Документација за Третата Домашна по Рударење на Масивни Податоци

Дарко Петрушевски 226013

1. Поделба - podelba.py
   Во оваа скрипта имам направено најобично поделување на датасетот , каде што го load-нувам и потоа го делим на offline_df и online_df со 80% и 20% и на крај зачувувам.

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

df = pd.read_csv('diabetes_binary_health_indicators_BRFSS2015.csv')

target = df["Diabetes_binary"]

offline_df, online_df = train_test_split( *arrays: df, test_size=0.2, random_state=42, stratify=target)
offline_df.to_csv('offline.csv', index=False)
online_df.to_csv('online.csv', index=False)
```

2. Офлајн фаза – offline_faza.py
   Во offline_faza.py креирам SparkSession и го вчитувам offline.csv. Потоа креирам целна колона label и правам random split на 80% train и 20% test. За да можам да применам ML модели во Spark, ги трансформирам feature колоните во една вектор колона features со VectorAssembler и ја пакувам оваа трансформација заедно со моделот во Pipeline. Потоа обучувам три класификатори Logistic Regression, Random Forest и Gradient Boosting и за секој користам CrossValidator со ParamGrid за избор на најдобри хиперпараметри според F1 метрика. На крај го споредувам најдобриот модел од секој алгоритам на test делот, го избирам моделот со највисок F1 и го зачувувам локално. Поради Windows околина поставувам HADOOP_HOME/winutils за успешно серијализирање на Spark моделот.
   Резултати од ова:

```
#
# |Diabetes_binary|HighBP|HighChol|CholCheck|
BMI|Smoker|Stroke|HeartDiseaseorAttack|PhysActivity|Fruits|Veggies|HvyAlcoholConsump
|AnyHealthcare|NoDocbcCost|GenHlth|MentHlth|PhysHlth|DiffWalk|Sex|
Age|Education|Income|
# +---------------+------+--------+---------+----+------+------+--------------------+------------+------+---
----+-----------------+-------------+-----------+-------+--------+--------+--------+---+----+---------+------
+
# |            0.0|   0.0|     0.0|      1.0|28.0|   1.0|   0.0|             0.0|       1.0|   1.0|    1.0|           0.0|
1.0|      0.0|    2.0|     0.0|     0.0|     0.0|1.0| 2.0|     4.0|  5.0|
# |            0.0|   1.0|     0.0|      1.0|23.0|   1.0|   0.0|             0.0|       1.0|   1.0|    1.0|           0.0|
```

```
1.0|     0.0|  2.0|    0.0|    0.0|    0.0|1.0|13.0|    4.0|  7.0|
# |       0.0|  1.0|    1.0|    1.0|29.0|  0.0|  0.0|         0.0|     1.0|  1.0|   1.0|         0.0|
1.0|     0.0|  1.0|    0.0|    0.0|    0.0|1.0| 9.0|    6.0|  8.0|
# |       0.0|  1.0|    1.0|    1.0|39.0|  0.0|  0.0|         0.0|     0.0|  0.0|   0.0|         0.0|
1.0|     0.0|  4.0|    0.0|    0.0|    0.0|1.0| 7.0|    4.0|  7.0|
# |       0.0|  0.0|    1.0|    1.0|16.0|  1.0|  0.0|         0.0|     1.0|  1.0|   1.0|         0.0|
1.0|     1.0|  5.0|   30.0|   30.0|    1.0|0.0| 7.0|    5.0|  1.0|
# |       0.0|  1.0|    0.0|    1.0|32.0|  0.0|  0.0|         0.0|     1.0|  0.0|   1.0|         0.0|
1.0|     0.0|  2.0|    0.0|    0.0|    0.0|1.0| 7.0|    6.0|  8.0|
# |       1.0|  1.0|    0.0|    1.0|37.0|  1.0|  0.0|         0.0|     0.0|  0.0|   1.0|         0.0|
1.0|     0.0|  4.0|    0.0|    0.0|    0.0|0.0| 4.0|    5.0|  4.0|
# |       0.0|  0.0|    0.0|    1.0|27.0|  0.0|  0.0|         0.0|     1.0|  1.0|   0.0|         0.0|
1.0|     0.0|  2.0|    3.0|    1.0|    0.0|1.0| 5.0|    5.0|  6.0|
# |       0.0|  0.0|    0.0|    1.0|24.0|  0.0|  0.0|         0.0|     1.0|  1.0|   0.0|         0.0|
1.0|     0.0|  3.0|   10.0|   15.0|    1.0|0.0| 5.0|    6.0|  8.0|
# |       0.0|  0.0|    0.0|    1.0|23.0|  1.0|  0.0|         0.0|     1.0|  1.0|   1.0|         1.0|
1.0|     0.0|  2.0|    0.0|    3.0|    0.0|0.0| 6.0|    6.0|  8.0|
# +---------------+------+--------+---------+----+------+------+--------------------+------------+------+---
----+-----------------+-------------+-----------+-------+--------+--------+--------+---+----+---------+------
+
# only showing top 10 rows
#
# Logistic Regression F1-score: 0.8307658510684387
# Random Forest F1-score: 0.8241547930166929
# Gradient Boosting F1-score: 0.8343585722026596
# Best model: PipelineModel_58bdafdf4ea4 with F1-score 0.8343585722026596 and
Gradient Boosting
```

```python
def build_pipeline(estimator, features):  3 usages
    assembler = VectorAssembler(inputCols=features, outputCol="features")
    return Pipeline(stages=[assembler, estimator])


def crossval_fit(pipeline, param_grid, train_df, evaluator, folds=3, seed=42, parallelism=4):  3 usages
    cv = CrossValidator(
        estimator=pipeline,
        estimatorParamMaps=param_grid,
        evaluator=evaluator,
        numFolds=folds,
        seed=seed,
        parallelism=parallelism
    )
    return cv.fit(train_df)
```

```python
def main():  1 usage
    spark = SparkSession.builder \
        .appName("Domashna3") \
        .master("local[*]") \
        .getOrCreate()

    df = spark.read.csv( path: "offline.csv", header=True, inferSchema=True)
    df.show(10)

    features = [
        "HighBP", "HighChol", "CholCheck", "BMI", "Smoker",
        "Stroke", "HeartDiseaseorAttack", "PhysActivity",
        "Fruits", "Veggies", "HvyAlcoholConsump",
        "AnyHealthcare", "NoDocbcCost", "GenHlth",
        "MentHlth", "PhysHlth", "DiffWalk", "Sex",
        "Age", "Education", "Income"
    ]

    data = df.withColumn( colName: "label", col("Diabetes_binary").cast("double"))
    train_data, test_data = data.randomSplit( weights: [0.8, 0.2], seed=42)
    evaluator = MulticlassClassificationEvaluator(labelCol="label",predictionCol="prediction", metricName="f1")
```

```python
    lr = LogisticRegression(featuresCol="features", labelCol="label")
    lr_pipe = build_pipeline(lr, features)
    lr_grid = (ParamGridBuilder().
                addGrid(lr.regParam,  values: [0.001, 0.01, 0.1])
                .addGrid(lr.elasticNetParam,  values: [0.0, 0.5, 1.0])
                .addGrid(lr.maxIter,  values: [10, 30, 80])
                .build())

    lr_cv_model = crossval_fit(lr_pipe, lr_grid, train_data, evaluator)
    lr_best = lr_cv_model.bestModel
    lr_f1 = evaluator.evaluate(lr_best.transform(test_data))
    print(f"Logistic Regression F1-score: {lr_f1}")

    rf = RandomForestClassifier(featuresCol="features", labelCol="label", seed=42)
    rf_pipe = build_pipeline(rf, features)

    rf_grid = (ParamGridBuilder()
               .addGrid(rf.numTrees,  values: [50, 100])
               .addGrid(rf.maxDepth,  values: [5, 10])
               .addGrid(rf.featureSubsetStrategy,  values: ["auto", "sqrt"])
               .build())

    rf_cv_model = crossval_fit(rf_pipe, rf_grid, train_data, evaluator)
    rf_best = rf_cv_model.bestModel
    rf_f1 = evaluator.evaluate(rf_best.transform(test_data))
    print(f"Random Forest F1-score: {rf_f1}")
```

```python
  gbt_grid = (ParamGridBuilder()
              .addGrid(gbt.maxDepth,  values: [3, 5])
              .addGrid(gbt.maxIter,  values: [20, 50])
              .addGrid(gbt.stepSize,  values: [0.05, 0.1])
              .build())

  gbt_cv_model = crossval_fit(gbt_pipe, gbt_grid, train_data, evaluator)
  gbt_best = gbt_cv_model.bestModel
  gbt_f1 = evaluator.evaluate(gbt_best.transform(test_data))
  print(f"Gradient Boosting F1-score: {gbt_f1}")

  best_models = [(lr_best, lr_f1, "Logistic Regression"),(rf_best, rf_f1, "Random Forest"),(gbt_best, gbt_f1, "Gradient Boosting")]
  model, f1, name = max(best_models, key=lambda x: x[1])
  print(f"Best model: {model} with F1-score {f1} and {name}")

  out_dir = Path("C:/Temp/spark_models") / f"best_model_{name}"
  out_uri = out_dir.resolve().as_uri()
  print("Saving model to:", out_uri)
  model.write().overwrite().save(out_uri)
```

3. Продусер – producer.py
   Во producer_script.py прво ги дефинирам feature колоните и со argparse овозможувам скриптата лесно да се стартува со параметри за CSV датотека, Kafka broker, topic и пауза меѓу пораки. Потоа креирам KafkaProducer, го читам csv со pandas и ја отстранувам Diabetes_binary колоната бидејќи во online фазата не смее да се праќа класата. Следно ги задржувам само feature колоните и секој ред го претворам во JSON објект кој го испраќам во Kafka topic health_data, со мала пауза за да симулирам поток на податоци, а периодично правам flush() за сигурно праќање и печатам колку записи се испратени.

```python
def main():  1 usage
    parser = argparse.ArgumentParser()
    parser.add_argument( *name_or_flags: "--file", default="offline.csv")
    parser.add_argument( *name_or_flags: "--bootstrap", default="localhost:9092")
    parser.add_argument( *name_or_flags: "--topic", default="health_data")
    parser.add_argument( *name_or_flags: "--sleep", type=float, default=0.05)
    args = parser.parse_args()

    producer = KafkaProducer(
        bootstrap_servers=args.bootstrap,
        value_serializer=lambda v: v.encode("utf-8")
    )

    df = pd.read_csv(args.file)

    if label in df.columns:
        df = df.drop(columns=[label])

    df = df[features]
    counter = 0
    for _, row in df.iterrows():
        msg = row.to_frame().T.to_json(orient="records")
        payload = msg[1:-1]

        producer.send(args.topic, payload)
        counter += 1

        if counter % 1000 == 0:
            producer.flush()
            print(f"Sent {counter}")

        time.sleep(args.sleep)

    producer.flush()
    print(f"Sent {counter}")
```

4. Онлајн фазата – online_faza.py
   Во online_faza.py креирам SparkSession и ја додавам потребната Kafka интеграција за да можам да читам/пишувам Structured Streaming од Kafka. Потоа го вчитувам најдобриот зачуван PipelineModel од offline фазата и дефинирам шема за feature колоните. Следно преку readStream читам JSON записи од Kafka topic health_data, ги парсирам во колони со from_json, и врз секој запис го применувам моделот со model.transform() за да добијам prediction и probability. На крај правам enrich на записите со predicted_class и probability, ги претворам назад во JSON и ги праќам во нов Kafka topic health_data_predicted, користејќи checkpoint за стабилно streaming извршување.

```
spark = SparkSession.builder \
    .appName("Domashna3") \
    .master("local[*]") \
    .config("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.3") \
    .getOrCreate()

features = [
    "HighBP", "HighChol", "CholCheck", "BMI", "Smoker",
    "Stroke", "HeartDiseaseorAttack", "PhysActivity",
    "Fruits", "Veggies", "HvyAlcoholConsump",
    "AnyHealthcare", "NoDocbcCost", "GenHlth",
    "MentHlth", "PhysHlth", "DiffWalk", "Sex",
    "Age", "Education", "Income"
]

label = "Diabetes_binary"

model_path = r".\spark_models\best_model_Gradient%20Boosting"
model = PipelineModel.load(model_path)
```

```
schema = StructType([StructField(f, DoubleType(), nullable: True) for f in features])

kafka_df = (spark.readStream.format("kafka").option( key: "kafka.bootstrap.servers", value: "localhost:9092")
            .option( key: "subscribe", value: "health_data").load())

parsed_df = kafka_df.selectExpr("CAST(value AS STRING)").select(from_json(col("value"), schema).alias("data")).select("data.*")

preds = model.transform(parsed_df)
out_df = preds.select(*[col(f) for f in features],
    col("prediction").cast("int").alias("predicted_class"),
    col("probability").alias("probability")
)
out_kafka_df = out_df.select(to_json(struct(*out_df.columns)).alias("value"))

query = (out_kafka_df.writeStream
         .format("kafka")
         .option( key: "kafka.bootstrap.servers", value: "localhost:9092")
         .option( key: "topic", value: "health_data_predicted")
         .option( key: "checkpointLocation", value: "checkpoints/health_data_predicted_sparkmodel")
         .outputMode("append")
         .start())

query.awaitTermination()
```

5. Конзумер – consumer_script.py
   Оваа скрипта ми овозможува да го видам output-от. На крај може да се видат и predicated_class со probability. Резултатите кога ќе се пушти цел pipeline:

Listening on topic 'health_data_predicted' @ localhost:9092

Offsets: latest

{

"HighBP": 0.0,

"HighChol": 0.0,

"CholCheck": 1.0,

"BMI": 23.0,

"Smoker": 1.0,

"Stroke": 1.0,

"HeartDiseaseorAttack": 0.0,

"PhysActivity": 1.0,

"Fruits": 1.0,

"Veggies": 1.0,

"HvyAlcoholConsump": 0.0,

"AnyHealthcare": 1.0,

"NoDocbcCost": 0.0,

"GenHlth": 2.0,

"MentHlth": 0.0,

"PhysHlth": 30.0,

"DiffWalk": 0.0,

"Sex": 0.0,

"Age": 12.0,

"Education": 6.0,

"Income": 8.0,

"predicted_class": 0,

```
 "probability": {
  "type": 1,
  "values": [
   0.9447798162713832,
   0.05522018372861681
  ]
 }
}
```

```
{
 "HighBP": 1.0,
 "HighChol": 0.0,
 "CholCheck": 1.0,
 "BMI": 25.0,
 "Smoker": 1.0,
 "Stroke": 0.0,
 "HeartDiseaseorAttack": 0.0,
 "PhysActivity": 0.0,
 "Fruits": 0.0,
 "Veggies": 1.0,
 "HvyAlcoholConsump": 0.0,
 "AnyHealthcare": 1.0,
 "NoDocbcCost": 0.0,
 "GenHlth": 3.0,
 "MentHlth": 0.0,
 "PhysHlth": 20.0,
 "DiffWalk": 0.0,
 "Sex": 1.0,
 "Age": 9.0,
```

```
  "Education": 6.0,
  "Income": 4.0,
  "predicted_class": 0,
  "probability": {
    "type": 1,
    "values": [
      0.8498360839950161,
      0.1501639160049839
    ]
  }
}
```