Домашна задача 1
Андреј Велков – 225126

Содржина

Поделба на множеството на offline и online (splitData.py)

Во овој чекор се вчитува оригиналното податочно множество и се врши негова поделба на training и test подмножества со користење на train_test_split. Поделбата се изведува на начин што ја задржува распределбата на класите преку stratified sampling, со што се обезбедува балансираност помеѓу податоците.

Потоа, добиените подмножества се зачувуваат како посебни CSV датотеки, при што training подмножеството се користи за offline тренирање на моделите, а test подмножеството се користи како online податоци во понатамошниот streaming процес.

```python
from sklearn.model_selection import train_test_split
import pandas as pd

df = pd.read_csv("diabetes_012_health_indicators_BRFSS2015.csv")

train, test = train_test_split(
    *arrays: df,
    test_size=0.2,
    stratify=df["Diabetes_012"],
    random_state=42
)

train.to_csv("data/offline.csv", index=False)
test.to_csv("data/online.csv", index=False)
```

Тренирање и избор на најдобар модел (offline-train.py)

За Dataset-от да биде компатибилен со Spark ML, сите колони освен колоната за класа мора да бидат претставени како вектор. За таа цел користиме VectorAssembler, а за нормализација и подобрување на рамнотежата на вредностите користиме и StandardScaler.

Во следниот дел ги дефинираме трите модели, од кои со тестирање во понатамошниот дел од кодот ќе определиме кој модел е најоптимален за нашата задача и ќе го користиме во Online делот.

```
assembler = VectorAssembler(inputCols=feature_columns, outputCol="raw_features")
scaler = StandardScaler(inputCol="raw_features", outputCol="features", withStd=True, withMean=True)

train_df, test_df = df.randomSplit( weights: [0.8, 0.2], seed=42)

models = [
    ("LogisticRegression",
     LogisticRegression(featuresCol="features", labelCol="Diabetes_012", maxIter=20, regParam=0.01)),
    ("RandomForest", RandomForestClassifier(featuresCol="features", labelCol="Diabetes_012", numTrees=50, maxDepth=10)),
    ("DecisionTree", DecisionTreeClassifier(featuresCol="features", labelCol="Diabetes_012", maxDepth=10))
]

evaluator = MulticlassClassificationEvaluator(labelCol="Diabetes_012", predictionCol="prediction", metricName="f1")
```

Како метрика за оценување на моделите користиме F1-score. По итерирање и тренирање на сите модели, најдобриот модел го зачувуваме локално за понатамошна употреба.

```
best_model = None
best_score = -1
best_name = ""

for name, model in models:
    pipeline = Pipeline(stages=[assembler, scaler, model])
    trained_model = pipeline.fit(train_df)

    predictions = trained_model.transform(test_df)
    f1 = evaluator.evaluate(predictions)

    accuracy = evaluator.setMetricName("accuracy").evaluate(predictions)

    print(f"\n{'=' * 50}")
    print(f"Model: {name}")
    print(f"F1 Score: {f1:.4f}")
    print(f"Accuracy: {accuracy:.4f}")

    print(f"Prediction distribution:")
    predictions.groupBy("prediction").count().orderBy("prediction").show()

    if f1 > best_score:
        best_score = f1
        best_model = trained_model
        best_name = name

model_path = "../models/best_model"
best_model.write().overwrite().save(model_path)
print(f"\n Best model: {best_name} saved with F1={best_score:.4f}")

spark.stop()
```

Streaming на online подмножеството (consume_messages.py)

Бидејќи претходно извршивме поделба на податочното множество на offline и online дел, online подмножеството можеме да го испраќаме во Kafka topic, од каде што понатаму ќе биде преземено и обработено со помош на Spark.

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

KAFKA_BROKER = "localhost:9092"
LABEL_COL = "Diabetes_012"
DATA_PATH = "/Users/andrej/Desktop/rnmp/RNMP_homework1/data/online.csv"
TOPIC_NAME = "health-data"

producer = KafkaProducer(
    bootstrap_servers=KAFKA_BROKER,
    value_serializer=lambda v: json.dumps(v).encode('utf-8'),
    security_protocol="PLAINTEXT"
)

df = pd.read_csv(DATA_PATH)

for idx, row in df.iterrows():
    data = row.drop(LABEL_COL).to_dict()
    producer.send(TOPIC_NAME, value=data)
    print(f"✅ Sent row {idx}: {data}")
    time.sleep(0.1)

producer.flush()
producer.close()
print(f"🎉 All data sent to Kafka topic: {TOPIC_NAME}")
```

Предвидување на класата (online-stream.py)

Во овој чекор се креира нова Spark сесија која, со примена на Structured Streaming, прима batch податоци пристигнати од Kafka topic.

```
TOPIC_NAME = "health-data"
OUTPUT_TOPIC = "health-data-predicted"
KAFKA_BOOTSTRAP_SERVER = "localhost:9092"

spark = SparkSession.builder \
    .appName("OnlineStreaming") \
    .master("local[*]") \
    .getOrCreate()

spark.sparkContext.setLogLevel("WARN")

model_path = "/Users/andrej/Desktop/rnmp/RNMP_homework1/models/best_model"
model = PipelineModel.load(model_path)

raw_stream = spark.readStream \
    .format("kafka") \
    .option( key: "kafka.bootstrap.servers", KAFKA_BOOTSTRAP_SERVER) \
    .option( key: "subscribe", TOPIC_NAME) \
    .option( key: "startingOffsets",  value: "latest") \
    .load()
```

Со цел правилно да ги обработиме податоците што пристигнуваат од Kafka, најпрво дефинираме **експлицитна шема (schema)** со користење на StructType и StructField. На овој начин точно ги опишуваме сите атрибути и нивните типови, со што овозможуваме коректно парсирање на JSON пораките.

Потоа, со помош на from_json функцијата, вредноста од Kafka пораката се кастира во string и се парсира според дефинираната шема, при што се добива структуриран Spark DataFrame подготвен за понатамошна обработка и анализа во streaming процесот.

```
schema = StructType([
    StructField( name: "HighBP", DoubleType()),
    StructField( name: "HighChol", DoubleType()),
    StructField( name: "CholCheck", DoubleType()),
    StructField( name: "BMI", DoubleType()),
    StructField( name: "Smoker", DoubleType()),
    StructField( name: "Stroke", DoubleType()),
    StructField( name: "HeartDiseaseorAttack", DoubleType()),
    StructField( name: "PhysActivity", DoubleType()),
    StructField( name: "Fruits", DoubleType()),
    StructField( name: "Veggies", DoubleType()),
    StructField( name: "HvyAlcoholConsump", DoubleType()),
    StructField( name: "AnyHealthcare", DoubleType()),
    StructField( name: "NoDocbcCost", DoubleType()),
    StructField( name: "GenHlth", DoubleType()),
    StructField( name: "MentHlth", DoubleType()),
    StructField( name: "PhysHlth", DoubleType()),
    StructField( name: "DiffWalk", DoubleType()),
    StructField( name: "Sex", DoubleType()),
    StructField( name: "Age", DoubleType()),
    StructField( name: "Education", DoubleType()),
    StructField( name: "Income", DoubleType()),
])

stream_df = raw_stream.selectExpr("CAST(value AS STRING) as json_str")
parsed_df = stream_df.select(from_json(col("json_str"), schema).alias("data")).select("data.*")
```

Откако податоците се парсирани и подготвени, врз нив се применува претходно истренираниот ML модел со користење на transform методот, при што се добиваат предикции за секој влезен запис.

Потоа се креира излезен DataFrame во кој, покрај оригиналните карактеристики, се вклучуваат и резултатите од моделот – предвидената класа и веројатноста на предикцијата. Овие податоци се структуираат и серијализираат во JSON формат, со цел да бидат соодветни за испраќање преку Kafka.

На крај, со користење на Spark Structured Streaming, резултатите се испраќаат во излезен Kafka topic, со овозможен checkpoint механизам за сигурна и континуирана streaming обработка.

```
predictions = model.transform(parsed_df)

feature_columns = [field.name for field in schema.fields]

output_df = predictions.withColumn(
    "value",
    to_json(
        struct(
            *[col(c) for c in feature_columns],
            col("prediction").alias("predicted_class"),
            col("probability").alias("prediction_probability")
        )
    )
).selectExpr("CAST(value AS STRING)")

kafka_query = output_df.writeStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", KAFKA_BOOTSTRAP_SERVER) \
    .option("topic", OUTPUT_TOPIC) \
    .option("checkpointLocation", "/tmp/kafka-checkpoint") \
    .outputMode("append") \
    .start()

spark.streams.awaitAnyTermination()
```

Прикажување на модифицираните податоци (consume_messages.py)

Претходно, модифицираните податоци, со додадената предвидена класа се испраќаат во нов Kafka topic. Преку Kafka Consumer, вршиме визуелен приказ на испратените податоци заедно со предвидената класа.

```python
from kafka import KafkaConsumer

bootstrap_servers = 'localhost:9092'
topic = 'health-data-predicted'

consumer = KafkaConsumer( *topics: topic,
                          bootstrap_servers=bootstrap_servers,
                          group_id='my_consumer_group',
                          auto_offset_reset='earliest',
                          enable_auto_commit=False)

try:
    for message in consumer:
        print(f"Received message: {message.value.decode('utf-8')}")
except KeyboardInterrupt:
    pass
finally:
    consumer.close()
```