Лабораторная работа №3-4, Часть 3: Интеграция с MLflow для трекинга экспериментов

Цель работы: Освоить интеграцию процесса тонкой настройки моделей с платформой MLflow для комплексного трекинга экспериментов. Научиться автоматически логировать гиперпараметры, метрики, артефакты и модели в ходе обучения.

Стек технологий:

- **ОС:** Ubuntu 24.04 LTS
- **Окружение:** Conda (`mlops-lab`)
- **Библиотеки:** `mlflow`, `transformers`, `datasets`, `torch`
- **Платформа:** MLflow Tracking Server
- **Интеграция:** MLflow + Hugging Face Transformers

Теоретическая часть

1. Интеграция MLflow с Transformers MLflow предоставляет нативные интеграции с популярными ML-фреймворками, включая Hugging Face Transformers. Ключевые возможности:

- **Автоматическое логирование:** Автологирование параметров, метрик и артефактов
- **Модельный регистр:** Версионирование и управление моделями
- **Воспроизводимость:** Фиксация всех компонентов эксперимента

2. Компоненты трекинга для NLP

- **Параметры:** learning rate, batch size, архитектура модели
- **Метрики:** accuracy, F1-score, perplexity, loss
- **Артефакты:** модель, токенизатор, графики обучения
- **Тэги:** задача, датасет, версия модели

3. Стратегии логирования

- **Ручное логирование:** Полный контроль над процессом
- **Автологирование:** Автоматическая фиксация метрик
- **Колбэки:** Интеграция через системные хуки

Задание на практическую реализацию

Этап 1: Подготовка среды и конфигурация

1. **Активация окружения и проверка зависимостей:**

```
conda activate mlops-lab
cd text-classification-project
pip install mlflow
```

2. **Запуск MLflow Tracking Server:**

```
mlflow server --backend-store-uri sqlite:///mlflow.db --default-artifact-
root ./mlruns --host 0.0.0.0 --port 5000
```

3. **Создание скрипта для интегрированного обучения:**

```
touch mlflow_integration.py
```

Этап 2: Модификация скрипта обучения с интеграцией MLflow

```
import mlflow
import mlflow.transformers
from datasets import load_dataset
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    TrainingArguments,
    Trainer,
    DataCollatorWithPadding
)
import numpy as np
from sklearn.metrics import accuracy_score, f1_score
import torch
import os

# Настройка MLflow
mlflow.set_tracking_uri("http://localhost:5000")
mlflow.set_experiment("Emotion-Classification-FineTuning")

def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    predictions = np.argmax(predictions, axis=1)

    acc = accuracy_score(labels, predictions)
    f1 = f1_score(labels, predictions, average="weighted")

    return {"accuracy": acc, "f1_score": f1}

def tokenize_function(examples):
    tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
    return tokenizer(
        examples["text"],
        truncation=True,
        padding=True,
        max_length=128
```

```
    )

# Начало эксперимента MLflow
with mlflow.start_run():
    # Загрузка и подготовка данных
    dataset = load_dataset("emotion")
    tokenized_datasets = dataset.map(tokenize_function, batched=True)
    tokenized_datasets = tokenized_datasets.rename_column("label", "labels")
    tokenized_datasets.set_format("torch", columns=["input_ids", "attention_mask",
"labels"])

    # Параметры модели и обучения
    model_params = {
        "model_name": "distilbert-base-uncased",
        "num_labels": 6,
        "learning_rate": 2e-5,
        "batch_size": 16,
        "num_epochs": 3,
        "weight_decay": 0.01
    }

    # Логирование параметров
    mlflow.log_params(model_params)

    # Загрузка модели
    model = AutoModelForSequenceClassification.from_pretrained(
        model_params["model_name"],
        num_labels=model_params["num_labels"],
        id2label={0: 'sadness', 1: 'joy', 2: 'love', 3: 'anger', 4: 'fear', 5:
'surprise'},
        label2id={'sadness': 0, 'joy': 1, 'love': 2, 'anger': 3, 'fear': 4,
'surprise': 5}
    )

    # Настройка обучения
    training_args = TrainingArguments(
        output_dir="./results",
        learning_rate=model_params["learning_rate"],
        per_device_train_batch_size=model_params["batch_size"],
        per_device_eval_batch_size=model_params["batch_size"],
        num_train_epochs=model_params["num_epochs"],
        weight_decay=model_params["weight_decay"],
        evaluation_strategy="epoch",
        save_strategy="epoch",
        load_best_model_at_end=True,
        metric_for_best_model="f1_score",
        logging_dir="./logs",
        logging_steps=100,
        report_to="none"
    )

    data_collator =
DataCollatorWithPadding(tokenizer=AutoTokenizer.from_pretrained(model_params["mode
l_name"]))
```

```
    # Создание тренера
    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=tokenized_datasets["train"],
        eval_dataset=tokenized_datasets["validation"],
        tokenizer=AutoTokenizer.from_pretrained(model_params["model_name"]),
        data_collator=data_collator,
        compute_metrics=compute_metrics,
    )

    # Обучение с логированием метрик
    print("Начало обучения с трекингом в MLflow...")
    train_result = trainer.train()

    # Логирование метрик обучения
    mlflow.log_metrics({
        "train_loss": train_result.metrics["train_loss"],
        "eval_loss": train_result.metrics["eval_loss"],
        "eval_accuracy": train_result.metrics["eval_accuracy"],
        "eval_f1_score": train_result.metrics["eval_f1_score"]
    })

    # Оценка на тестовых данных
    test_results = trainer.evaluate(tokenized_datasets["test"])
    mlflow.log_metrics({
        "test_accuracy": test_results["eval_accuracy"],
        "test_f1_score": test_results["eval_f1_score"]
    })

    # Сохранение и логирование модели
    model_path = "./emotion-classifier-mlflow"
    trainer.save_model(model_path)

    # Логирование модели в MLflow
    mlflow.transformers.log_model(
        transformers_model={
            "model": model,
            "tokenizer": AutoTokenizer.from_pretrained(model_params["model_name"])
        },
        artifact_path="emotion-classifier",
        registered_model_name="distilbert-emotion-classifier"
    )

    # Логирование дополнительных артефактов
    with open("training_summary.txt", "w") as f:
        f.write(f"Training completed successfully!\n")
        f.write(f"Final training loss:
{train_result.metrics['train_loss']:.4f}\n")
        f.write(f"Validation accuracy:
{train_result.metrics['eval_accuracy']:.4f}\n")
        f.write(f"Test accuracy: {test_results['eval_accuracy']:.4f}\n")
```

```
    mlflow.log_artifact("training_summary.txt")

    print("Эксперимент успешно завершен и записан в MLflow!")
```

Этап 3: Запуск и мониторинг эксперимента

1. **Запуск скрипта:**

```
python mlflow_integration.py
```

2. **Мониторинг в MLflow UI:**
   - Откройте `http://localhost:5000` в браузере
   - Найдите эксперимент "Emotion-Classification-FineTuning"
   - Изучите записанные параметры и метрики
   - Проверьте залогированную модель в разделе Artifacts

Этап 4: Дополнительные эксперименты

1. **Создание скрипта для сравнения гиперпараметров:**

```
touch hyperparameter_tuning.py
```

2. **Код для сравнения разных конфигураций:**

```
import mlflow
from mlflow_integration import train_model

# Эксперимент с разными learning rates
learning_rates = [1e-5, 2e-5, 5e-5]

for lr in learning_rates:
    with mlflow.start_run(nested=True):
        mlflow.log_param("learning_rate", lr)
        results = train_model(learning_rate=lr)
        mlflow.log_metrics(results)

print("Эксперимент по подбору learning rate завершен!")
```

Этап 5: Анализ результатов

1. **Создание скрипта для анализа:**

```
touch analyze_results.py
```

2. **Код для анализа экспериментов:**

```
import mlflow
from mlflow.tracking import MlflowClient

client = MlflowClient()

# Получение всех запусков эксперимента
experiment = client.get_experiment_by_name("Emotion-Classification-
FineTuning")
runs = client.search_runs(experiment.experiment_id)

print("Результаты экспериментов:")
for run in runs:
    print(f"Run ID: {run.info.run_id}")
    print(f"Parameters: {run.data.params}")
    print(f"Metrics: {run.data.metrics}")
    print("-" * 50)

# Нахождение лучшего запуска
best_run = min(runs, key=lambda x: x.data.metrics.get('eval_loss',
float('inf')))
print(f"Лучший запуск: {best_run.info.run_id}")
print(f"Лучшие метрики: {best_run.data.metrics}")
```

Требования к оформлению и отчету

Критерии оценки для Части 3:

- **Удовлетворительно:** Успешно выполнены Этапы 1-2 (настройка MLflow, модификация скрипта). Эксперимент запускается и базовые параметры записываются в MLflow.
- **Хорошо:** Дополнительно успешно выполнен Этап 3 (полное обучение с логированием всех метрик, модель зарегистрирована в MLflow). В UI отображаются все артефакты.
- **Отлично:** Все задания выполнены в полном объеме. Проведены дополнительные эксперименты (Этап 4) и выполнен анализ результатов (Этап 5). Сравнены разные конфигурации гиперпараметров.

Рекомендуемая литература

1. **MLflow Documentation:** https://mlflow.org/docs/latest/index.html
2. **MLflow Transformers Integration:** https://mlflow.org/docs/latest/python_api/mlflow.transformers.html
3. **Hugging Face Transformers Training:** https://huggingface.co/docs/transformers/training
4. **MLflow Tracking API:** https://mlflow.org/docs/latest/tracking.html
5. **Experiment Tracking Best Practices:** https://mlflow.org/docs/latest/tracking.html#organizing-runs-in-experiments