:

- Добавить в Streamlit-приложение зависимостный график (dependence_plot) с интерактивным выбором признака.
- Реализовать сравнение интерпретаций для двух моделей (например, дерево решений и случайный лес).

Использовать `shap.Explainer` (универсальный) вместо специфических.

Критерии оценивания

Оценка	Критерии
Отлично	Работа выполнена полностью. Модель обучена, SHAP-значения корректно вычислены. Все обязательные графики (summary, bar, force) построены и интерпретированы. Streamlit-приложение работает без ошибок, позволяет выбрать наблюдение и отображает интерактивный force_plot. Реализована вариативная часть. Защита: свободное владение материалом SHAP.
Хорошо	Работа выполнена полностью, но отсутствует вариативная часть, либо приложение не отображает force_plot (только статические картинки), либо не проанализирована бизнес-интерпретация. Защита: ответы на большинство вопросов.
Удовлетворительно	SHAP-значения вычислены, но графики построены не полностью (например, только bar_plot), или Streamlit-приложение не работает (ошибки), или интерпретация отсутствует. Студент не различает локальную и глобальную интерпретацию.
Неудовлетворительно	Работа не представлена, или SHAP не использован, или модель не обучена, или отчёт не соответствует требованиям.

Лабораторная работа №9.

Fine-tuning предобученного BERT для классификации текстов

Цель работы: Освоить технологию адаптации (fine-tuning) предобученных трансформерных моделей (BERT) для задач классификации текстов с использованием экосистемы Hugging Face. Научиться загружать предобученную модель, токенизировать тексты, дообучать модель с помощью Trainer API и сравнивать качество fine-tuned BERT с классическим подходом на основе TF IDF и логистической регрессии.

Задачи:

- Загрузить датасет для бинарной или многоклассовой классификации текстов.
- Выполнить предобработку данных: очистку текста (при необходимости) и разделение на обучающую/валидационную/тестовую выборки.
- Реализовать baseline: векторизация текстов с помощью `TfidfVectorizer` и обучение `LogisticRegression`. Измерить качество (accuracy, F1-score).
- Загрузить предобученную модель (например, `distilbert-base-uncased`) и соответствующий токенизатор из Hugging Face.
- Токенизировать тексты с учётом максимальной длины (padding, truncation).
- Настроить параметры дообучения (`TrainingArguments`) и запустить обучение с использованием `Trainer`.
- Оценить качество fine-tuned модели на тестовой выборке.
- Сравнить результаты TF-IDF + логистической регрессии и fine-tuned BERT (метрики, время обучения, размер модели).
- Сформулировать вывод об эффективности трансформеров для данной задачи.

Теоретическая часть

1. Трансформеры и BERT

BERT (Bidirectional Encoder Representations from Transformers) — архитектура на основе трансформера, обученная на больших корпусах текста (Wikipedia + BookCorpus) с использованием двух задач: masked language modeling (MLM) и next sentence prediction (NSP). BERT учитывает контекст слева и справа, что даёт глубокие контекстуализированные эмбединги.

DistilBERT — облегчённая версия BERT, обученная с использованием дистилляции знаний. Сохраняет 95% качества при уменьшении размера на 40% и ускорении инференса на 60%.

2. Fine-tuning (дообучение)

Fine-tuning — процесс дальнейшего обучения предобученной модели на целевой задаче с небольшим датасетом. Все веса модели обновляются (или часть замораживается), а к выходу трансформера добавляется классификационная головка (линейный слой). Для классификации текстов используется выход [CLS]-токена, который подаётся на линейный классификатор.

Преимущества fine-tuning:

- использование знаний, выученных на миллиардах слов;
- хорошее качество даже при ограниченном размере целевого датасета (от нескольких тысяч примеров);
- возможность обработки сложных языковых явлений (омонимия, перенос смысла).

3. Hugging Face экосистема

Библиотека `transformers` предоставляет:

- готовые архитектуры (`BertForSequenceClassification`, `DistilBertForSequenceClassification`);
- токенизаторы (`AutoTokenizer`) с поддержкой subword tokenization (`WordPiece`);

- `Trainer` — высокоуровневый API для обучения с поддержкой чекпоинтов, логирования, `mixed precision`.

`datasets` — библиотека для эффективной загрузки и обработки датасетов (кэширование, многопоточность).

4. Сравнение с TF-IDF + LogisticRegression

Классический подход:

- TF-IDF преобразует тексты в разреженные числовые векторы, основанные на частоте слов. Не учитывает порядок слов и контекст.

- Логистическая регрессия обучается быстро, интерпретируема, но уступает нейросетевым методам при наличии сложных семантических зависимостей.

Ожидается, что BERT превзойдёт TF-IDF, особенно на датасетах с тонкой семантикой (тональность, сарказм, стиль). Однако BERT требует больше вычислительных ресурсов (GPU) и времени обучения.

Пример практического выполнения

Сценарий: Классификация тональности отзывов на фильмы IMDb (бинарная: положительный/отрицательный). Используется DistilBERT-base-uncased. Сравнение с TF-IDF + LR.

1. Установка библиотек

```
bash
```

```
pip install transformers datasets torch scikit-learn pandas tqdm
```

2. Загрузка и подготовка данных

```
python
```

```
import numpy as np
```

```
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import accuracy_score, f1_score, classification_report
```

```
from datasets import Dataset, DatasetDict
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    TrainingArguments,
    Trainer,
    EarlyStoppingCallback
)
import torch

# Загрузка IMDb (можно из datasets)
from datasets import load_dataset
imdb = load_dataset("imdb")

# Используем малую выборку для ускорения (полный датасет ~25k)
small_train = imdb["train"].shuffle(seed=42).select(range(2000))
small_test = imdb["test"].shuffle(seed=42).select(range(500))

# Преобразование в pandas для TF-IDF
train_texts = small_train["text"]
train_labels = small_train["label"]
test_texts = small_test["text"]
test_labels = small_test["label"]

3. Baseline: TF-IDF + LogisticRegression

python

vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1,2))
X_train_tfidf = vectorizer.fit_transform(train_texts)
X_test_tfidf = vectorizer.transform(test_texts)

lr = LogisticRegression(max_iter=1000)
```

```
lr.fit(X_train_tfidf, train_labels)
y_pred_lr = lr.predict(X_test_tfidf)
```

```
print("=== TF-IDF + LogisticRegression ===")
print(f'Accuracy: {accuracy_score(test_labels, y_pred_lr):.4f}')
print(f'F1-score (macro): {f1_score(test_labels, y_pred_lr, average='macro'):
.4f}')

print(classification_report(test_labels, y_pred_lr))
```

4. Fine-tuning DistilBERT

```
python
```

```
model_name = "distilbert-base-uncased"
```

```
tokenizer = AutoTokenizer.from_pretrained(model_name)
```

```
def tokenize_function(examples):
```

```
    return tokenizer(
        examples["text"],
        padding="max_length",
        truncation=True,
        max_length=512
    )
```

```
# Преобразование datasets в формат Hugging Face
```

```
dataset_train = Dataset.from_dict({"text": train_texts, "label": train_labels})
```

```
dataset_test = Dataset.from_dict({"text": test_texts, "label": test_labels})
```

```
tokenized_train = dataset_train.map(tokenize_function, batched=True)
```

```
tokenized_test = dataset_test.map(tokenize_function, batched=True)
```

```
# Загрузка модели для классификации (2 класса)
```

```
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)
```

Параметры обучения

```
training_args = TrainingArguments(  
    output_dir="./results",  
    evaluation_strategy="epoch",  
    save_strategy="epoch",  
    learning_rate=2e-5,  
    per_device_train_batch_size=16,  
    per_device_eval_batch_size=16,  
    num_train_epochs=3,  
    weight_decay=0.01,  
    logging_dir="./logs",  
    logging_steps=10,  
    load_best_model_at_end=True,  
    metric_for_best_model="accuracy"  
)
```

```
def compute_metrics(eval_pred):  
    logits, labels = eval_pred  
    predictions = np.argmax(logits, axis=-1)  
    acc = accuracy_score(labels, predictions)  
    f1 = f1_score(labels, predictions, average="macro")  
    return {"accuracy": acc, "f1": f1}
```

```
trainer = Trainer(  
    model=model,  
    args=training_args,  
    train_dataset=tokenized_train,  
    eval_dataset=tokenized_test,  
    tokenizer=tokenizer,
```

```
compute_metrics=compute_metrics,
callbacks=[EarlyStoppingCallback(early_stopping_patience=1)]
)
```

```
# Запуск обучения
```

```
trainer.train()
```

```
# Оценка на тесте
```

```
eval_results = trainer.evaluate()
```

```
print("=== DistilBERT fine-tuned ===")
```

```
print(f'Accuracy: {eval_results["eval_accuracy"]:.4f}')
```

```
print(f'F1-score: {eval_results["eval_f1"]:.4f}')
```

5. Сравнение результатов (пример)

Модель	Accuracy	F1-score (macro)	Время обучения (сек)	Размер модели
TF-IDF + LR	0.856	0.853	2	50 МБ (векторизатор + модель)
DistilBERT fine-tuned	0.892	0.891	180 (на CPU) / 45 (на GPU)	260 МБ

Вывод: Fine-tuned DistilBERT превосходит TF-IDF + LR по качеству на 3–4 процентных пункта, но требует значительно больше вычислительных ресурсов. Для приложений с высокими требованиями к точности и возможностью использовать GPU, трансформеры предпочтительнее.

Индивидуальные задания

Каждый вариант получает задачу классификации текстов на русском или английском языке. Необходимо:

1. Загрузить датасет (указанный в варианте).
2. Выполнить baseline: TF-IDF + LogisticRegression (или другой классический метод).
3. Выполнить fine-tuning DistilBERT (или `bert-base-uncased`/`rubert-tiny2` для русского).
4. Сравнить метрики (accuracy, precision, recall, F1).
5. Провести анализ ошибок (misclassified examples).
6. Сделать вывод о целесообразности применения трансформеров для данного датасета.

Вариант	Язык	Датасет / задача	Размер выборки (рекомендуемый для учебных целей)
1	Английский	SST-2 (тональность фраз)	6 920 train / 872 test
2	Английский	Spam Detection (SMS Spam Collection)	4 000 train / 1 000 test
3	Русский	RuTweetSent (тональность твитов)	5 000 train / 1 000 test
4	Английский	AG News (категории новостей, 4 класса)	10 000 train / 2 000 test
5	Русский	RuReviews (отзывы на фильмы, тональность)	3 000 train / 500 test
6	Английский	Toxic Comments (токсичность, бинарная)	5 000 train / 1 000 test

Вариант	Язык	Датасет / задача	Размер выборки (рекомендуемый для учебных целей)
7	Русский	LCQuAD (вопросы по классам) – адаптированный	2 000 train / 400 test
8	Английский	Emotion Dataset (6 эмоций)	4 000 train / 1 000 test
9	Русский	RuSentiTweet (тональность, 3 класса)	4 000 train / 800 test
10	Английский	Subjectivity (объектив/субъектив)	5 000 train / 1 000 test
11	Русский	KhresChat (чат-боты, диалоговые акты) – упрощённый	3 000 train / 600 test
12	Английский	Fake News (правда/фейк)	5 000 train / 1 000 test
13	Русский	RuCode (классификация кода vs естественный язык)	2 000 train / 400 test
14	Английский	IMDb Large (полный датасет, но взять сэмпл)	12 500 train / 12 500 test (или сэмпл 5k/1k)
15	Русский	RuBERT fine-tuning на задаче определения пола автора	3 000 train / 600 test

Примечания:

- Для русскоязычных датасетов рекомендуется использовать `DeepPavlov/rubert-tiny2` (компактный) или `cointegrated/rubert-tiny` для ускорения.
- При отсутствии GPU разрешается уменьшить `max_length` до 256 и количество эпох до 2.

Допускается использование `Trainer` или низкоуровневого PyTorch с `AdamW`.

Контрольные вопросы

1. Что такое fine-tuning в контексте трансформеров и почему он работает лучше обучения с нуля?
2. Какую роль выполняет токенизатор BERT? В чём отличие от простого split по пробелам?
3. Почему BERT использует специальные токены `[CLS]` и `[SEP]`? Как используется `[CLS]` для классификации?
4. Какие гиперпараметры являются наиболее критичными при fine-tuning (learning rate, batch size, количество эпох)?
5. Что такое `EarlyStoppingCallback` и как он помогает избежать переобучения?
6. Какие метрики использованы для сравнения baseline и fine-tuned модели? Почему F1 может быть предпочтительнее accuracy?
7. Как изменится качество, если увеличить `max_length` с 128 до 512? Какие компромиссы возникают?
8. В чём преимущества DistilBERT перед BERT base? Когда вы предпочли бы BERT base?
9. Как вычислительная сложность fine-tuning BERT соотносится с обучением логистической регрессии на TF-IDF?
10. По результатам вашего варианта: на каких примерах BERT ошибается, а логистическая регрессия права, и наоборот? Сделайте предположение о причинах.

Требования к отчёту

Отчёт оформляется по ГОСТ 7.32-2017. Содержит:

1. Титульный лист.
2. Цель и задачи.

3. Теоретическое обоснование (трансформеры, fine-tuning, Hugging Face).
4. Практическая часть:
 - код загрузки и предобработки данных;
 - код обучения TF-IDF + LR;
 - код токенизации и fine-tuning BERT;
 - скриншоты процесса обучения (loss, accuracy).
5. Результаты выполнения: таблица сравнения метрик, время обучения, размеры моделей.
6. Анализ результатов (на каких данных BERT выигрывает/проигрывает, примеры ошибок).
7. Ответы на контрольные вопросы.
8. Вывод.
9. Список литературы.

Вариативная часть для оценки «отлично»:

- Реализовать заморозку нижних слоёв BERT (тренировать только классификационную головку и последние 2–3 слоя).
 - Использовать `WandbCallback` для логирования в Weights & Biases.
- Провести эксперимент с разными длинами последовательностей (128, 256, 512) и сравнить качество/скорость.

Критерии оценивания

Оценка	Критерии
Отлично	Работа выполнена полностью. Baseline реализован корректно. Fine-tuning BERT успешно выполнен (модель обучается, качество измерено). Проведено сравнение с классическим подходом, дана интерпретация результатов. Реализована вариативная часть. Отчёт содержит анализ ошибок. Защита: свободное владение темой fine-tuning.

Хорошо	Работа выполнена полностью, но отсутствует вариативная часть, или fine-tuning выполнен без сохранения лучшей модели (early stopping), или сравнение с TF IDF неполное (нет F1). Защита: ответы на большинство вопросов.
Удовлетворительно	Fine-tuning запущен, но не завершён (оборван), или качество BERT не превышает baseline (возможно из-за ошибок в токенизации). Отчёт не содержит анализа ошибок. Студент путает понятия предобучения и дообучения.
Неудовлетворительно	Работа не представлена, или BERT не использован, или код неработоспособен, или отчёт отсутствует.

Лабораторная работа №10.

Fine-tuning YOLOv8 для детекции пользовательских объектов

Цель работы: Освоить полный цикл адаптации (fine-tuning) современной модели компьютерного зрения YOLOv8 под задачу детекции пользовательских объектов. Научиться размечать датасет с использованием Roboflow, выполнять дообучение модели, экспортировать её в формат ONNX для оптимизированного инференса и разворачивать микросервис на FastAPI для приёма изображений и возврата детекций.

Задачи:

- Собрать или загрузить датасет изображений, содержащий целевые объекты.
- Выполнить разметку bounding box'ов с использованием Roboflow (или другого инструмента) и экспортировать датасет в формате YOLO.
- Установить Ultralytics YOLOv8 и настроить конфигурацию обучения (гиперпараметры, аугментации).
- Выполнить fine-tuning предобученной модели YOLOv8 (например, yolov8n.pt) на целевом датасете.
- Оценить качество обученной модели (mAP50, mAP50-95).
- Экспортировать модель в формат ONNX для ускоренного инференса на CPU.
- Разработать микросервис на FastAPI с эндпоинтами:
 - POST /detect – принимает изображение (base64 или multipart) и возвращает список обнаруженных объектов (координаты, класс, уверенность).
 - GET /health – проверка работоспособности.
- Протестировать сервис на тестовых изображениях и измерить время инференса (исходная модель vs ONNX).

Теоретическая часть

1. YOLO (You Only Look Once)

YOLO – семейство одноэтапных (one-stage) детекторов, которые предсказывают bounding boxes и классы непосредственно за один проход свёрточной сети. Ключевые особенности:

- Высокая скорость инференса (десятки–сотни fps) при конкурентоспособной точности.
- Разделение изображения на сетку $S \times S$, каждая ячейка отвечает за объекты, центр которых находится внутри.
- Предсказываются координаты, уверенность и вероятности классов.

YOLOv8 – последняя версия от Ultralytics, поддерживающая:

- легкость настройки (единый CLI и Python API);
- продвинутые аугментации (Mosaic, MixUp, копирование-вставка);
- экспорт в ONNX, TensorRT, CoreML, TFLite.

2. Fine-tuning детектора

Предобученная на COCO (80 классов) модель YOLOv8 уже умеет распознавать общие объекты (люди, автомобили, стулья). Fine-tuning позволяет адаптировать её под уникальный набор классов, не обучая с нуля. Стратегии:

- **Заморозка backbone** (начальных слоёв) – тренируются только головы детекции; быстрее, но может не хватить для сильно отличающихся объектов.
- **Полное дообучение** (рекомендуется для небольших датасетов) – все веса обновляются.

3. Разметка датасета

Для обучения детектора необходимы изображения и аннотации – прямоугольники (bounding boxes) с метками классов. Roboflow предоставляет:

- удобный веб-интерфейс для разметки;

- автоматическую аугментацию и препроцессинг (ресайз, нормализация);

- экспорт в форматы YOLO, Pascal VOC, COCO.

Альтернативы: LabellImg, CVAT, [Makesense.ai](#) (бесплатные онлайн-инструменты).

4. ONNX и оптимизация инференса

Open Neural Network Exchange (ONNX) – открытый формат для представления моделей. Преимущества для YOLO:

- аппаратно-независимый граф вычислений;
- возможность использовать оптимизированные рантаймы (ONNX Runtime, OpenVINO);

- ускорение на CPU в 2-3 раза по сравнению с PyTorch.

Экспорт: `model.export(format="onnx", half=True)` – создаёт файл `.onnx`.

Инференс через ONNX Runtime требует минимальных изменений в коде.

5. Архитектура микросервиса

FastAPI обрабатывает входящие изображения, передаёт их в ONNX-сессию, выполняет постобработку (масштабирование координат, фильтрация по confidence) и возвращает JSON. Ключевые компоненты:

- загрузка модели при старте (singleton);
- асинхронная обработка (без блокировки);

валидация входных данных (размер, формат).

Пример практического выполнения

Сценарий: Детекция деталей Lego (3 класса: brick, plate, wheel). Датасет из 200 изображений, разметка в Roboflow, обучение YOLOv8n, экспорт в ONNX, FastAPI-сервис.

1. Разметка датасета

1. Регистрация на [Roboflow](#), создание проекта "Lego-detection".

2. Загрузка изображений (или использование готового датасета от Roboflow).
3. Разметка bounding box для каждого экземпляра детали.
4. Применение аугментаций: поворот на $\pm 15^\circ$, изменение яркости, blur.
5. Экспорт в формате YOLOv8 (создаётся архив с `data.yaml`, папками train/valid).

2. Установка и обучение

bash

```
pip install ultralytics roboflow onnx onnxruntime fastapi uvicorn python-multipart
```

python

```
from ultralytics import YOLO
import yaml
```

Загрузка конфигурации датасета

```
with open("Lego-1/data.yaml", "r") as f:
    data_config = yaml.safe_load(f)
```

Загрузка предобученной модели

```
model = YOLO("yolov8n.pt") # nano версия
```

Fine-tuning

```
results = model.train(
    data="Lego-1/data.yaml",
    epochs=50,
    imgsz=640,
    batch=16,
    device="cuda" if torch.cuda.is_available() else "cpu",
    patience=10,
```

```

    augment=True,
    project="lego_detector",
    name="yolov8n_finetune"
)

# Оценка на валидационном наборе
metrics = model.val()
print(f'mAP50: {metrics.box.map50:.4f}, mAP50-95: {metrics.box.map:.4f}
")

# Сохранение лучших весов
model.export(format="onnx", half=True, imgsz=640) # создаст lego_detect
or/weights/best.onnx

```

3. FastAPI сервис с ONNX Runtime

```

python
import io
import cv2
import numpy as np
import onnxruntime as ort
from fastapi import FastAPI, UploadFile, File, HTTPException
from pydantic import BaseModel
from typing import List, Optional
import base64

app = FastAPI(title="YOLOv8 ONNX Detector")

# Загрузка ONNX модели
session = ort.InferenceSession("lego_detector/weights/best.onnx", providers
=["CPUExecutionProvider"])
input_name = session.get_inputs()[0].name

```

```
output_name = session.get_outputs()[0].name
```

```
# Параметры
```

```
IMG_SIZE = 640
```

```
CONF_THRESHOLD = 0.5
```

```
IOU_THRESHOLD = 0.45
```

```
def letterbox(img, new_shape=(640, 640), color=(114,114,114)):
```

```
# Изменение размера с сохранением пропорций (добавление полей)
```

```
shape = img.shape[:2]
```

```
r = min(new_shape[0]/shape[0], new_shape[1]/shape[1])
```

```
new_unpad = int(round(shape[1]*r)), int(round(shape[0]*r))
```

```
img = cv2.resize(img, new_unpad, interpolation=cv2.INTER_LINEAR)
```

```
dw, dh = new_shape[1] - new_unpad[0], new_shape[0] - new_unpad[1]
```

```
dw //= 2; dh //= 2
```

```
img = cv2.copyMakeBorder(img, dh, dh, dw, dw, cv2.BORDER_CONSTANT,  
ANT, value=color)
```

```
return img, (dw, dh), r
```

```
def postprocess(pred, original_shape, pad, scale):
```

```
# pred shape: (1, 84, 8400) — 84 = 4 координаты + 80 классов COCO; н  
о при fine-tuning число классов меняется
```

```
# Упрощённая обработка для демонстрации (реальная сложнее)
```

```
# Возвращает список detections: [ [x1,y1,x2,y2, confidence, class_id], ...
```

```
]
```

```
detections = []
```

```
pred = pred[0].transpose() # (8400, 84)
```

```
boxes, scores, class_ids = [], [], []
```

```
# Здесь нужна фильтрация по confidence и NMS. Для примера покаже  
м заглушку.
```

На практике используем `non\_max\_suppression` из ultralytics.utils.op

S

return detections

@app.post("/detect")

async def detect(file: UploadFile = File(...)):

Чтение изображения

contents = await file.read()

np_arr = np.frombuffer(contents, np.uint8)

img = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)

if img is None:

raise HTTPException(400, "Invalid image")

original_h, original_w = img.shape[:2]

Предобработка

img_padded, (pad_w, pad_h), scale = letterbox(img, (IMG_SIZE, IMG_SIZE))

img_input = img_padded / 255.0

img_input = img_input.transpose(2,0,1)[np.newaxis, ...].astype(np.float32)

Инференс ONNX

outputs = session.run([output_name], {input_name: img_input})

pred = outputs[0] # shape (1, num_predictions, 6) для YOLOv8 ONNX

Постобработка (NMS)

from ultralytics.utils.ops import non_max_suppression

Преобразуем в тензор PyTorch для удобства, затем обратно в numpy

import torch

pred_tensor = torch.tensor(pred)

```
dets = non_max_suppression(pred_tensor, conf_thres=CONF_THRESHOL
LD, iou_thres=IOU_THRESHOLD)[0]
```

```
result = []
```

```
if dets is not None and len(dets):
```

```
    for *xyxy, conf, cls in dets.cpu().numpy():
```

```
        # Масштабирование координат к исходному размеру
```

```
        x1 = int((xyxy[0] - pad_w) / scale)
```

```
        y1 = int((xyxy[1] - pad_h) / scale)
```

```
        x2 = int((xyxy[2] - pad_w) / scale)
```

```
        y2 = int((xyxy[3] - pad_h) / scale)
```

```
        # Классификация
```

```
        class_name = data_config['names'][int(cls)]
```

```
        result.append({
```

```
            "class": class_name,
```

```
            "confidence": float(conf),
```

```
            "bbox": [x1, y1, x2, y2]
```

```
        })
```

```
    return {"detections": result}
```

```
@app.get("/health")
```

```
async def health():
```

```
    return {"status": "ok"}
```

```
# Запуск: uvicorn main:app --host 0.0.0.0 --port 8000
```

4. Тестирование

```
bash
```

```
curl -X POST "http://localhost:8000/detect" -F "file=@test_lego.jpg"
```

Ответ:

```
json
```

```
{
  "detections": [
    {"class": "brick", "confidence": 0.92, "bbox": [120, 80, 210, 165]},
    {"class": "wheel", "confidence": 0.87, "bbox": [300, 150, 360, 190]}
  ]
}
```

Результаты:

- Обучение: 50 эпох за 25 минут на GPU (T4); mAP50 = 0.83.
- ONNX инференс: 12 мс на CPU (Intel i5) против 28 мс у PyTorch.

Индивидуальные задания

Каждый вариант выбирает категорию объектов (не менее 2–3 классов). Необходимо: собрать/найти датасет (≥ 150 изображений), разметить в Roboflow, выполнить fine-tuning YOLOv8n (или YOLOv8s), экспортировать в ONNX, реализовать FastAPI сервис, измерить качество и скорость.

Вариант	Объекты для детекции	Рекомендуемый источник данных	Классы
1	Детали на производственной линии	Собственная съёмка или поиск по запросу "industrial objects detection"	screw, nut, washer
2	Фрукты и овощи	Готовые датасеты (Fruit-360, но переразметить)	apple, banana, orange
3	Дорожные знаки (РФ)	Russian Traffic Sign Dataset	speed_limit_50, stop, crosswalk
4	Автомобильные номера (подсчёт)	CCPD (Chinese) или свои фото	license_plate

Вариант	Объекты для детекции	Рекомендуемый источник данных	Классы
5	Мебель в комнате	Images from IKEA catalog или Roboflow Universe	chair, table, sofa
6	Животные на ферме	Open Images Dataset v6 (subset)	cow, pig, chicken
7	Спортивный инвентарь	Roboflow "Sports balls"	soccer_ball, basketball, tennis_ball
8	Инструменты	Home Depot dataset	hammer, screwdriver, wrench
9	Транспорт на дороге (велосипеды, самокаты)	BDD100K (subset)	bicycle, scooter
10	Лица в масках	Face Mask Detection dataset	with_mask, without_mask
11	Медицинские инструменты (симуляция)	Roboflow "Surgical instruments"	scalpel, forceps, clamp
12	Упакованные товары на полке	Grocery products dataset	cereal_box, milk_carton, soda_can
13	Птицы на кормушке	CUB-200 (переразметить несколько видов)	sparrow, tit, finch

Вариант	Объекты для детекции	Рекомендуемый источник данных	Классы
14	Компьютерная периферия	Собственные фото	monitor, keyboard, mouse
15	Пожарные датчики и знаки	Fire safety dataset	smoke_detector, fire_extinguisher

Дополнительно:

- Для вариантов с одним классом (номера, маски) достаточно одного класса, но можно добавить "фон".

Допускается использовать готовую разметку из Roboflow Universe (без ручной разметки), но студент должен описать процесс адаптации.

Контрольные вопросы

1. В чем отличие fine-tuning детектора от обучения с нуля? Почему fine-tuning эффективнее?
2. Какие аугментации в YOLOv8 помогают улучшить обобщение на малых датасетах?
3. Что такое mAP (mean Average Precision) и как интерпретировать mAP50 и mAP50-95?
4. Каковы минимальные требования к датасету для успешного fine-tuning YOLOv8?
5. Зачем экспортировать модель в ONNX? Какие преимущества даёт ONNX Runtime?
6. Как в FastAPI обрабатывать изображения больших размеров, чтобы не превысить лимит памяти?
7. Как можно улучшить скорость инференса, если модель запускается на CPU (квантование, OpenVINO)?

8. Как в постобработке применить Non-Maximum Suppression (NMS) и почему это необходимо?
9. Какие метрики времени инференса (preprocess, inference, postprocess) следует замерять?
10. По результатам вашего варианта: как изменилось качество после fine-tuning по сравнению с предобученной моделью на COCO?

Требования к отчёту

Отчёт оформляется по ГОСТ 7.32-2017. Содержит:

1. Титульный лист.
2. Цель и задачи.
3. Теоретическое обоснование (YOLOv8, fine-tuning, Roboflow, ONNX, FastAPI).
4. Практическая часть:
 - скриншоты процесса разметки в Roboflow;
 - конфигурационный файл data.yaml;
 - код обучения с параметрами;
 - графики кривых обучения (loss, mAP);
 - код сервиса FastAPI;
 - примеры запросов и ответов.

Результаты выполнения: таблица метрик качества (mAP50, mAP50-95) до и после fine-tuning, сравнение времени инференса (PyTorch vs ONNX).

Критерии оценивания

Оценка	Критерии
Отлично	Работа выполнена полностью. Разметка датасета завершена (не менее 150 аннотированных изображений). Fine-tuning успешно завершён, получена модель с $mAP50 > 0.7$. Модель экспортирована в ONNX. FastAPI сервис работает, возвращает корректные детекции. Проведено сравнение

	производительности. Реализована вариативная часть. Защита: понимание архитектуры YOLO и ONNX.
Хорошо	Работа выполнена полностью, но датасет менее 100 изображений, или mAP50 < 0.6, или отсутствует сравнение ONNX vs PyTorch, или вариативная часть не выполнена. Защита: ответы на большинство вопросов.
Удовлетворительно	Fine-tuning выполнен, но модель не экспортирована в ONNX, или сервис FastAPI не реализован, или разметка не описана. Отчёт неполный. Студент не объясняет разницу между детекцией и классификацией.
Неудовлетворительно	Работа не представлена, или обучение не завершено, или код неработоспособен, или отсутствует сервис. Отчёт не соответствует требованиям.

Лабораторная работа №11.

Развёртывание модели в Docker + MLflow Tracking

Цель работы: Освоить практики управления экспериментами и воспроизводимости моделей машинного обучения с использованием MLflow в контейнеризированной среде. Научиться разворачивать инфраструктуру MLflow (tracking server, backend database, artifact store) с помощью Docker Compose, логировать параметры, метрики и артефакты моделей, а также запускать обучение внутри Docker-контейнера.

Задачи:

- Развернуть инфраструктуру MLflow с использованием Docker Compose:
 - **MLflow Tracking Server** – веб-интерфейс для просмотра экспериментов.
 - **PostgreSQL** – бэкенд для хранения метаданных экспериментов (параметры, метрики, теги).
 - **MinIO** (S3-совместимое хранилище) – для хранения артефактов (модели, графики, файлы).
- Создать Docker-образ для обучения модели, который:
 - загружает данные;
 - обучает модель (например, RandomForest или XGBoost);
 - логирует параметры, метрики (ROC-AUC, accuracy) и артефакты (confusion matrix, model.pkl) в MLflow.
- Запустить обучение внутри контейнера, убедиться, что эксперимент появляется в MLflow UI.
- Зарегистрировать лучшую модель в MLflow Model Registry.
- (Опционально) Развернуть serving контейнер, который загружает зарегистрированную модель из MLflow и запускает FastAPI-сервер для инференса.

Теоретическая часть

1. Проблема управления экспериментами в ML

В процессе разработки ML-моделей специалисты пробуют множество комбинаций гиперпараметров, версий данных, алгоритмов. Без систематического учёта возникает "экспериментальный хаос": невозможно воспроизвести результат, сравнить метрики, отследить, какая модель попала в production. MLflow решает эти проблемы.

2. Компоненты MLflow

MLflow – open-source платформа для управления жизненным циклом ML, включающая:

- **MLflow Tracking** – API и UI для логирования параметров, метрик, артефактов. Поддерживает несколько экспериментов, возможность сравнения runs.
- **MLflow Projects** – спецификация для воспроизводимых запусков (код + окружение).
- **MLflow Models** – стандартный формат упаковки модели (включая зависимости и функцию predict).
- **MLflow Model Registry** – централизованное хранилище моделей с этапами (Staging, Production, Archived).

Архитектура Tracking Server:

- **Backend Store** – база данных (PostgreSQL, MySQL, SQLite) для хранения метаданных экспериментов.
- **Artifact Store** – хранилище файлов (локальная файловая система, S3, Azure Blob, GCS, MinIO).

3. Почему Docker и MLflow вместе?

- **Воспроизводимость** – обучение и serving в одинаковых окружениях.
- **Изоляция** – зависимости не конфликтуют с хост-системой.
- **Масштабируемость** – легко поднять MLflow server в облаке или Kubernetes.

- **CI/CD** – интеграция с пайплайнами (GitHub Actions, GitLab CI).

4. MinIO как S3-совместимое хранилище

MinIO – высокопроизводительное объектное хранилище, совместимое с S3 API. Идеально для локального развёртывания (аналог AWS S3 на своём сервере). MLflow настраивается на MinIO через переменные окружения:

text

```
MLFLOW_S3_ENDPOINT_URL = http://minio:9000
```

```
AWS_ACCESS_KEY_ID = minioadmin
```

```
AWS_SECRET_ACCESS_KEY = minioadmin
```

Пример практического выполнения

Сценарий: Обучение модели классификации (датасет Breast Cancer) с логированием в MLflow. Инфраструктура: MLflow + PostgreSQL + MinIO. Обучение запускается в отдельном контейнере.

1. Структура проекта

text

mlflow-docker/

├── docker-compose.yml

├── Dockerfile.train

├── train.py

├── requirements.txt

└── .env

2. Файл `docker-compose.yml`

yaml

```
version: '3.8'
```

```
services:
```

```
  postgres:
```

```
    image: postgres:15
```

```
    container_name: mlflow_postgres
```

environment:

POSTGRES_USER: mlflow

POSTGRES_PASSWORD: mlflow123

POSTGRES_DB: mlflow_db

volumes:

- postgres_data:/var/lib/postgresql/data

ports:

- "5432:5432"

networks:

- mlflow_net

minio:

image: minio/minio:latest

container_name: mlflow_minio

command: server /data --console-address ":9001"

environment:

MINIO_ROOT_USER: minioadmin

MINIO_ROOT_PASSWORD: minioadmin

ports:

- "9000:9000"

- "9001:9001"

volumes:

- minio_data:/data

networks:

- mlflow_net

mlflow-server:

image: python:3.10-slim

container_name: mlflow_server

depends_on:

- postgres

- minio

environment:

MLFLOW_S3_ENDPOINT_URL: http://minio:9000

AWS_ACCESS_KEY_ID: minioadmin

AWS_SECRET_ACCESS_KEY: minioadmin

command: >

bash -c "pip install mlflow psycpg2-binary boto3 &&

mlflow server

--backend-store-uri postgresql://mlflow:mlflow123@postgres/mlflo

w_db

--default-artifact-root s3://mlflow-artifacts/

--host 0.0.0.0"

ports:

- "5000:5000"

networks:

- mlflow_net

volumes:

postgres_data:

minio_data:

networks:

mlflow_net:

driver: bridge

Перед запуском: Создать бакет `mlflow-artifacts` в MinIO (через веб-интерфейс <http://localhost:9001>, логин/пароль minioadmin/minioadmin).

3. Файл requirements.txt

text

mlflow==2.8.0

```
scikit-learn==1.3.0
pandas==2.0.3
numpy==1.24.3
psycopg2-binary==2.9.9
boto3==1.28.0
```

4. Файл `train.py`

```
python
```

```
import mlflow
import mlflow.sklearn
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matri
```

x

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import os

# Настройка MLflow (подключение к tracking server)
mlflow.set_tracking_uri("http://mlflow-server:5000")
mlflow.set_experiment("breast_cancer_classification")

# Загрузка данных
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Параметры модели
```

```
params = {  
    "n_estimators": 100,  
    "max_depth": 5,  
    "random_state": 42,  
    "criterion": "gini"  
}
```

```
# Обучение с логированием
```

```
with mlflow.start_run(run_name="RandomForest_v1") as run:
```

```
    # Логирование параметров
```

```
    mlflow.log_params(params)
```

```
# Обучение
```

```
model = RandomForestClassifier(**params)
```

```
model.fit(X_train, y_train)
```

```
# Предсказания и метрики
```

```
y_pred = model.predict(X_test)
```

```
y_proba = model.predict_proba(X_test)[:, 1]
```

```
accuracy = accuracy_score(y_test, y_pred)
```

```
roc_auc = roc_auc_score(y_test, y_proba)
```

```
cv_scores = cross_val_score(model, X, y, cv=5, scoring='roc_auc')
```

```
mlflow.log_metric("accuracy", accuracy)
```

```
mlflow.log_metric("roc_auc", roc_auc)
```

```
mlflow.log_metric("cv_roc_auc_mean", cv_scores.mean())
mlflow.log_metric("cv_roc_auc_std", cv_scores.std())
```

```
# Логирование confusion matrix как артефакта
```

```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.savefig("confusion_matrix.png")
mlflow.log_artifact("confusion_matrix.png")
plt.close()
```

```
# Логирование важности признаков
```

```
feature_importance = pd.DataFrame({
    'feature': data.feature_names,
    'importance': model.feature_importances_
}).sort_values('importance', ascending=False)
```

```
plt.figure(figsize=(10,6))
sns.barplot(data=feature_importance.head(10), x='importance', y='feature')
plt.title("Top 10 Feature Importances")
plt.savefig("feature_importance.png")
mlflow.log_artifact("feature_importance.png")
```

```
# Логирование модели (формат MLflow)
```

```
mlflow.sklearn.log_model(model, "model", registered_model_name="BreastCancerClassifier")
```

```
# Сохранение модели также в локальный файл (для демонстрации)
```

```
import joblib
```

```
joblib.dump(model, "model.joblib")
mlflow.log_artifact("model.joblib")

print(f'Run ID: {run.info.run_id}')
print(f'Accuracy: {accuracy:.4f}, ROC-AUC: {roc_auc:.4f}')

print("Обучение завершено. Проверьте MLflow UI на http://localhost:5000")
```

5. Файл **Dockerfile.train**

dockerfile

FROM python:3.10-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY train.py .

Переменные окружения для подключения к MLflow (будут переданы при запуске)

ENV MLFLOW_TRACKING_URI=http://mlflow-server:5000

CMD ["python", "train.py"]

6. Запуск инфраструктуры и обучения

bash

1. Запуск сервисов (MLflow, PostgreSQL, MinIO)

docker-compose up -d

2. Создание бакета в MinIO (один раз)

Через браузер: <http://localhost:9001> (login: minioadmin/minioadmin) создать бакет "mlflow-artifacts"

3. Сборка Docker-образа для обучения

```
docker build -t mlflow-train -f Dockerfile.train .
```

4. Запуск обучения (контейнер должен быть в одной сети с MLflow)

```
docker run --rm --network mlflow-docker_mlflow_net -e MLFLOW_TRACKING_URI=http://mlflow-server:5000 mlflow-train
```

7. Проверка результатов

Открыть в браузере <http://localhost:5000> → Experiments → breast_cancer_classification. Будут видны:

- Параметры (n_estimators, max_depth, ...)
- Метрики (accuracy, roc_auc, cv_roc_auc_mean)
- Артефакты (confusion_matrix.png, feature_importance.png, model.joblib)

Зарегистрированная модель "BreastCancerClassifier" в Model Registry

Индивидуальные задания

Каждый вариант должен:

1. Развернуть MLflow инфраструктуру (PostgreSQL + MinIO + MLflow) через docker-compose.
2. Выбрать датасет и модель в соответствии с вариантом.
3. Написать скрипт обучения с логированием в MLflow (параметры, метрики, артефакты).
4. Запустить обучение в Docker-контейнере.
5. Проверить, что все данные отображаются в MLflow UI.
6. Зарегистрировать модель в Model Registry с комментарием.
7. (Вариативно) Реализовать serving контейнер, который загружает модель из MLflow и запускает REST API.

Вариант	Датасет	Модель	Ключевые параметры для логирования
1	Iris (классификация)	LogisticRegression	C, solver, max_iter
2	Wine (классификация)	RandomForestClassifier	n_estimators, max_depth, min_samples_split
3	Diabetes (регрессия)	LinearRegression	fit_intercept, normalize (если есть)
4	Boston Housing (регрессия)	GradientBoostingRegressor	learning_rate, n_estimators, max_depth
5	Digits (классификация)	XGBoostClassifier	learning_rate, max_depth, subsample
6	Breast Cancer (классификация)	LightGBMClassifier	num_leaves, learning_rate, feature_fraction
7	Telco Churn (классификация)	RandomForestClassifier	n_estimators, max_features, min_samples_leaf
8	Credit Card Fraud (классификация)	XGBoostClassifier	scale_pos_weight, max_depth, reg_lambda
9	California Housing (регрессия)	RandomForestRegressor	n_estimators, max_depth, min_samples_split

Вариант	Датасет	Модель	Ключевые параметры для логирования
10	MNIST (уменьшенный)	MLPClassifier	hidden_layer_sizes, alpha, batch_size
11	Titanic (классификация)	GradientBoostingClassifier	learning_rate, n_estimators, subsample
12	Spambase (классификация)	LogisticRegression	penalty, C, solver
13	News Popularity (регрессия)	XGBoostRegressor	learning_rate, max_depth, reg_alpha
14	Adult Income (классификация)	RandomForestClassifier	n_estimators, max_depth, class_weight
15	Heart Disease (классификация)	LightGBMClassifier	num_leaves, min_child_samples, subsample

Дополнительные требования:

- Для регрессии логировать метрики: RMSE, MAE, R^2 .
- Всегда логировать confusion matrix (для классификации) или график actual vs predicted (для регрессии).

Использовать `mlflow.log_artifact` для сохранения любых дополнительных файлов.

Контрольные вопросы

1. Какие проблемы решает MLflow Tracking в процессе разработки ML-моделей?
2. Для чего нужны два хранилища: backend store (PostgreSQL) и artifact store (MinIO)?
3. Как MLflow обеспечивает воспроизводимость экспериментов?
4. В чём разница между `mlflow.log_param`, `mlflow.log_metric` и `mlflow.log_artifact`?
5. Как зарегистрировать модель в Model Registry и перевести её из Staging в Production?
6. Какие преимущества даёт использование MinIO по сравнению с локальной файловой системой для артефактов?
7. Как можно загрузить зарегистрированную модель из MLflow в отдельное приложение (например, FastAPI)?
8. Почему важно указывать версии зависимостей (в `requirements.txt` или `conda.yaml`) при логировании модели?
9. Как настроить MLflow для работы с удалённым S3 (например, AWS S3) вместо MinIO?
10. По результатам вашего варианта: какой эксперимент показал лучшую метрику и какие параметры использовались?

Требования к отчёту

Отчёт оформляется по ГОСТ 7.32-2017. Содержит:

1. Титульный лист.
2. Цель и задачи.
3. Теоретическое обоснование (MLflow, компоненты, Docker).
4. Практическая часть:
 - Файл `docker-compose.yml` и пояснения к сервисам.
 - Скрипт обучения с комментариями по логированию.
 - Команды для сборки и запуска.

- Скриншоты MLflow UI (список экспериментов, run с метриками и артефактами, Model Registry).
- 5. Результаты выполнения (таблица залогированных метрик, ссылки на артефакты).
- 6. Анализ результатов (сравнение нескольких запусков, если выполнялись).
- 7. Ответы на контрольные вопросы.
- 8. Вывод.
- 9. Список литературы.

Вариативная часть для оценки «отлично»:

- Реализовать второй контейнер для serving (FastAPI), который при запуске загружает модель из MLflow Model Registry по этапу "Production" и отвечает на запросы.
- Настроить автоматическое логирование `autolog()` для выбранной библиотеки (например, `mlflow.sklearn.autolog()`).

Провести несколько экспериментов с разными гиперпараметрами и сравнить их в MLflow UI.

Критерии оценивания

Оценка	Критерии
Отлично	Работа выполнена полностью. Инфраструктура (PostgreSQL, MinIO, MLflow) успешно развёрнута через docker-compose. Скрипт обучения логирует параметры, метрики и артефакты. Обучение запускается в отдельном Docker-контейнере. В MLflow UI отображаются все данные, модель зарегистрирована. Реализована вариативная часть (serving контейнер или autolog). Защита: глубокое понимание архитектуры MLflow.
Хорошо	Работа выполнена полностью, но отсутствует вариативная часть, или MinIO заменён локальным хранилищем (не S3),

	или модель не зарегистрирована в Registry. Защита: ответы на большинство вопросов.
Удовлетворительно	Инфраструктура запущена, но обучение происходит не в отдельном контейнере (запуск из ноутбука), или артефакты не сохраняются, или логируются не все обязательные метрики. Отчёт неполный. Студент не объясняет роль артефакт-стора.
Неудовлетворительно	Работа не представлена, или MLflow не используется, или контейнеры не запускаются, или отсутствует логирование. Отчёт не соответствует требованиям.

Лабораторная работа №12.

Создание CI/CD пайплайна для модели с GitHub Actions

Цель работы: Освоить практики непрерывной интеграции и непрерывного развёртывания (CI/CD) для ML-моделей с использованием GitHub Actions. Научиться автоматизировать процесс тестирования кода, проверки качества модели и деплоя Docker-образа при каждом изменении в репозитории.

Задачи:

- Подготовить репозиторий проекта с ML-моделью, включающий:
 - код обучения и предобработки;
 - тесты (pytest) для проверки функций предобработки, пайплайна и метрик качества;
 - Dockerfile для упаковки сервиса инференса (FastAPI).
- Настроить GitHub Actions workflow, который автоматически запускается при push в ветку `main` (или при создании pull request).
- В workflow реализовать следующие этапы:
 - установка зависимостей;
 - запуск линтера (flake8) и форматтера (black) – опционально;
 - запуск pytest для проверки юнит-тестов и интеграционных тестов;
 - проверка качества модели: обучение на тестовом подмножестве и проверка, что ROC-AUC > заданного порога (например, 0.75);
 - сборка Docker-образа сервиса инференса;
 - пуш Docker-образа в реестр (Docker Hub, GitHub Container Registry или другой);
 - (опционально) деплой на удалённый сервер (например, через SSH).
- Убедиться, что пайплайн срабатывает корректно при успешных тестах и блокирует слияние при падении тестов или низком качестве модели

Теоретическая часть

1. CI/CD для машинного обучения

CI/CD (Continuous Integration / Continuous Deployment) – практика автоматизации сборки, тестирования и развёртывания программного обеспечения. Для ML-проектов добавляется специфический этап: **валидация качества модели**. Если новая версия модели (или изменение кода) ухудшает метрику ниже порога, пайплайн должен завершаться ошибкой, предотвращая деплой.

Типовой CI/CD пайплайн для ML:

1. **Линтинг и форматирование** – обеспечение единого стиля кода.
2. **Юнит-тесты** – проверка функций препроцессинга, загрузки данных, трансформаций.
3. **Тест качества модели** – обучение на малой выборке или загрузка предобученной модели и вычисление метрики на тестовом наборе.
4. **Сборка Docker-образа** – упаковка сервиса инференса.
5. **Публикация образа** – в container registry (Docker Hub, GHCR, Amazon ECR).
6. **Деплой** – обновление сервера (Kubernetes, VM, облачная функция).

2. GitHub Actions

GitHub Actions – платформа для автоматизации внутри GitHub. Workflow описывается в YAML-файле в директории `.github/workflows/`. Основные концепции:

- **Event** – триггер (push, pull_request, schedule, workflow_dispatch).
- **Job** – набор шагов, выполняющихся на одном runner (виртуальная машина).
- **Step** – отдельное действие: запуск команды, использование предопределённого action.
- **Runner** – окружение (ubuntu-latest, windows-latest, macOS-latest или self-hosted).

Для ML-проекта часто используется `ubuntu-latest` с установкой Python, Docker, необходимых библиотек.

3. Тестирование ML-моделей

Помимо классических юнит-тестов (проверка размерности, типов данных, обработки пропусков), в CI/CD пайплайн включают:

- **Тест на переобучение** – качество на обучении и валидации не должно расходиться более чем на заданную величину.
- **Тест на воспроизводимость** – повторный запуск обучения с тем же random seed даёт те же метрики.
- **Порог качества** – ROC-AUC, F1, MAE не должны упасть ниже заданного значения (обычно определяемого бизнес-требованиями).

4. Docker и реестры

Для деплоя модели в production её удобно упаковать в Docker-образ, содержащий веб-сервер (FastAPI, Flask) и код загрузки модели. Образ публикуется в реестре:

- **Docker Hub** – публичный реестр; для приватных образов требуется платная подписка.
- **GitHub Container Registry (GHCR)** – интегрирован с GitHub, позволяет управлять доступом через права репозитория.
- **Amazon ECR, Google Container Registry** – облачные решения.

В GitHub Actions для аутентификации в реестре используются `secrets` (токены, логины, пароли).

Пример практического выполнения

Сценарий: Проект с моделью классификации (RandomForest на датасете Breast Cancer). Репозиторий содержит:

- `train.py` – обучение и сохранение модели.
- `inference.py` – FastAPI сервис загрузки модели и предсказаний.
- `Dockerfile` для сервиса.
- `tests/test_pipeline.py` – юнит-тесты.

- `.github/workflows/ci_cd.yml` – CI/CD пайплайн.

4.1 Структура репозитория

text

ml-ci-cd-demo/

```
├── .github/
│   └── workflows/
│       └── ci_cd.yml
├── app/
│   ├── __init__.py
│   ├── main.py          # FastAPI приложение
│   ├── model.py         # Загрузка модели
│   └── schemas.py       # Pydantic модели
├── models/
│   └── (пусто, модель создаётся при обучении)
├── tests/
│   ├── __init__.py
│   ├── test_preprocessing.py
│   └── test_model_quality.py
├── train.py
├── requirements.txt
├── Dockerfile
└── README.md
```

4.2 Файл `train.py` (упрощённый)

python

```
import joblib
```

```
import pandas as pd
```

```
from sklearn.datasets import load_breast_cancer
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import roc_auc_score
```

```

def train_model():
    data = load_breast_cancer()
    X = pd.DataFrame(data.data, columns=data.feature_names)
    y = data.target
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    y_proba = model.predict_proba(X_test)[:,-1]
    roc_auc = roc_auc_score(y_test, y_proba)
    print(f'ROC-AUC: {roc_auc:.4f}')

    joblib.dump(model, "models/model.joblib")
    return roc_auc

if __name__ == "__main__":
    train_model()

```

4.3 Файл app/main.py

```

python

from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import joblib
import numpy as np
import os

app = FastAPI()

```

```
# Загрузка модели при старте
```

```
MODEL_PATH = os.getenv("MODEL_PATH", "models/model.joblib")
```

```
model = joblib.load(MODEL_PATH)
```

```
class PredictRequest(BaseModel):
```

```
    features: list[float]
```

```
class PredictResponse(BaseModel):
```

```
    prediction: int
```

```
    probability: float
```

```
@app.post("/predict", response_model=PredictResponse)
```

```
def predict(request: PredictRequest):
```

```
    try:
```

```
        X = np.array(request.features).reshape(1, -1)
```

```
        proba = model.predict_proba(X)[0][1]
```

```
        pred = int(proba >= 0.5)
```

```
        return PredictResponse(prediction=pred, probability=float(proba))
```

```
    except Exception as e:
```

```
        raise HTTPException(status_code=400, detail=str(e))
```

```
@app.get("/health")
```

```
def health():
```

```
    return {"status": "ok"}
```

4.4 Файл Dockerfile

```
dockerfile
```

```
FROM python:3.10-slim
```

```
WORKDIR /app
```

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY app/ ./app/

COPY models/ ./models/

ENV MODEL_PATH=/app/models/model.joblib

EXPOSE 8000

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]

4.5 Файл requirements.txt

text

fastapi==0.104.1

uvicorn[standard]==0.24.0

scikit-learn==1.3.0

joblib==1.3.2

pandas==2.0.3

numpy==1.24.3

pytest==7.4.3

pytest-cov==4.1.0

4.6 Тесты (tests/test_model_quality.py)

python

import pytest

import joblib

import pandas as pd

from sklearn.datasets import load_breast_cancer

from sklearn.model_selection import train_test_split

from sklearn.metrics import roc_auc_score

```

def test_model_quality():
    # Загружаем обученную модель (если есть) или обучаем новую
    try:
        model = joblib.load("models/model.joblib")
    except FileNotFoundError:
        from train import train_model
        train_model()
        model = joblib.load("models/model.joblib")

    data = load_breast_cancer()
    X = pd.DataFrame(data.data, columns=data.feature_names)
    y = data.target
    _, X_test, _, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    y_proba = model.predict_proba(X_test)[:,-1]
    roc_auc = roc_auc_score(y_test, y_proba)

    # Порог качества
    THRESHOLD = 0.75
    assert roc_auc > THRESHOLD, f"ROC-AUC {roc_auc:.4f} ниже порога
{THRESHOLD}"

```

4.7 CI/CD workflow (`.github/workflows/ci_cd.yml`)

yaml

name: ML CI/CD Pipeline

on:

push:

branches: ["main"]

pull_request:

branches: ["main"]

workflow_dispatch: # ручной запуск

env:

REGISTRY: ghcr.io

IMAGE_NAME: \${{ github.repository }}/ml-inference

jobs:

test-and-build:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v4

- name: Set up Python

uses: actions/setup-python@v5

with:

python-version: '3.10'

- name: Install dependencies

run: |

python -m pip install --upgrade pip

pip install -r requirements.txt

- name: Lint with flake8

run: |

pip install flake8

flake8 app/ tests/ --max-line-length=120 --count --show-source --statisti

- **name:** Run unit tests

run: |

```
pytest tests/ -v --cov=app --cov-report=xml
```

- **name:** Train model and check quality

run: |

```
python train.py # должен создать model.joblib и вывести ROC-AUC
```

- **name:** Verify model quality (explicit threshold)

run: |

```
python -c "import joblib; from sklearn.datasets import load_breast_cancer; from sklearn.model_selection import train_test_split; from sklearn.metrics import roc_auc_score; model = joblib.load('models/model.joblib'); data = load_breast_cancer(); X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, test_size=0.2, random_state=42); proba = model.predict_proba(X_test)[:,-1]; roc_auc = roc_auc_score(y_test, proba); assert roc_auc > 0.75, f'ROC-AUC {roc_auc:.4f} < 0.75'"
```

- **name:** Log in to Container Registry

uses: docker/login-action@v3

with:

registry: \${{ env.REGISTRY }}

username: \${{ github.actor }}

password: \${{ secrets.GITHUB_TOKEN }}

- **name:** Build and push Docker image

uses: docker/build-push-action@v5

with:

context: .

push: true

tags: |

\${{ env.REGISTRY }}/\${{ env.IMAGE_NAME }}:latest

\${{ env.REGISTRY }}/\${{ env.IMAGE_NAME }}:\${{ github.sha }}

Опционально: деплой на виртуальную машину через SSH

- name: Deploy to server

uses: appleboy/ssh-action@v0.1.5

with:

host: \${{ secrets.DEPLOY_HOST }}

username: \${{ secrets.DEPLOY_USER }}

key: \${{ secrets.DEPLOY_KEY }}

script: |

docker pull \${{ env.REGISTRY }}/\${{ env.IMAGE_NAME }}:late

st

docker stop ml-app || true

docker rm ml-app || true

docker run -d --name ml-app -p 8000:8000 \${{ env.REGISTRY }}/\${{ env.IMAGE_NAME }}:latest

st

4.8 Настройка secrets

Для работы с GitHub Container Registry используется автоматический токен `GITHUB_TOKEN` (не требует дополнительной настройки). Для деплоя на внешний сервер необходимо добавить secrets: `DEPLOY_HOST`, `DEPLOY_USER`, `DEPLOY_KEY`.

4.9 Результат

При push в `main`:

1. Запускаются тесты (pytest).
2. Обучается модель, проверяется ROC-AUC > 0.75.
3. Собирается Docker-образ с FastAPI сервисом.

4. Образ пушится в `ghcr.io/username/ml-ci-cd-demo/ml-inference:latest`.

5. (Опционально) Обновляется сервер.

Если тесты падают или $\text{ROC-AUC} < 0.75$ – pipeline завершается ошибкой, образ не публикуется.

Индивидуальные задания

Каждый вариант должен создать репозиторий (локально или на GitHub) со своей ML-моделью и настроить CI/CD по аналогии с примером.

Условия:

- Модель может быть любой (из предыдущих работ).
- Порог качества (ROC-AUC, accuracy, F1, R^2) задаётся в соответствии с вариантом.
- Тесты должны включать минимум 3 юнит-теста (например, проверка размерности, проверка обработки пропусков, проверка типов).
- Docker-образ должен содержать сервис инференса (FastAPI) и автоматически загружать модель.
- Workflow должен срабатывать при push и pull request.

Вариант	Датасет (или модель)	Целевая метрика	Порог	Дополнительное требование
1	Breast Cancer (RandomForest)	ROC-AUC	0.95	—
2	Iris (LogisticRegression)	Accuracy	0.90	—
3	Diabetes (XGBoost)	ROC-AUC	0.75	добавить тест на обработку пропусков
4	Wine (RandomForest)	Accuracy	0.95	—

Вариант	Датасет (или модель)	Целевая метрика	Порог	Дополнительное требование
5	Telco Churn (LightGBM)	ROC-AUC	0.80	тест на проверку категориальных признаков
6	Credit Card Fraud (XGBoost)	PR-AUC	0.60	использовать PR-AUC вместо ROC-AUC
7	California Housing (RandomForestRegressor)	R ²	0.70	—
8	Titanic (GradientBoosting)	Accuracy	0.80	тест на размер входных данных
9	Spambase (LogisticRegression)	ROC-AUC	0.90	—
10	Adult Income (RandomForest)	F1 (macro)	0.65	добавить тест на корректность кодирования
11	MNIST (упрощённый, 10 классов)	Accuracy	0.85	модель MLPClassifier
12	Heart Disease (XGBoost)	ROC-AUC	0.85	—
13	Bike Sharing (RandomForestRegressor)	MAE	50	порог по MAE (меньше 50)
14	IMDB sentiment (LogisticRegression с TF-IDF)	Accuracy	0.85	тест на воспроизводимость (seed)

Вариант	Датасет (или модель)	Целевая метрика	Порог	Дополнительное требование
15	Своя модель из лаб. №10 (YOLO)	mAP50	0.60	сложный вариант; тестировать только валидацию метаданных (не сам инференс)

Примечания:

- Для варианта 15 (YOLO) CI/CD может проверять только корректность экспорта в ONNX и загрузку модели, без полного обучения (из-за длительности).

Пороги выбраны ориентировочно; студент может обосновать свой порог в отчёте.

Контрольные вопросы

1. В чём отличие CI от CD? Приведите пример для ML-проекта.
2. Почему важно проверять качество модели (ROC-AUC > порога) в CI/CD пайплайне?
3. Как можно в GitHub Actions заэкшировать зависимости (pip) для ускорения сборки?
4. Что такое GitHub Container Registry и как аутентифицироваться в нём через GITHUB_TOKEN?
5. Как запустить тесты только при изменении определённых файлов (например, app/)?
6. Как в workflow использовать матричную стратегию для тестирования на нескольких версиях Python?
7. Какие риски возникают при автоматическом деплое каждой успешной сборки в production?

8. Как можно реализовать ручной approval перед деплоем (production gate) в GitHub Actions?
9. Как в тестах подменить реальную модель на mock для ускорения?
10. По результатам вашего варианта: что произойдёт с workflow, если случайно закоммитить код, который снижает ROC-AUC ниже порога?

Требования к отчёту

Отчёт оформляется по ГОСТ 7.32-2017. Содержит:

1. Титульный лист.
2. Цель и задачи.
3. Теоретическое обоснование (CI/CD, GitHub Actions, тестирование ML).
4. Практическая часть:
 - Структура репозитория.
 - Ключевые файлы: код модели, тесты, Dockerfile, workflow YAML.
 - Скриншоты успешного запуска workflow в GitHub Actions (лог).
 - Скриншот опубликованного Docker-образа в реестре.
5. Результаты выполнения (прохождение тестов, проверка порога качества).
6. Анализ результатов (что было 最难, как решали проблемы с производительностью тестов).
7. Ответы на контрольные вопросы.
8. Вывод.

Список литературы.

Вариативная часть для оценки «отлично»:

- Добавить этап автоматического обновления сервера (например, через SSH или webhook).
 - Настроить уведомления в Slack/Telegram о результате пайплайна.
- Использовать `actions/cache` для кэширования виртуального окружения Python.

Критерии оценивания

Оценка	Критерии
Отлично	Работа выполнена полностью. Репозиторий содержит все необходимые файлы. Workflow настроен корректно и успешно проходит при выполнении условий (тесты, качество). Docker-образ публикуется в реестре. Реализована вариативная часть (деплой или уведомления). Защита: свободное понимание CI/CD концепций для ML.
Хорошо	Работа выполнена полностью, но отсутствует вариативная часть, или порог качества задан неявно (не проверяется в workflow), или Docker-образ не публикуется (только сборка). Защита: ответы на большинство вопросов.
Удовлетворительно	Workflow настроен, но некоторые этапы пропущены (нет линтера или нет тестов качества), или тесты не проходят, но студент не исправил. Отчёт неполный. Студент не различает CI и CD.
Неудовлетворительно	Работа не представлена, или workflow отсутствует, или код неработоспособен, или Docker-образ не собирается. Отчёт не соответствует требованиям.

Лабораторная работа №13.

A/B-тестирование двух версий модели

Цель работы: Освоить методику A/B-тестирования в контексте машинного обучения: сравнение двух версий модели (control – старая, treatment – новая) на реальном (симулированном) трафике. Научиться реализовывать рандомизированный сплит трафика на уровне микросервиса, собирать бизнес-метрики с помощью Prometheus и визуализировать результаты в Grafana для принятия решения о раскатке новой модели.

Задачи:

- Реализовать FastAPI-сервис, который содержит две загруженные модели (старая и новая).
- Добавить middleware (или эндпоинт), который случайным образом направляет запрос на одну из моделей с заданными долями трафика (например, 90% – старая, 10% – новая).
- Обеспечить уникальную идентификацию пользователя/сессии для консистентного сплита (например, по user_id, bucket по хешу).
- Собирать метрики в Prometheus:
 - количество запросов на каждую модель;
 - распределение предсказаний (гистограмма);
 - время ответа (latency);
 - (опционально) бизнес-метрику (конверсия, CTR, стоимость).
- Настроить дашборд в Grafana для визуализации ключевых метрик в реальном времени.
- Провести симуляцию трафика (например, с помощью locust или простого скрипта) и собрать данные за период.
- Проанализировать результаты: есть ли статистически значимое улучшение новой модели по целевой метрике.
- Принять решение: раскатывать новую модель на 100% или откатить.

Теоретическая часть

1. A/B-тестирование в ML

A/B-тест (сплит-тест) – эксперимент, в котором две группы пользователей (контрольная и тестовая) получают разные версии модели. Цель – определить, какая модель даёт лучшее значение бизнес-метрики (конверсия, удержание, доход). В отличие от offline-оценки по историческим данным, A/B-тест позволяет учесть поведение пользователей в реальных условиях (например, влияние предсказаний на действия).

Ключевые понятия:

- **Контрольная группа (Control)** – получает предсказания от старой модели (baseline).
- **Тестовая группа (Treatment)** – получает предсказания от новой модели.
- **Рандомизация** – равномерное распределение пользователей по группам (обычно на основе хеша `user_id`).
- **Статистическая значимость** – $p\text{-value} < 0.05$, чтобы исключить случайные различия.

2. Стратегии сплита трафика

- **Случайный детерминированный сплит** – по модулю хеша идентификатора пользователя (например, `hash(user_id) % 100 < 10` для 10% трафика). Это гарантирует, что один и тот же пользователь всегда попадает в одну группу (консистентность).
- **Взвешенный случайный** – каждому запросу назначается группа независимо (подходит для stateless-сценариев, но может нарушать пользовательский опыт).
- **Канареечный деплой (canary)** – сначала 1-5% трафика на новую модель, затем постепенное увеличение.

3. Метрики для A/B-теста

- **Технические метрики:** latency (p50, p95, p99), error rate, throughput.

- **Бизнес-метрики:** конверсия, CTR, средний чек, retention, стоимость.

- **ML-метрики:** средняя уверенность модели, калибровка.

Важно: Бизнес-метрика должна собираться из downstream-систем (например, совершил ли пользователь покупку после предсказания). В рамках лабораторной работы можно симулировать бизнес-исход на основе предсказания (например, если вероятность положительного класса > 0.8 , то конверсия = 1).

4. Prometheus и Grafana

- **Prometheus** – система мониторинга и сбора метрик с pull-моделью. FastAPI должен выставить эндпоинт `/metrics` в формате, понятном Prometheus (библиотека `prometheus_client`).

- **Grafana** – инструмент для визуализации. Подключается к Prometheus, позволяет строить дашборды с графиками, гистограммами, таблицами.

Типовые метрики в Prometheus:

python

```
from prometheus_client import Counter, Histogram, Gauge
```

```
requests_total = Counter('requests_total', 'Total requests', ['model_version'])
```

```
request_duration = Histogram('request_duration_seconds', 'Request latency',  
['model_version'])
```

```
prediction_value = Histogram('prediction_value', 'Predicted probability',  
['model_version'])
```

Пример практического выполнения

Сценарий: Сервис предсказания вероятности клика (CTR) для рекламного баннера. Старая модель – LogisticRegression, новая – XGBoost. Сплит: 90% трафика на старую, 10% на новую. Бизнес-метрика: клик (симулируется как `Bernoulli(prediction)`).

4.1 Структура проекта

text

ab_testing/

|— app/

| |— main.py

| |— models.py (загрузка двух моделей)

| |— metrics.py (Prometheus метрики)

|— docker-compose.yml (FastAPI + Prometheus + Grafana)

|— prometheus/

| |— prometheus.yml

|— grafana/

| |— provisioning/

|— simulate_traffic.py (скрипт для генерации запросов)

|— requirements.txt

4.2 Загрузка двух моделей (app/models.py)

python

```
import joblib
```

```
# Загружаем старую и новую модель (предварительно обученные)
```

```
old_model = joblib.load("old_model.joblib") # LogisticRegression
```

```
new_model = joblib.load("new_model.joblib") # XGBoost
```

```
def get_model(version: str):
```

```
    return old_model if version == "old" else new_model
```

4.3 FastAPI приложение с middleware для сплита (app/main.py)

python

```
import hashlib
```

```
import random
```

```
import time
```

```
from fastapi import FastAPI, Request
```

```

from pydantic import BaseModel

import numpy as np

from prometheus_client import generate_latest, CONTENT_TYPE_LATES
T

from .metrics import REQUESTS, LATENCY, PREDICTION_VALUE, BU
SINESS_CONVERSION

from .models import old_model, new_model


app = FastAPI()


# Параметры A/B-теста
TREATMENT_PERCENT = 10 # 10% на новую модель
SPLIT_BY = "user_id" # ключ для детерминированного сплита


def get_model_version(user_id: str) -> str:
    """Детерминированное распределение по хешу user_id"""
    hash_val = int(hashlib.md5(user_id.encode()).hexdigest()[:8], 16)
    return "new" if (hash_val % 100) < TREATMENT_PERCENT else "old"


class PredictRequest(BaseModel):
    user_id: str
    features: list[float]


class PredictResponse(BaseModel):
    model_version: str
    prediction: float
    click_probability: float # для симуляции бизнес-метрики


@app.middleware("http")
async def add_metrics(request: Request, call_next):

```

```

start = time.perf_counter()
response = await call_next(request)
duration = time.perf_counter() - start
# Запись длительности (можно по эндпоинту)
return response

@app.post("/predict")
async def predict(req: PredictRequest):
    # Определение версии модели
    version = get_model_version(req.user_id)
    model = old_model if version == "old" else new_model

    # Предсказание
    features = np.array(req.features).reshape(1, -1)
    proba = model.predict_proba(features)[0][1]

    # Обновление Prometheus метрик
    REQUESTS.labels(model_version=version).inc()
    LATENCY.labels(model_version=version).observe(0.01) # реальная задержка

    PREDICTION_VALUE.labels(model_version=version).observe(proba)

    # Симуляция бизнес-исхода: клик происходит с вероятностью proba
    clicked = 1 if np.random.random() < proba else 0
    BUSINESS_CONVERSION.labels(model_version=version, outcome="click").inc(clicked)
    BUSINESS_CONVERSION.labels(model_version=version, outcome="no_click").inc(1 - clicked)

    return PredictResponse(

```

```
    model_version=version,  
    prediction=proba,  
    click_probability=clicked  
)
```

```
@app.get("/metrics")  
def metrics():  
    return generate_latest()
```

4.4 Prometheus метрики (app/metrics.py)

python

```
from prometheus_client import Counter, Histogram
```

```
REQUESTS = Counter('requests_total', 'Total requests', ['model_version'])
```

```
LATENCY = Histogram('request_duration_seconds', 'Request latency', ['model_version'])
```

```
PREDICTION_VALUE = Histogram('prediction_value', 'Predicted probability', ['model_version'])
```

```
BUSINESS_CONVERSION = Counter('business_conversion_total', 'Business conversion', ['model_version', 'outcome'])
```

4.5 Docker Compose (docker-compose.yml)

yaml

```
version: '3.8'
```

```
services:
```

```
  fastapi:
```

```
    build: .
```

```
    ports:
```

```
      - "8000:8000"
```

```
    environment:
```

```
      - TREATMENT_PERCENT=10
```

networks:

- ab_net

prometheus:

image: prom/prometheus:latest

volumes:

- ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml

ports:

- "9090:9090"

networks:

- ab_net

grafana:

image: grafana/grafana:latest

ports:

- "3000:3000"

environment:

- GF_SECURITY_ADMIN_PASSWORD=admin

volumes:

- ./grafana/provisioning:/etc/grafana/provisioning

networks:

- ab_net

networks:

ab_net:

4.6 Prometheus конфигурация (prometheus/prometheus.yml)

yaml

global:

scrape_interval: 5s

scrape_configs:

- job_name: 'fastapi'

static_configs:

- targets: ['fastapi:8000']

4.7 Симуляция трафика (simulate_traffic.py)

python

import requests

import random

import time

import numpy as np

USER_IDS = [f'user_{i}' for i in range(1000)]

def generate_features():

return [random.random() for _ in range(10)]

for _ in range(5000):

user_id = random.choice(USER_IDS)

features = generate_features()

resp = requests.post("http://localhost:8000/predict", json={

"user_id": user_id,

"features": features

})

time.sleep(0.05) # 20 запросов в секунду

4.8 Анализ результатов в Grafana

Дашборд включает:

- Количество запросов на старую и новую модель (bar chart).
- Гистограммы предсказанных вероятностей.
- Конверсию (клики / показы) для каждой модели.
- Latency p50, p95.

Пример вывода через сутки симуляции:

- Старая модель (90% трафика): 4500 запросов, конверсия 3.2%.
- Новая модель (10% трафика): 500 запросов, конверсия 3.9%.
- Статистический тест (z-test) показывает $p\text{-value} = 0.03 \rightarrow$ различие

значимо.

Решение: раскатывать новую модель на 100%.

Индивидуальные задания

Каждый вариант выбирает домен и две модели для сравнения (можно взять модели из предыдущих лабораторных). Необходимо:

1. Реализовать FastAPI с A/B сплитом (доли трафика указаны).
2. Собирать метрики в Prometheus (минимум: количество запросов, латентность, бизнес-метрика).
3. Настроить Grafana дашборд.
4. Симулировать трафик (не менее 5000 запросов).
5. Провести анализ и принять решение.

Вариант	Домен	Старая модель	Новая модель	Сплит (старая/новая)	Бизнес-метрика
1	Кредитный скоринг	LogisticRegression	XGBoost	80/20	Одобрение заявки (порог 0.7)
2	Отток клиентов	DecisionTree	RandomForest	90/10	Сохранение клиента (если proba > 0.5, to offer)

Вариант	Домен	Старая модель	Новая модель	Сплит (старая/новая)	Бизнес-метрика
3	Рекомендации товаров	MF (матричная факторизация)	LightGBM	70/30	CTR (клик)
4	Мошенничество	LogisticRegression	IsolationForest + LR	95/5	Обнаружение мошенничества (TPR)
5	Прогноз цены акций	ARIMA	LSTM	85/15	MAE (чем меньше, тем лучше)
6	Диагностика заболевания	KNN	RandomForest	80/20	Чувствительность (TPR)
7	Чат-бот (класс интента)	TF-IDF + LR	DistilBERT fine-tuned	90/10	Accuracy
8	Распознавание изображений	ResNet18 (small)	ResNet50	75/25	Accuracy, latency
9	Автоматическое пополнение запасов	LinearRegression	XGBoostRegressor	80/20	MAPE

Вариант	Домен	Старая модель	Новая модель	Сплит (старая/новая)	Бизнес-метрика
10	Определение тональности отзыва	Naive Bayes	BERT (small)	85/15	F1-score
11	Оценка риска ДТП	LogisticRegression	RandomForest	90/10	Уровень аварий (снижение)
12	Прогноз урожайности	LinearRegression	GradientBoosting	80/20	R ²
13	Сегментация клиентов (cluster assignment)	KMeans	DBSCAN	70/30	Внутрикластерная сумма квадратов
14	Распознавание эмоций по голосу	SVM	CNN	85/15	Accuracy
15	Определение спама в email	Naive Bayes	XGBoost	95/5	FPR (меньше – лучше)

Контрольные вопросы

1. Почему для A/B-теста важно использовать детерминированный сплит на основе user_id, а не случайный по запросу?

2. Какие статистические критерии применяются для сравнения двух групп (бинарная метрика)?
3. Что такое p-value и как его интерпретировать?
4. Как выбрать минимальный размер выборки для A/B-теста?
5. В чём опасность преждевременной остановки эксперимента при достижении $p\text{-value} < 0.05$?
6. Как в Prometheus собирать гистограммы латентности (p50, p95)?
7. Как в Grafana настроить алерт при падении конверсии в тестовой группе ниже контрольной?
8. Как можно реализовать канареечный деплой с автоматическим откатом при аномалиях?
9. Почему важно собирать бизнес-метрику, а не только ML-метрики (например, ассигасу)?
10. По результатам вашего варианта: какое решение вы приняли и почему?

Требования к отчёту

Отчёт оформляется по ГОСТ 7.32-2017. Содержит:

1. Титульный лист.
2. Цель и задачи.
3. Теоретическое обоснование (A/B-тестирование, сплит, метрики).
4. Практическая часть: код сервиса, конфигурации Prometheus/Grafana, скрипт симуляции.
5. Результаты выполнения: скриншоты дашборда Grafana, таблицы собранных метрик.
6. Анализ результатов (статистический тест, вывод о значимости).
7. Ответы на контрольные вопросы.
8. Вывод.
9. Список литературы.

Вариативная часть для оценки «отлично»:

- Реализовать автоматический подсчёт p-value и вывод решения в лог.
- Добавить эндпоинт `/switch` для динамического изменения процента трафика.

Настроить алерт в Grafana на превышение порога ошибок новой модели.

Критерии оценивания

Оценка	Критерии
Отлично	Работа выполнена полностью. FastAPI корректно реализует сплит трафика (детерминированный). Prometheus собирает метрики, Grafana визуализирует. Проведена симуляция, выполнен статистический анализ, принято обоснованное решение. Реализована вариативная часть. Защита: глубокое понимание A/B-тестирования.
Хорошо	Работа выполнена полностью, но отсутствует вариативная часть, или сплит недетерминированный (случайный), или нет бизнес-метрики. Защита: ответы на большинство вопросов.
Удовлетворительно	Метрики собираются, но дашборд не настроен, или анализ статистической значимости не выполнен. Студент путает A/B-тест с просто сравнением offline метрик.
Неудовлетворительно	Работа не представлена, или FastAPI не работает, или Prometheus/Grafana не настроены. Отчёт не соответствует требованиям.

Лабораторная работа №14.

Сквозной проект: от Data Lake до A/B-теста

Цель работы: Интегрировать все ранее изученные компоненты MLOps-конвейера в единый автоматизированный сквозной проект. Реализовать полный цикл обработки данных и эксплуатации модели машинного обучения: от извлечения данных из Data Lake (инжиниринг данных) до A/B-тестирования двух версий модели с мониторингом дрейфа. Оркестрация этапов выполняется с помощью Apache Airflow.

Задачи:

- Развернуть инфраструктуру, включающую:
 - **Data Lake (MinIO)** – хранение сырых данных (Parquet/CSV);
 - **PostgreSQL** – оперативная витрина и метаданные;
 - **Feast** – feature store (признаки);
 - **Redis** – онлайн-хранилище признаков;
 - **MLflow** – отслеживание экспериментов и реестр моделей;
 - **Apache Airflow** – оркестрация DAG;
 - **FastAPI** – сервис инференса с A/B-тестированием;
 - **Prometheus + Grafana** – мониторинг и визуализация.
- Реализовать DAG в Airflow, последовательно выполняющий:
 - **ETL** – извлечение данных из Data Lake, очистка, агрегация, загрузка в витрину.
 - **Feast** – регистрация сущностей, источников, feature view; материализация признаков в Redis.
 - **Обучение модели** – запуск скрипта с логированием параметров и метрик в MLflow.
 - **Регистрация модели** – выбор лучшей модели по метрике и регистрация в MLflow Model Registry.
 - **Канареечный деплой** – обновление A/B-сервиса: новая модель получает, например, 10% трафика.

- **Мониторинг дрейфа** – проверка дрейфа данных (Evidently AI) между периодами, отправка алерта при обнаружении.
- Обеспечить возможность A/B-тестирования (лабораторная №13) с визуализацией метрик в Grafana.
- Продемонстрировать работоспособность конвейера: запустить DAG, проверить логи, показать результаты в MLflow, Feast, Grafana.

Теоретическая часть

1. Сквозной MLOps-конвейер

Производственная ML-система требует автоматизации многих этапов, каждый из которых может быть источником ошибок и рассинхронизации. Сквозной конвейер (end-to-end pipeline) объединяет:

- **Data Lake** – хранилище сырых данных (неструктурированные, полуструктурированные).
- **ETL (ELT)** – процессы извлечения, трансформации, загрузки в структурированную витрину.
- **Feature Store (Feast)** – унифицированное управление признаками для обучения и инференса.
- **Эксперименты (MLflow)** – логирование параметров, метрик, артефактов.
- **Реестр моделей (MLflow Model Registry)** – версионирование и этапы (Staging, Production).
- **Оркестрация (Airflow)** – запуск задач по расписанию или событию с обработкой ошибок.
- **Сервинг (FastAPI + Docker)** – low-latency инференс с A/B-тестированием.
- **Мониторинг** – дрейф данных, концептуальный дрейф, качество предсказаний.

- **Визуализация (Grafana)** – дашборды для бизнес-метрик и технических метрик.

2. Роль Apache Airflow

Airflow позволяет определять DAG (Directed Acyclic Graph) – направленный ациклический граф зависимостей задач. Преимущества:

- Планирование по времени (например, ежедневно).
- Повторные запуски, обработка сбоев, уведомления.
- Интеграция с Docker, Python, Bash, Spark и другими инструментами.
- Прозрачность логов и истории выполнения.

3. Интеграция Feast с Airflow

Feast предоставляет CLI команды (`feast apply`, `feast materialize`), которые можно вызывать из Airflow через `BashOperator` или использовать Python API внутри `PythonOperator`. Материализация признаков должна выполняться за временной интервал, соответствующий периоду обучения.

4. MLflow в конвейере

В DAG обучение модели запускается как отдельный скрипт, который логирует метрики и сохраняет модель. После обучения задача регистрации (например, `PythonOperator`) выбирает run с наилучшей метрикой (ROC-AUC) и регистрирует модель как `Staging`. Канареечный деплой обновляет конфигурацию сервиса, указывая новую модель как кандидата с малым процентом трафика.

5. Мониторинг дрейфа

Сравниваются распределения признаков за два периода: `reference` (например, данные, на которых обучалась модель) и `current` (свежие данные за последний день). Evidently AI вычисляет PSI, Jensen-Shannon, p-value и определяет, какие признаки изменились статистически значимо. При превышении порога отправляется алерт в Telegram/Slack, а DAG может быть помечен как успешный, но с предупреждением.

Пример практического выполнения

Полный код конвейера из-за объёма не приводится, но описываются критические части.

4.1 Архитектурная схема

text

[Data Lake (MinIO)] → (ETL: Spark/Pandas) → [PostgreSQL витрина]



[Feast] ← (регистрация feature view) ← исторические данные



[Redis (online store)] ← материализация



[Airflow DAG] → (train.py) → [MLflow Tracking] → [Model Registry]



[Canary Deploy]



[FastAPI A/B service] ← 10% трафика новая модель, 90% старая



[Prometheus] → [Grafana dashboard]



[Мониторинг дрейфа (Evidently)] → Telegram alert

4.2 Пример DAG (`mlops_pipeline.py`)

python

```
from airflow import DAG
```

```
from airflow.operators.bash import BashOperator
```

```
from airflow.operators.python import PythonOperator
```

```
from datetime import datetime, timedelta
```

```
default_args = {
```

```
    'owner': 'mlops',
```

```
    'depends_on_past': False,
```

```
'start_date': datetime(2024, 1, 1),  
'retries': 1,  
'retry_delay': timedelta(minutes=5),  
}
```

```
dag = DAG(  
    'mlops_end_to_end',  
    default_args=default_args,  
    description='Full MLOps pipeline: Data Lake → A/B test',  
    schedule_interval='@daily',  
    catchup=False,  
)
```

```
etl = BashOperator(  
    task_id='etl_from_datalake',  
    bash_command='python /scripts/etl.py --date {{ ds }}',  
    dag=dag,  
)
```

```
feast_apply = BashOperator(  
    task_id='feast_apply',  
    bash_command='cd /feature_repo && feast apply',  
    dag=dag,  
)
```

```
feast_materialize = BashOperator(  
    task_id='feast_materialize',  
    bash_command='cd /feature_repo && feast materialize --end-date {{ ds }}',  
    dag=dag,
```

)

```
train = BashOperator(  
    task_id='train_model',  
    bash_command='python /scripts/train.py --run-date {{ ds }}',  
    dag=dag,  
)
```

```
register_best = PythonOperator(  
    task_id='register_best_model',  
    python_callable=register_best_model_func,  
    dag=dag,  
)
```

```
canary_deploy = BashOperator(  
    task_id='canary_deploy',  
    bash_command='python /scripts/deploy_canary.py --model-version {{ ti.x  
com_pull("register_best_model") }} --traffic 10',  
    dag=dag,  
)
```

```
drift_monitor = BashOperator(  
    task_id='monitor_drift',  
    bash_command='python /scripts/drift_monitor.py --ref-date {{ prev_ds }}  
--curr-date {{ ds }}',  
    dag=dag,  
)
```

```
etl >> feast_apply >> feast_materialize >> train >> register_best >> canary_  
deploy >> drift_monitor
```

4.3 Скрипт обучения с MLflow

```
python
```

```
import mlflow
```

```
from feast import FeatureStore
```

```
import pandas as pd
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import roc_auc_score
```

```
mlflow.set_tracking_uri("http://mlflow-server:5000")
```

```
mlflow.set_experiment("customer_churn")
```

```
fs = FeatureStore(repo_path="/feature_repo")
```

```
# Получение исторических признаков для обучения
```

```
entity_df = pd.read_parquet(f"/data/features_{run_date}.parquet")
```

```
historical_features = fs.get_historical_features(  
    entity_df=entity_df,  
    features=["user_features:tenure", "user_features:monthly_charges", "user_  
features:payment_method"]  
)
```

```
X = historical_features.drop(columns=["churn"])
```

```
y = historical_features["churn"]
```

```
with mlflow.start_run() as run:
```

```
    model = RandomForestClassifier(n_estimators=100)
```

```
    model.fit(X, y)
```

```
    auc = roc_auc_score(y, model.predict_proba(X)[:,1])
```

```
    mlflow.log_metric("roc_auc", auc)
```

```
    mlflow.sklearn.log_model(model, "model")
```

```
# Сохраняем run_id в XCom для следующей задачи
```

```
print(run.info.run_id)
```

4.4 Регистрация модели

```
python
```

```
def register_best_model_func(**context):  
    from mlflow.tracking import MlflowClient  
    client = MlflowClient()  
    # В реальности нужно найти лучший run за период, упрощённо:  
    run_id = context["ti"].xcom_pull(task_ids="train_model")  
    result = client.create_registered_model("churn_model")  
    client.create_model_version(  
        name="churn_model",  
        source=f'mlruns/0/{run_id}/artifacts/model',  
        run_id=run_id,  
        description="Best model from daily training"  
    )  
    # Передаём версию дальше  
    context["ti"].xcom_push(key="model_version", value=1)
```

4.5 Канареечный деплой (deploy_canary.py)

Скрипт обновляет конфигурацию А/В-сервиса (например, через Redis или API самого сервиса). Пример:

```
python
```

```
import redis
```

```
import sys
```

```
model_version = sys.argv[1]
```

```
traffic_percent = int(sys.argv[3]) # 10
```

```
r = redis.Redis(host='redis-service', port=6379)
```

```
r.set("ab_config", f'{{"old_model": "churn_model:v1", "new_model": "churn_model:{model_version}", "new_traffic_percent": {traffic_percent}}})
```

```
print("Canary configuration updated")
```

4.6 Мониторинг дрейфа (Evidently)

```
python
```

```
import pandas as pd
```

```
from evidently.report import Report
```

```
from evidently.metric_preset import DataDriftPreset
```

```
import requests
```

```
ref_data = pd.read_parquet(f"/data/daily_features/{prev_date}.parquet")
```

```
curr_data = pd.read_parquet(f"/data/daily_features/{curr_date}.parquet")
```

```
report = Report(metrics=[DataDriftPreset()])
```

```
report.run(reference_data=ref_data, current_data=curr_data)
```

```
drift_result = report.as_dict()
```

```
drifted_features = [col for col, res in drift_result['metrics'][0]['result']['drift_by_columns'].items() if res['drift_detected']]
```

```
if drifted_features:
```

```
    message = f"Дрейф обнаружен в признаках: {drifted_features}"
```

```
    requests.post("https://api.telegram.org/bot<TOKEN>/sendMessage", json={
        "chat_id": "<ID>", "text": message})
```

```
else:
```

```
    print("Дрейфа нет")
```

Индивидуальные задания

Интеграция A/B-тестирования.

Сервис FastAPI аналогичен лабораторной работе №13, но динамически загружает модели из MLflow Model Registry или по указанному в конфиге пути. При старте сервис подписывается на обновления конфигурации (например, перечитывает каждые 30 секунд).

Контрольные вопросы

1. Почему для оркестрации сквозного конвейера выбран Airflow, а не, например, cron или Jenkins?
2. Как в Airflow обрабатывать зависимости между задачами (завершение ETL перед Feast)?
3. Какие метрики качества модели должны быть залогированы в MLflow для последующего сравнения?
4. Как обеспечить, чтобы канареечный деплой не нарушал работу сервиса (zero-downtime)?
5. Что произойдёт, если мониторинг дрейфа обнаружит сильный дрейф? Какие действия должен предпринять DAG?
6. Как версионировать признаки в Feast?
7. Как в одном DAG объединить обучение модели и A/B-тест, который идёт параллельно?
8. Как часто следует переобучать модель? Как это связать с расписанием Airflow?
9. Какие метрики в Prometheus/Grafana наиболее важны для мониторинга конвейера в целом?
10. В чём отличие канареечного деплоя от A/B-теста? Как они сочетаются в данном проекте?

Требования к отчёту

Отчёт (40–60 страниц) оформляется по ГОСТ 7.32-2017 и должен содержать:

1. Титульный лист, задание, реферат.
2. Описание архитектуры всех компонентов (схема).
3. Листинги всех скриптов (ETL, Feast, train, deploy, drift, DAG, FastAPI).
4. Скриншоты:
 - Airflow UI (граф DAG, логи успешных тасок).

- MLflow UI (эксперименты, метрики, зарегистрированная модель).
 - Feast CLI (применённые объекты, материализация).
 - Grafana дашборд (метрики A/B-теста).
 - Пример алерта в Telegram.
5. Демонстрация A/B-теста: доля трафика на новую модель, бизнес-метрики.

Выводы о работе всего конвейера, возможные улучшения.

Критерии оценивания

Оценка	Критерии
Отлично	Развёрнута полная инфраструктура (Docker Compose всех сервисов). DAG работает без ошибок, все этапы проходят успешно. Реализованы A/B-тест и мониторинг дрейфа. Документация подробна, есть скриншоты и анализ.
Хорошо	Отсутствует один из второстепенных компонентов (например, мониторинг дрейфа), но основной пайплайн (ETL → Feast → обучение → деплой) работает.
Удовлетворительно	DAG запускается, но некоторые задачи завершаются ошибками, либо A/B-тест не работает.
Неудовлетворительно	Проект не представлен или неработоспособен.

СПСНОК ЛИТЕРАТУРЫ

1. Радченко, М. Г. 1С:Предприятие 8.3. Практическое пособие разработчика. — М.: 1С-Паблишинг, 2022. — 964 с. — ISBN 978-5-9677-2956-7.
2. Филиппов, Е. В. Язык запросов 1С:Предприятие 8. Подробное руководство. — СПб.: БХВ-Петербург, 2021. — 448 с. — ISBN 978-5-9775-6678-2.
3. Бойко, В. В., Савинков, В. М. Администрирование 1С:Предприятие 8. Кластеры, серверы, мониторинг. — М.: 1С-Паблишинг, 2023. — 512 с. — ISBN 978-5-9677-3124-9.
4. Краковский, Д. Ю. Управляемые формы в 1С:Предприятие 8. Разработка интерфейсов. — М.: 1С-Паблишинг, 2022. — 352 с. — ISBN 978-5-9677-3001-6.
5. Петлин, А. В. Интеграция 1С с внешними системами: HTTP-сервисы, REST, SOAP. — М.: 1С-Паблишинг, 2021. — 288 с. — ISBN 978-5-9677-2912-5.
6. Моргунов, Е. П. СУБД PostgreSQL для администраторов 1С. — М.: ДМК Пресс, 2022. — 304 с. — ISBN 978-5-9706-0987-2.
7. Бармин, Д. С. Паттерны разработки в 1С: Предприятие 8. От процедур к моделям данных. — М.: 1С-Паблишинг, 2023. — 420 с. — ISBN 978-5-9677-3156-0.
8. Кардашевский, А. Н. Транзакции, блокировки и оптимизация в 1С. — СПб.: БХВ-Петербург, 2022. — 272 с. — ISBN 978-5-9775-6912-3.
9. Виноградов, С. В. Docker и контейнеризация для разработчиков 1С. — М.: 1С-Паблишинг, 2022. — 192 с. — ISBN 978-5-9677-3088-4.
10. Рыжков, П. С. RabbitMQ в микросервисных интеграциях. — М.: ДМК Пресс, 2021. — 336 с. — ISBN 978-5-9706-0876-9.
11. Гришин, А. Ю. Prometheus + Grafana: мониторинг и наблюдение систем. — М.: ДМК Пресс, 2022. — 288 с. — ISBN 978-5-9706-1034-2.

12. Ньюмен, С. Создание микросервисов. Проектирование систем реального времени. — СПб.: Питер, 2022. — 528 с. — ISBN 978-5-4461-1895-3. (Главы по CQRS, Event Sourcing, Saga применимы к архитектуре ERP).