SISTEMI ZA OBRADU I ANALIZU VELIKE KOLIČINE PODATAKA - Projekat 3

Filip Trajković 1574

Spisak kontejnera

project3	-	Running (13/1			
producer-3 6ec4f81b6671	project3-producer-3	Running		2 hours ago	
grafana 71d45dc7da40	grafana/grafana	Running	3000:3000	2 hours ago	
influxdb 99e7fc7f22ae	influxdb:2.1.1	Running	8086:8086	2 hours ago	
spark-worker-1-3 9862025a776f	bde2020/spark-worker:3.1.2-hadoop3.2	Running	8071:8071	2 hours ago	
kafka-3 6116e0ba6bfc	wurstmeister/kafka:2.13-2.7.0	Running	9091:9091	2 hours ago	
spark-worker-2-3 f17857b32418	bde2020/spark-worker:3.1.2-hadoop3.2	Running	8072:8071	2 hours ago	
datanode-3 fcc091cff35e	bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8	Running		2 hours ago	
zookeeper-3 dffe329403ad	wurstmeister/zookeeper:latest	Running	2181:2181	2 hours ago	
namenode-3 660c9b46b75a	bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8	Running	9000:9000 9870:9870	2 hours ago	
nodemanager-3 8b289c347142	bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8	Running		2 hours ago	
resourcemanager-3 06cb7d146f32	bde2020/hadoop-resourcemanager:2.0.0-hadoop3.2.1-java8	Running		1 hour ago	
historyserver-3 0f10928ac545	bde2020/hadoop-historyserver:2.0.0-hadoop3.2.1-java8	Running		2 hours ago	
spark-master-3 40077f2a7138	bde2020/spark-master:3.1.2-hadoop3.2	Running	7077:7077 8070:8070	2 hours ago	

Izgled Producer skripte

```
producer = KafkaProducer(
    bootstrap_servers=[os.environ["KAFKA_HOST"]],
    value_serializer=lambda v: json.dumps(v).encode("utf-8"),
    api_version=(0, 11),
)

while True:
    with open(os.environ["DATA"], "r") as file:
        reader = csv.reader(file, delimiter=",")
        headers = next(reader)
        for row in reader:
            value = {headers[i]: row[i] for i in range(len(headers))}
            value["lat"] = float(value["lat"])
            value["lon"] = float(value["lon"])
            value["alt"] = float(value["alt"])
            value["user"] = int(value["user"])
            value["ts"] = int(time.time())
            value["year"] = int(value["year"])
            value["month"] = int(value["month"])
            value["day"] = int(value["day"])
            value["hour"] = int(value["hour"])
            value["minute"] = int(value["minute"])
            value["second"] = int(value["second"])
            print(value)
            producer.send(os.environ["KAFKA_TOPIC"], value=value)
            time.sleep(float(os.environ["KAFKA_INTERVAL"]))

```

Aplikacija za kreiranje modela za klasifikaciju

→

Dockerfile

Project 3 > model-creation > Dockerfile > ...

```
1   FROM bde2020/spark-python-template:3.1.2-hadoop3.2
2
3   RUN apk add --no-cache py3-numpy
4
5   ENV CLASSIFICATION_MODEL='hdfs://namenode:9000/model/RfModel'
6   ENV SCALER_LOC="hdfs://namenode:9000/model/Scaler"
7   ENV INDEXER_PIPELINE_LOC="hdfs://namenode:9000/model/IndexerPipeline"
8
9
10  ENV SPARK_APPLICATION_PYTHON_LOCATION /app/model_creation.py
11  ENV SPARK_APPLICATION_ARGS "hdfs://namenode:9000/user/root/geolife_gps_sorted_without_nodata.csv"
12  ENV SPARK_SUBMIT_ARGS --executor-memory 4G --executor-cores 4
```

Aplikacija za kreiranje modela za klasifikaciju

→

Shell skripta za pokretanje

Project 3 > model-creation > $ start-streaming-app.sh

```
1  #! /bin/bash
2
3  docker container rm ModelCreationApp
4
5  docker build --rm -t bde/model_creation_app .
6
7  docker run --name ModelCreationApp --net FT_project3_network -p 4042:4042 bde/model_creation_app
8
9
```

Aplikacija za kreiranje modela za klasifikaciju

Uključene biblioteke, šema podataka i environment promenljive

Project 3 > model-creation > model_creation.py > ...

```python
import os
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, FloatType, IntegerType, DoubleType
from pyspark.sql.functions import col, from_json, window, count, mean, max, min
from pyspark import SparkConf
import sys
from pyspark.ml.feature import StringIndexer, VectorAssembler, MinMaxScaler, StringIndexerModel
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml import PipelineModel
from pyspark.ml import Pipeline

schema = StructType(
    [
        StructField("lat", FloatType()),
        StructField("lon", FloatType()),
        StructField("alt", FloatType()),
        StructField("label", StringType()),
        StructField("user", StringType()),
        StructField("year", IntegerType()),
        StructField("month", IntegerType()),
        StructField("day", IntegerType()),
        StructField("hour", IntegerType()),
        StructField("minute", IntegerType()),
        StructField("second", IntegerType())
    ]
)

if __name__ == '__main__':
    if len(sys.argv) < 2:
        print("Usage: main.py <input folder> ")
        exit(-1)

    SCALER_LOCATION = os.getenv('SCALER_LOC')
    INDEXER_PIPELINE_LOC = os.getenv('INDEXER_PIPELINE_LOC')
    CLASSIFICATION_MODEL = os.getenv('CLASSIFICATION_MODEL')
```

Aplikacija za kreiranje modela za klasifikaciju

Preprocesiranje podataka

```
62      label_indexer = StringIndexer(
63          inputCol="label", outputCol="label_index", handleInvalid="keep")
64      user_indexer = StringIndexer(
65          inputCol="user", outputCol="user_index")
66      # pipeline = Pipeline(stages=[label_indexer])
67
68      pipeline = Pipeline(stages=[label_indexer, user_indexer])
69
70      fitted_indexer = pipeline.fit(df)
71
72      fitted_indexer.write().overwrite().save(INDEXER_PIPELINE_LOC) #("hdfs://namenode:9000/model/IndexerPipeline")
73
74      indexer_model = PipelineModel.load(INDEXER_PIPELINE_LOC) #("hdfs://namenode:9000/model/IndexerPipeline")
75
76      indexedDf = indexer_model.transform(df)
77
78      # indexedDf = fitted_label_indexer.transform(df)
79
80      indexedDf = indexedDf.drop("label")
81      indexedDf = indexedDf.drop("user")
82
83      features = ["lat", "lon", "alt", "user_index",
84                  "year", "month", "day", "hour", "minute", "second"]
85
86      featureAssembler = VectorAssembler(
87          inputCols=features, outputCol="features")
88
89      assembleredDf = featureAssembler.transform(indexedDf)
90
91      scaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")
92
93      fitScaler = scaler.fit(assembleredDf)
94
95      fitScaler.write().overwrite().save(SCALER_LOCATION)
96
97
98      scaledData = fitScaler.transform(assembleredDf)
99
100     scaledData.show()
101
102     finalDf = scaledData.select("scaledFeatures", "label_index")
103
```

Aplikacija za kreiranje modela za klasifikaciju

Kreiranje i evaluacija modela

```
112
113     train_data, test_data = finalDf.randomSplit([0.8, 0.2], seed=42)
114
115     rf = RandomForestClassifier(
116         featuresCol='scaledFeatures', labelCol='label_index', maxDepth=10)
117
118     rfModel = rf.fit(train_data)
119
120     predictions = rfModel.transform(test_data)
121
122
123 # metricName="f1"
124 # metricName="precisionByLabel"
125 # metricName="recallByLabel"
126
127     evaluatorAccuracy = MulticlassClassificationEvaluator(
128         labelCol="label_index", predictionCol="prediction", metricName="accuracy")
129     evaluatorF1 = MulticlassClassificationEvaluator(
130         labelCol="label_index", predictionCol="prediction", metricName="f1")
131     evaluatorByLabel = MulticlassClassificationEvaluator(
132         labelCol="label_index", predictionCol="prediction", metricName="precisionByLabel")
133     evaluatorRecallByLabel = MulticlassClassificationEvaluator(
134         labelCol="label_index", predictionCol="prediction", metricName="recallByLabel")
135
136     print(evaluatorAccuracy.evaluate(predictions))
137     print(evaluatorAccuracy.evaluate(predictions))
138     print(evaluatorByLabel.evaluate(predictions))
139     print(evaluatorRecallByLabel.evaluate(predictions))
140
141     rfModel.write().overwrite().save(CLASSIFICATION_MODEL) #("hdfs://namenode:9000/model/RfModel")
142
```

Aplikacija za predikciju novih podataka

Shell skripta za pokretanje

Project 3 > spark-streaming-prediction > $ start-streaming-app.sh

```
1  #! /bin/bash
2
3  docker container rm StreamingPredictionApp
4
5  docker build --rm -t bde/streaming_prediction_app .
6
7  docker run --name StreamingPredictionApp --net FT_project3_network -p 4043:4043 bde/streaming_prediction_app
8
9
```

Aplikacija za predikciju novih podataka

Dockerfile

Project 3 > spark-streaming-prediction > Dockerfile > ...

```
1   FROM bde2020/spark-python-template:3.1.2-hadoop3.2
2
3   RUN apk add --no-cache py3-numpy
4   RUN cd /app pip install -r requirements.txt
5
6   ENV INFLUXDB_HOST "influxdb"
7   ENV INFLUXDB_PORT "8086"
8   ENV INFLUXDB_USERNAME "admin"
9   ENV INFLUXDB_PASSWORD "admin"
10  ENV INFLUXDB_TOKEN "n6zAU61C3paPpmfY3dvbQLoVORcqcd0STCCw811VB7MUxEKbppao5CSUg8sMaRzSlWiNI09OH6iHMzAWfU_1yw=="
11  ENV INFLUXDB_ORG "bigdata_projekat"
12  ENV INFLUXDB_BUCKET "locations"
13
14  ENV CLASSIFICATION_MODEL='hdfs://namenode:9000/model/RfModel'
15  ENV SCALER_LOC="hdfs://namenode:9000/model/Scaler"
16  ENV INDEXER_PIPELINE_LOC="hdfs://namenode:9000/model/IndexerPipeline"
17
18  ENV KAFKA_HOST=kafka-3:9092
19  ENV KAFKA_TOPIC=locations
20  ENV KAFKA_CONSUMER_GROUP=Spark-Group
21
22  ENV SPARK_APPLICATION_PYTHON_LOCATION /app/streaming_prediction.py
23  ENV SPARK_APPLICATION_ARGS "hdfs://namenode:9000/user/root/geolife_gps_sorted_without_nodata.csv"
24  ENV SPARK_SUBMIT_ARGS --packages org.apache.spark:spark-streaming-kafka-0-10_2.12:3.1.2,org.apache.spark:spark-sql-kafka-0-10_2.12:3.1.2 --executor-memory 4G --executor-cores 4
```

Aplikacija za predikciju novih podataka

Uključene biblioteke, šema podataka i environment promenljive

```python
import os
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, FloatType, IntegerType, DoubleType
from pyspark.sql.functions import col, from_json, window, count, mean, max, min
from pyspark import SparkConf
import sys
from pyspark.ml.feature import StringIndexer, VectorAssembler, MinMaxScalerModel
from pyspark.ml.classification import RandomForestClassifier, RandomForestClassificationModel
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml import PipelineModel
from pyspark.ml import Pipeline

# InfluxDB libraries
from influxdb_client import InfluxDBClient, Point, WritePrecision
from influxdb_client.client.write_api import SYNCHRONOUS
from datetime import datetime

schema = StructType(
    [
        StructField("lat", FloatType()),
        StructField("lon", FloatType()),
        StructField("alt", FloatType()),
        StructField("label", StringType()),
        StructField("user", StringType()),
        StructField("year", IntegerType()),
        StructField("month", IntegerType()),
        StructField("day", IntegerType()),
        StructField("hour", IntegerType()),
        StructField("minute", IntegerType()),
        StructField("second", IntegerType())
    ]
)

INFLUXDB_ORG = os.getenv('INFLUXDB_ORG')
INFLUXDB_BUCKET = os.getenv('INFLUXDB_BUCKET')
INFLUXDB_HOST = os.getenv('INFLUXDB_HOST')
INFLUXDB_PORT = os.getenv('INFLUXDB_PORT')
INFLUXDB_TOKEN = os.getenv('INFLUXDB_TOKEN')
```

Aplikacija za predikciju novih podataka

Klasa za upis u InfluxDB bazu podataka

```python
class InfluxDBWriter:
    def __init__(self):
        self._org = INFLUXDB_ORG
        self._token = INFLUXDB_TOKEN
        self.client = InfluxDBClient(url=f"http://{INFLUXDB_HOST}:{INFLUXDB_PORT}", token=self._token, org=self._org)
        self.write_api = self.client.write_api(write_options=SYNCHRONOUS)

    def open(self, partition_id, epoch_id):
        print(f"Opened {partition_id}, {epoch_id}")
        return True

    def process(self, row):
        self.write_api.write(bucket=INFLUXDB_BUCKET, record=self._row_to_point(row))

    def close(self, error):
        self.write_api.__del__()
        self.client.__del__()
        print(f"Closed with error: {error}")

    def _row_to_point(self, row):
        print(row)

        # timestamp = datetime.utcnow().strftime('%Y-%m-%dT%H:%M:%SZ')

        point = (
            Point.measurement("locations_measurement2")
            .tag("measure", "locations_measurement2")
            .time(datetime.utcnow(), WritePrecision.NS)
            .field("Latitude", float(row['lat']))
            .field("Longitude", float(row['lon']))
            .field("Altitude", float(row['alt']))
            .field("Label_index", int(row['label_index']))
            .field("User_index", int(row['user_index']))
            .field("Label_prediction", int(row['prediction']))
        )
        return point

```

Aplikacija za predikciju novih podataka

Postavljanje environment promenljivih i učitavanje podataka sa Kafka topic-a

```python
if __name__ == '__main__':
    if len(sys.argv) < 2:
        print("Usage: main.py <input folder> ")
        exit(-1)

    INDEXER_PIPELINE_LOC = os.getenv('INDEXER_PIPELINE_LOC')
    SCALER_LOCATION = os.getenv('SCALER_LOC')
    CLASSIFICATION_MODEL = os.getenv('CLASSIFICATION_MODEL')

    INFLUXDB_HOST = os.getenv('INFLUXDB_HOST')
    INFLUXDB_PORT = os.getenv('INFLUXDB_PORT')
    INFLUXDB_USERNAME = os.getenv('INFLUXDB_USERNAME')
    INFLUXDB_PASSWORD = os.getenv('INFLUXDB_PASSWORD')
    INFLUXDB_DATABASE = os.getenv('INFLUXDB_DATABASE')

    KAFKA_HOST = os.getenv('KAFKA_HOST')
    KAFKA_TOPIC = os.getenv('KAFKA_TOPIC')
    KAFKA_CONSUMER_GROUP = os.getenv('KAFKA_CONSUMER_GROUP')


    conf = SparkConf()
    conf.setMaster("spark://spark-master-3:7077")
    #conf.setMaster("local")
    # conf.set("spark.driver.memory","4g")
    # conf.set("spark.cassandra.connection.host", "cassandra")
    # conf.set("spark.cassandra.connection.port", 9042)

    spark = SparkSession.builder.config(
        conf=conf).appName("StreamingPreditionApp").getOrCreate()

    spark.sparkContext.setLogLevel("ERROR")

    df = (
        spark.readStream.format("kafka")
        .option("kafka.bootstrap.servers", KAFKA_HOST)
        .option("subscribe", KAFKA_TOPIC)
        .option("startingOffsets", "latest")
        .option("groupIdPrefix", KAFKA_CONSUMER_GROUP)
        .load()
    )
```

Aplikacija za predikciju novih podataka

→

Primena preprocesiranja nad novim podacima, izvršenje predikcije i upis u InfluxDB bazu podataka

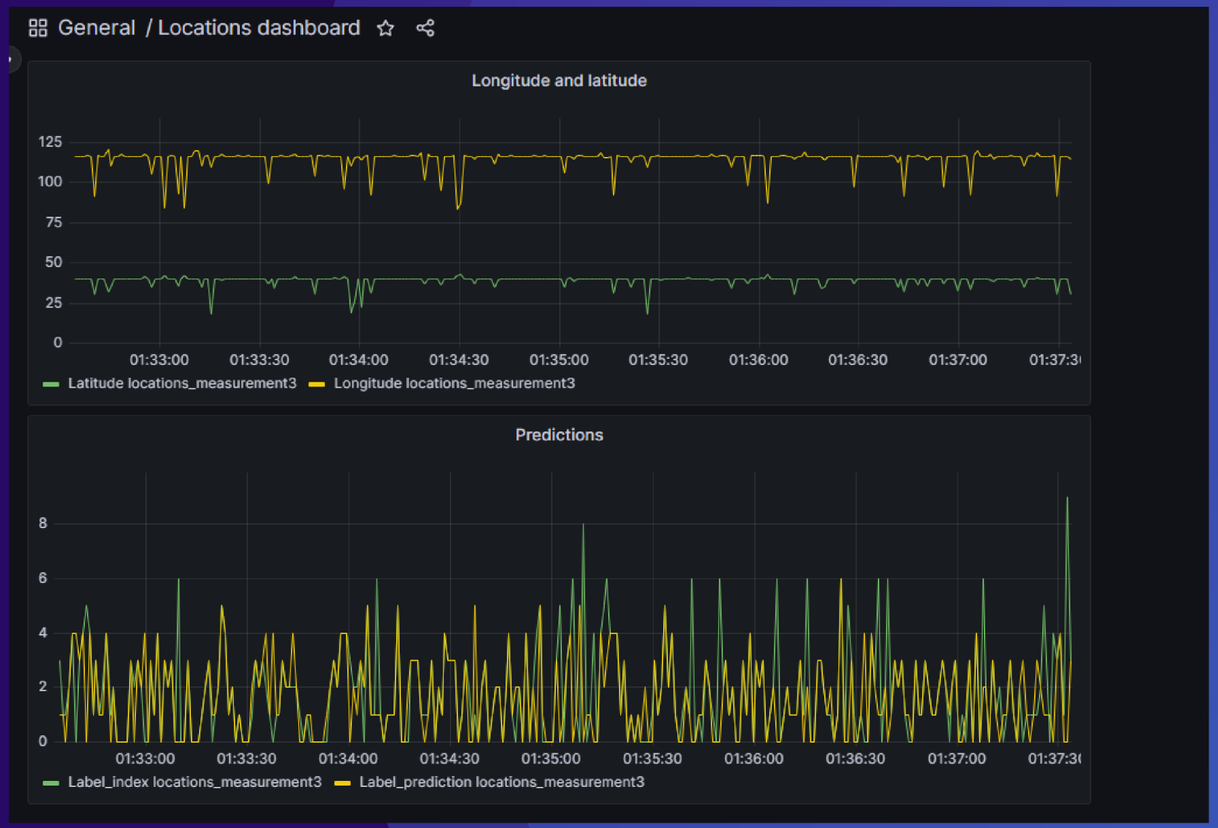
```
    loaded_values = df.select(
        from_json(col("value").cast(
            "string"), schema).alias("data")
    )

    parsed_values = loaded_values.selectExpr("data.*")
    parsed_values.printSchema()

    indexer_model = PipelineModel.load(INDEXER_PIPELINE_LOC) #("hdfs://namenode:9000/model/IndexerPipeline")
    scaler_model = MinMaxScalerModel.load(SCALER_LOCATION)
    classification_model = RandomForestClassificationModel.load(CLASSIFICATION_MODEL)


    indexedDf = indexer_model.transform(parsed_values)

    indexedDf = indexedDf.drop("label")
    indexedDf = indexedDf.drop("user")

    features = ["lat", "lon", "alt", "user_index",
                "year", "month", "day", "hour", "minute", "second"]

    featureAssembler = VectorAssembler(
        inputCols=features, outputCol="features")

    assembleredDf = featureAssembler.transform(indexedDf)

    finalData = scaler_model.transform(assembleredDf)

    prediction = classification_model.transform(finalData)

    prediction.printSchema()

    query = prediction.writeStream \
        .foreach(InfluxDBWriter()) \
        .start()
    query.awaitTermination()
```

Aplikacija za predikciju novih podataka

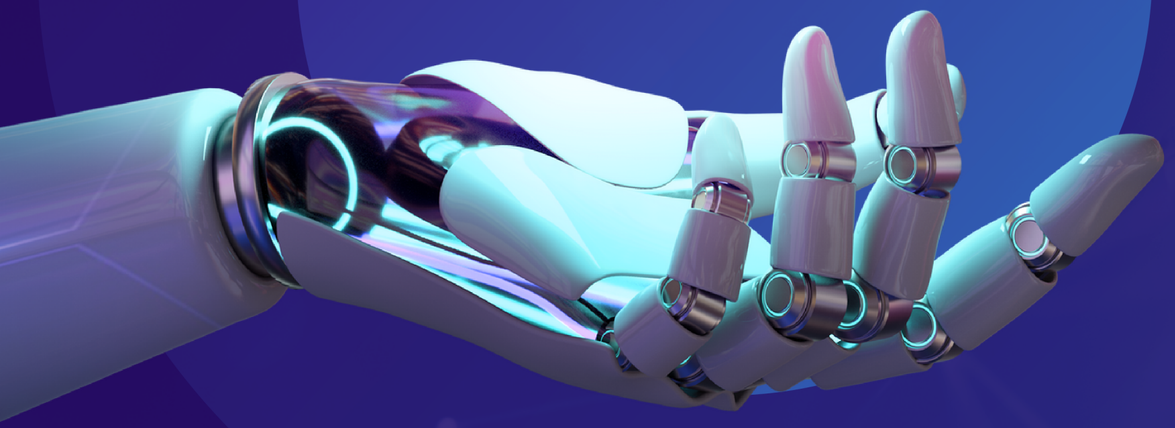
Prikaz podataka u InluxDB

Aplikacija za predikciju novih podataka

Prikaz podataka preko Grafane



SCAN ME



HVALA NA PAŽNJI

ficax.al@elfak.rs