Домашна Задача 3

Подготовка на податоците (Offline/Online CSV)

1. Податоци од **Diabetes Health Indicators Dataset**, во CSV формат: diabetes_binary_health_indicators_BRFSS2015.csv.
2. Податоците ги поделив на:
    - **offline.csv** - 80% од податоците (за тренирање и евалуација)
    - **online.csv** - 20% од податоците (за streaming и online предвидување)

[3]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv("data/diabetes_binary_health_indicators_BRFSS2015.csv")

offline_df, online_df = train_test_split(
    df,
    test_size=0.2,
    random_state=42
)

offline_df.to_csv("offline.csv", index=False)
online_df.to_csv("online.csv", index=False)
```

Offline фаза (offline.ipynb)

1. **Вчитување и трансформација на податоците**
    - Податоците од offline.csv се вчитуваат во Spark DataFrame.
    - Сите feature колони се комбинираат во вектор со VectorAssembler.
    - Скалирање на карактеристиките со StandardScaler за подобра конвергенција на ML моделите.
2. **Обучување на модели за класификација**

Тренирав 3 модели со различни хиперпараметри и 5-fold cross-validation:

- **Logistic Regression (LR)**
- **Random Forest (RF)**
- **Decision Tree (DT)**

```
pipelines = [
    (
        "lr",
        Pipeline(stages=[assembler, scaler, lr]),
        ParamGridBuilder()
            .addGrid(lr.regParam, [0.01, 0.1])
            .addGrid(lr.elasticNetParam, [0.0, 0.5])
            .build()
    ),
    (
        "rf",
        Pipeline(stages=[assembler, scaler, rf]),
        ParamGridBuilder()
            .addGrid(rf.numTrees, [20, 50])
            .addGrid(rf.maxDepth, [5, 10])
            .build()
    ),
    (
        "dt",
        Pipeline(stages=[assembler, scaler, dt]),
        ParamGridBuilder()
            .addGrid(dt.maxDepth, [5, 10, 20])
            .addGrid(dt.minInstancesPerNode, [1, 5])
            .build()
    )
]

evaluator = MulticlassClassificationEvaluator(
    labelCol=label_col,
    metricName="f1"
)
```

```
for name, pipeline, paramGrid in pipelines:
    cv = CrossValidator(
        estimator=pipeline,
        estimatorParamMaps=paramGrid,
        evaluator=evaluator,
        numFolds=5
    )

    cvModel = cv.fit(df)
    f1 = evaluator.evaluate(cvModel.transform(df))

    print(f"{name} CV F1 = {f1:.4f}")

    if f1 > best_f1:
        best_f1 = f1
        best_model = cvModel.bestModel
        best_name = name
```

Резултати:

lr CV F1 = 0.8258

rf CV F1 = 0.8264

dt CV F1 = 0.8319

Најдобар модел: Decision Tree со F1 = 0.8319

3. Online фаза

1. **Producer (producer.py праќа податоци ред по ред во Kafka)**
    - Податоците од online.csv се конвертираат во JSON.
    - Полето Diabetes_binary не се испраќа (за да се предвиди).
    - Секој ред се праќа на topic health_data.

```python
import json
import time
import pandas as pd
from kafka import KafkaProducer

df = pd.read_csv("online.csv")

producer = KafkaProducer(
    bootstrap_servers="host.docker.internal:9092",
    value_serializer=lambda v: json.dumps(v).encode("utf-8")
)

for _, row in df.iterrows():
    record = row.drop("Diabetes_binary").to_dict()
    producer.send("health_data", record)
    time.sleep(1)

```

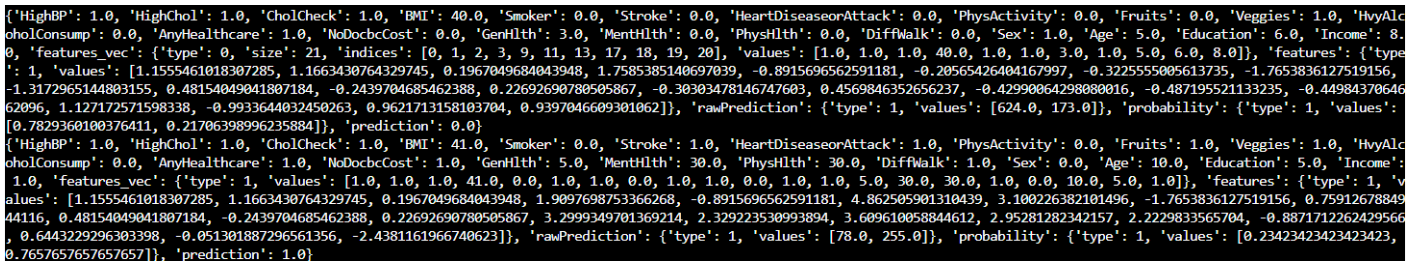
2. **Spark Structured Streaming апликација**
    - Вчитување на податоци од Kafka topic health_data.
    - Применување на истите трансформации како offline фазата (Pipeline).
    - Предвидување на класата со најдобриот модел.
    - Испраќање на резултатите во Kafka topic health_data_predicted.

```
model = PipelineModel.load("saved_models/best_dt")

stream_df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "host.docker.internal:9092") \
    .option("subscribe", "health_data") \
    .load()

parsed = stream_df.select(
    from_json(col("value").cast("string"), schema).alias("data")
).select("data.*")

predicted = model.transform(parsed)

output = predicted.selectExpr("to_json(struct(*)) AS value")

query = output.writeStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "host.docker.internal:9092") \
    .option("topic", "health_data_predicted") \
    .option("checkpointLocation", "/tmp/checkpoint") \
    .start()

query.awaitTermination()
```

Consumer.py

Дополнително креирав и едноставна Kafka consumer скрипта consumer.py. Оваа скрипта се поврзува на Kafka topic `health_data_predicted` и ги чита пораките што ги испраќа Spark апликацијата.

```
consumer = KafkaConsumer(
    "health_data_predicted",
    bootstrap_servers="host.docker.internal:9092",
    value_deserializer=lambda v: json.loads(v.decode("utf-8")),
    auto_offset_reset="earliest",
    enable_auto_commit=True,
)
```



Контејнеризација со Docker

- Користев Dockerfile за креирање на Spark околина со Python и сите ML библиотеки (pandas, scikit-learn, numpy, pyspark).
- Docker Compose конфигурација за Kafka и Zookeeper

Поставување на kafka:

docker compose up -d

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

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

Spark Structured Streaming се стартува со:

spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.4.3 online_stream.py

Jupyter Notebook околина:

docker run --rm -it -p 8888:8888 -v ${PWD}:/workspace spark-jupyter

Скриптите се извршуваат со: python offline.py, python producer.py, python consumer.py.

Редоследот на извршување за домашната задача е следниот

1. **Подигнување на Kafka и Zookeeper** (Docker Compose фајл):

   cd C:\docker-kafka

   docker compose up -d

2. **Креирање на Kafka topics**:

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

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

3. **Offline фаза - тренирање на модел**:

python offline.ipynb

- Се тренираат на моделот, се врши валидација и се зачувава најдобриот модел (best_dt) во saved_models/.

4. **Online фаза - Streaming предвидувања**:

spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.4.3 online_stream.py

- Ова стартува Spark Structured Streaming кој чита од health_data topic-от, го применува најдобриот модел и го пишува резултатот во health_data_predicted.

5. **Producer - испраќа податоци од online.csv во Kafka**:

python producer.py

6. **Consumer - прикажува предвидувања во конзола**:

python consumer.py

Изработил: Милош Кировски