

## 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.

```
1 import json
2 import time
3 import pandas as pd
4 from kafka import KafkaProducer
5
6 df = pd.read_csv("online.csv")
7
8 producer = KafkaProducer(
9     bootstrap_servers="host.docker.internal:9092",
10    value_serializer=lambda v: json.dumps(v).encode("utf-8")
11 )
12
13 for _, row in df.iterrows():
14     record = row.drop("Diabetes_binary").to_dict()
15     producer.send("health_data", record)
16     time.sleep(1)
17
```

### 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 креирам 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,
)

```

```

{'HighBP': 1.0, 'HighChol': 1.0, 'CholCheck': 1.0, 'BMI': 40.0, 'Smoker': 0.0, 'Stroke': 0.0, 'HeartDiseaseorAttack': 0.0, 'PhysActivity': 0.0, 'Fruits': 0.0, 'Veggies': 1.0, 'HvyAlc': 0.0, 'AnyHealthcare': 0.0, 'NoDacbcCost': 0.0, 'GenHlth': 3.0, 'MentHlth': 0.0, 'PhysHlth': 0.0, 'DiffWalk': 0.0, 'Sex': 1.0, 'Age': 5.0, 'Education': 6.0, 'Income': 8.0, 'features_vec': {'type': 0, 'size': 21, 'indices': [0, 1, 2, 3, 9, 11, 13, 17, 18, 19, 20], 'values': [1.0, 1.0, 1.0, 40.0, 1.0, 1.0, 3.0, 1.0, 5.0, 6.0, 8.0]}, 'features': {'type': 1, 'values': [1.1555461018307285, 1.1663430764329745, 0.1967049684043948, 1.7585385140697039, -0.8915696562591181, -0.20565426404167997, -0.3225555005613735, -1.7653836127519156, -1.3172965144803155, 0.48154049041807184, -0.2439704685462388, 0.22692690780505867, -0.30303478146747603, 0.4569846352656237, -0.42990064298080016, -0.487195521133235, -0.4498437064662096, 1.127172571598338, -0.9933644032450263, 0.9621713158103704, 0.9397046609301062]}, 'rawPrediction': {'type': 1, 'values': [624.0, 173.0]}, 'probability': {'type': 1, 'values': [0.7829360100376411, 0.21706398996235884]}, 'prediction': 0.0}
{'HighBP': 1.0, 'HighChol': 1.0, 'CholCheck': 1.0, 'BMI': 41.0, 'Smoker': 0.0, 'Stroke': 1.0, 'HeartDiseaseorAttack': 1.0, 'PhysActivity': 0.0, 'Fruits': 1.0, 'Veggies': 1.0, 'HvyAlc': 0.0, 'AnyHealthcare': 1.0, 'NoDacbcCost': 1.0, 'GenHlth': 5.0, 'MentHlth': 30.0, 'PhysHlth': 30.0, 'DiffWalk': 1.0, 'Sex': 0.0, 'Age': 10.0, 'Education': 5.0, 'Income': 1.0, 'features_vec': {'type': 1, 'values': [1.0, 1.0, 1.0, 41.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 5.0, 30.0, 30.0, 1.0, 0.0, 10.0, 5.0, 1.0]}, 'features': {'type': 1, 'values': [1.1555461018307285, 1.1663430764329745, 0.1967049684043948, 1.9097698753366268, -0.8915696562591181, 4.862505901310439, 3.100226382101496, -1.7653836127519156, 0.7591267884944116, 0.48154049041807184, -0.2439704685462388, 0.22692690780505867, 3.2999349701369214, 2.329223530993894, 3.609610058844612, 2.95281282342157, 2.2229833565704, -0.8871712262429566, 0.6443229296303398, -0.051301887296561356, -2.4381161966740623]}, 'rawPrediction': {'type': 1, 'values': [78.0, 255.0]}, 'probability': {'type': 1, 'values': [0.23423423423423423, 0.7657657657657657]}, 'prediction': 1.0}

```

## 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 зачувува 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