BigData – Projekat 3

Student: Uroš Pešić 1465
Profesor: Dragan Stojanović

Pokretanje

- **Za pokretanje aplikacija potrebno je izvršiti sledeći niz komandi**

  - Za kreiranje docker slika sa job-ovima

  docker build –rm -t streaming-job ./streaming/

  docker build –rm -t batch-job ./batch/

  docker build –rm -t consumer ./consumer/

  - Za pokretanje docker kontejnera

  docker compose up -d (opciono –scale worker=n)

  init.sh (skripta za kreiranje foldera na hdfs-u i ostala inicijalizaciju spark-mastera)

Spark Batch – treniranje modela regresije i klasterizacije nad offline podacima

- Batch job trenira model linearne regresije i klasterizacije koristeći podatke o kretanju vozila koje se nalaze na hdfs-u. Unakrsnom validacijom se vrši podešavanje hiperparametara i dva najbolja modela se smeštaju na hdfs. Kao metrike za performanse su korišćeni *silhouette-score* (za klasterizaciju) i *srednja kvadratna greška – RMSE* (za regresiju).
  Napomena: Trenira se više modela klasterizacije, za različite vremenske periode – broj vremenskih perioda zavisi od ulaznog parametra aplikacije I treniranje se vrši samo nad delom dataset-a koji je vezan za definisan vremenski period.
- Atributi na osnobu kojih se vrši klasifikacija su *vehicle_x* i *vehicle_y*
- Atributi za predikciju zagađenja su *vehicle_fuel vehicle_speed, vehicle_type* i *vehicle_noise*

Klasterizacija

```
# train_test split
train, test = df.randomSplit(weights=[0.8, 0.2], seed=42)

# TASK 1 ---------- K-MEANS CLUSTERING ----------------------

# data preprocessing
assembler = VectorAssembler(inputCols=("vehicle_x", "vehicle_y"), outputCol="features")

# k-means
kmeans = KMeans(seed=42, initSteps=3, predictionCol="cluster_prediction")
pipeline = Pipeline(stages=[assembler, kmeans])

#hyperparameter tuning with cross-validation
paramGrid = ParamGridBuilder() \
        .addGrid(kmeans.k, [4, 5, 6, 7, 8, 9, 10, 11]) \
        .build()

evaluator = ClusteringEvaluator(distanceMeasure="squaredEuclidean", metricName="silhouette", predictionCol="cluster_prediction")

cv = CrossValidator(estimator=pipeline, evaluator=evaluator, numFolds=3, estimatorParamMaps=paramGrid)
print("Clustering...")
model = cv.fit(train)
print(f'Clustering avg. Silhouette score: {model.avgMetrics}')
best_model = model.bestModel

# model evaluation
predictions = best_model.transform(test)
test_silhouette = evaluator.evaluate(predictions)
print(f'Test Silhouette: {test_silhouette}')
predictions[["features", "cluster_prediction"]].show(10, truncate=False)

kmeans_path = f'hdfs://namenode:9000/user/root/models/k_means/model_{time_period}'
best_model.write().overwrite().save(kmeans_path)
```

Linearna regresija

```
# TASK 2 ---------- LINEAR REGRESSION  ----------

train, test = dataset.sample(fraction=0.5, seed=42).randomSplit(weights=[0.8, 0.2], seed=42)

# data preprocessing
indexer = StringIndexer(inputCol="vehicle_type", outputCol="vehicle_type_indexed")
oh_encoder = OneHotEncoder(inputCol=indexer.getOutputCol(), outputCol="vehicle_type_onehot")

regression_cols = ("vehicle_fuel", "vehicle_noise", "vehicle_speed", "vehicle_type_onehot")
target_col = "emission_sum"

assembler = VectorAssembler(inputCols=regression_cols, outputCol="features")
scaler = StandardScaler(inputCol=assembler.getOutputCol(), outputCol="features_scaled")

# linear regression
lreg = LinearRegression(featuresCol="features_scaled", labelCol=target_col, standardization=False, predictionCol="emission_prediction")
pipeline = Pipeline(stages=[indexer, oh_encoder, assembler, scaler, lreg])

# hyperparameter tuning with cross-validation
paramGrid = ParamGridBuilder() \
        .addGrid(lreg.regParam, [0.1, 0.01, 0.001, 0.0001]) \
        .addGrid(lreg.maxIter, [20, 50]) \
        .build()

evaluator = RegressionEvaluator(labelCol=target_col, metricName="rmse", predictionCol="emission_prediction")

cv = CrossValidator(estimator=pipeline, evaluator=evaluator, numFolds=5, estimatorParamMaps=paramGrid)
print("Linear Regression...")
model = cv.fit(train)
print(f'Linear regression average RMSE: {model.avgMetrics}')
best_model = model.bestModel

# model evaluation
predictions = best_model.transform(test)
test_rmse = evaluator.evaluate(predictions)
print(f'Test RMSE: {test_rmse}')
predictions[["features", target_col, "emission_prediction"]].show(10, truncate=False)

lreg_path = f'hdfs://namenode:9000/user/root/models/lreg'
best_model.write().overwrite().save(lreg_path)
```

Batch job - Pokretanje

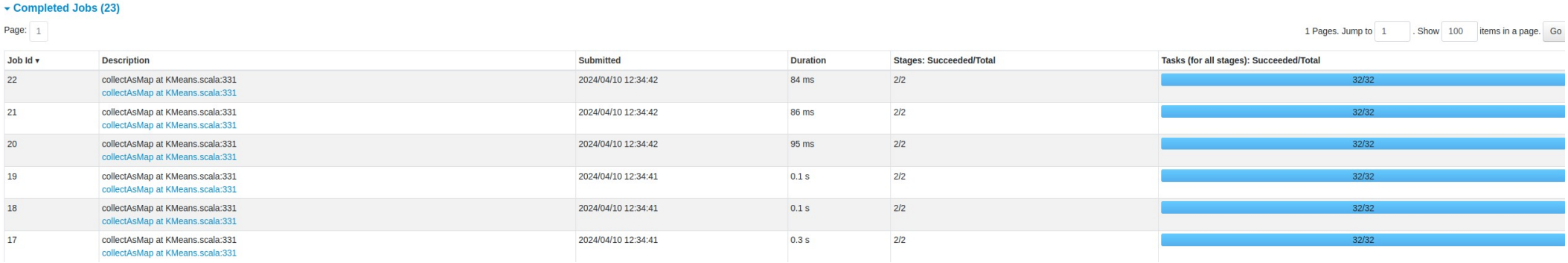
- Za pokretanje batch job-a koristi se komanda

  docker run -d --name batch_job -p 4041:4040 --network mobility_infr -e INPUT_PATH=hdfs://namenode:9000/user/root/input/emission_data_og.csv -e TIME_PERIOD_DURATION=45 batch-job

- Potrebno je navesti dva parametra, preko promenljivih okruženja:

  1) INPUT_PATH – putanja do fajla sa podacima za treniranje

  2) TIME_PERIOD_DURATION – dužina jednog vremenskog perioda za klasterizaciju u minutima



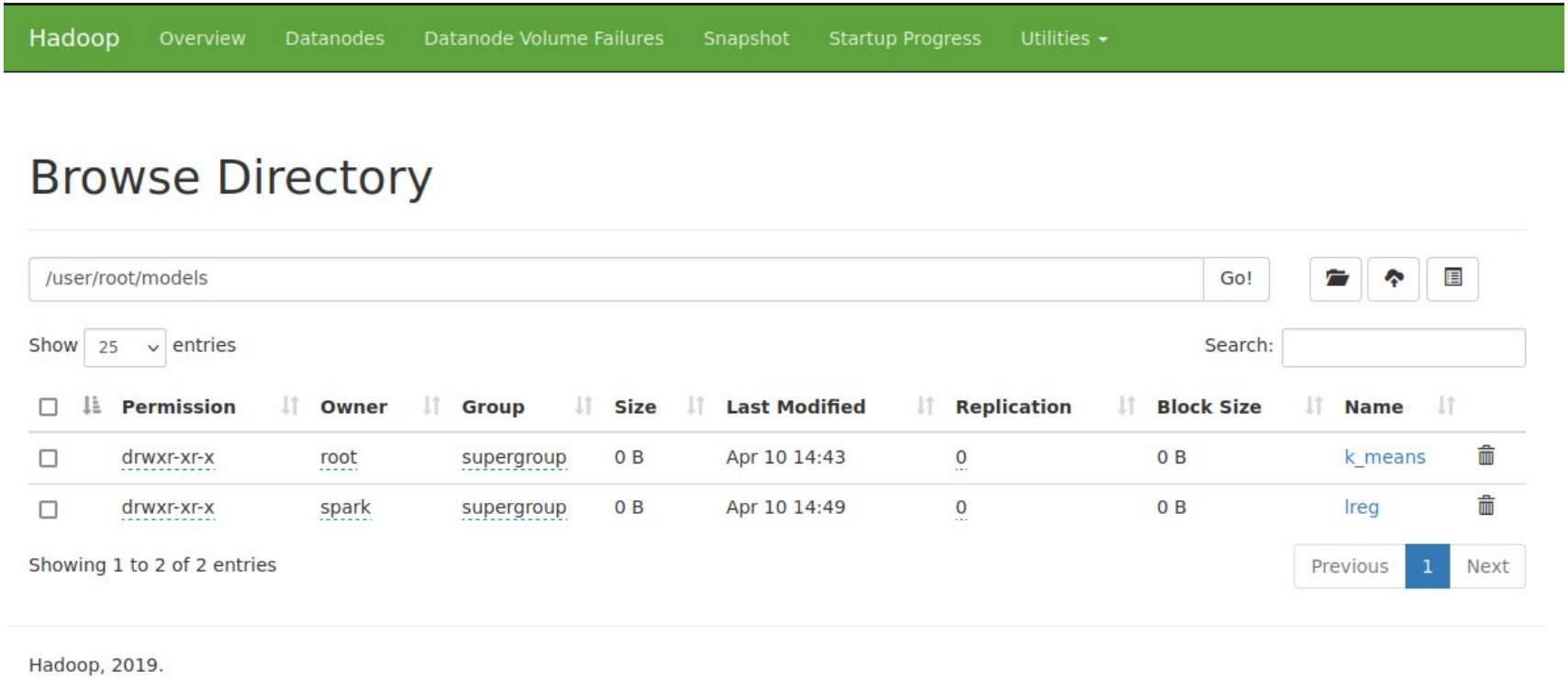
Completed Jobs (23)

Page: 1

1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
22	collectAsMap at KMeans.scala:331 collectAsMap at KMeans.scala:331	2024/04/10 12:34:42	84 ms	2/2	32/32
21	collectAsMap at KMeans.scala:331 collectAsMap at KMeans.scala:331	2024/04/10 12:34:42	86 ms	2/2	32/32
20	collectAsMap at KMeans.scala:331 collectAsMap at KMeans.scala:331	2024/04/10 12:34:42	95 ms	2/2	32/32
19	collectAsMap at KMeans.scala:331 collectAsMap at KMeans.scala:331	2024/04/10 12:34:41	0.1 s	2/2	32/32
18	collectAsMap at KMeans.scala:331 collectAsMap at KMeans.scala:331	2024/04/10 12:34:41	0.1 s	2/2	32/32
17	collectAsMap at KMeans.scala:331 collectAsMap at KMeans.scala:331	2024/04/10 12:34:41	0.3 s	2/2	32/32

Batch job – Rezultati



Batch job – Rezultati

Browse Directory

/user/root/models/k_means Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	spark	supergroup	0 B	Apr 10 14:37	0	0 B	model_0
drwxr-xr-x	spark	supergroup	0 B	Apr 10 14:40	0	0 B	model_1
drwxr-xr-x	spark	supergroup	0 B	Apr 10 14:42	0	0 B	model_2
drwxr-xr-x	spark	supergroup	0 B	Apr 10 14:43	0	0 B	model_3

Showing 1 to 4 of 4 entries Previous 1 Next

Hadoop, 2019.

Streaming job

```
kmeans_pipelines = dict()
for i in range(0, time_period_num):
    model_path = f'/models/k_means/model_{i}'
    kmeans_pipelines[i] = PipelineModel.load(hdfs + model_path)

lreg_pipeline = PipelineModel.load(hdfs + '/models/lreg')

data = spark.readStream \
        .format("kafka") \
        .option("kafka.bootstrap.servers", kafka_broker) \
        .option("startingOffsets", "earliest") \
        .option("subscribe", source_topic) \
        .load()

# deserialize, flatten...

chk_loc = chk_loc = f'{hdfs}/chk/raw'
flat_df.withColumn("value", F.encode(F.to_json(F.struct(F.col("*"))), "iso-8859-1")) \
                .writeStream \
                .queryName("raw_data_query") \
                .format("kafka") \
                .option("kafka.bootstrap.servers", kafka_broker) \
                .option("topic", "raw_data") \
                .option("checkpointLocation", chk_loc) \
                .outputMode("append") \
                .start()

flat_df = flat_df.withColumn("time_period", (flat_df["timestep"] / time_period_size_sim).cast("int"))

cols = ["event_time", "offset", "time_period", "vehicle_id", "vehicle_x", "vehicle_y", "cluster_prediction", "vehicle_fuel", "vehicle_noise", "vehicle_type", "vehicle_speed"]

for i in range(0, time_period_num):
    df_time_period = flat_df.filter(flat_df["time_period"] == i)

    cluster_predictions = kmeans_pipelines[i].transform(df_time_period)
    cluster_predictions = cluster_predictions.select(cols)
    emission_predictions = lreg_pipeline.transform(cluster_predictions)

    lreg_udf = F.udf(lambda x: x if x >= 0 else abs(x), FloatType())
    predictions = emission_predictions.withColumn("emission_prediction", lreg_udf(emission_predictions["emission_prediction"])).select(cols + ["emission_prediction"])

    chk_loc = f'{hdfs}/chk/tp{i}'
    predictions.withColumn("value", F.encode(F.to_json(F.struct(F.col("*"))), "iso-8859-1")) \
                .writeStream \
                .queryName(f'time_period_{i}') \
                .format("kafka") \
                .option("kafka.bootstrap.servers", kafka_broker) \
                .option("topic", sink_topic) \
                .option("checkpointLocation", chk_loc) \
                .outputMode("append") \
                .start()
```

- Podaci koji stignu na kafka topic “traffic_data” se deserializuju, a zatim šalju na dva nova topica:

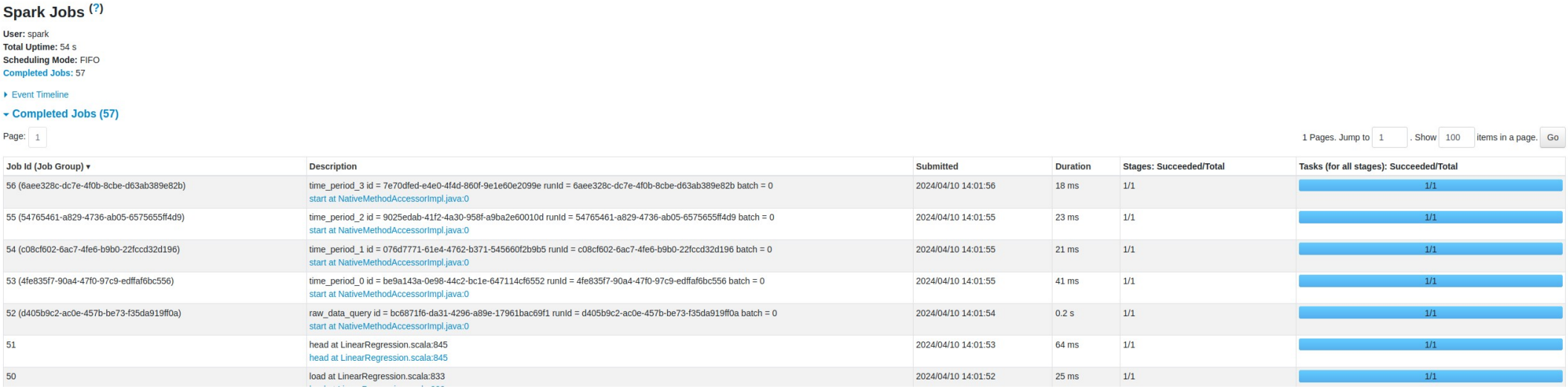
  1) *Raw_data* – “sirovi” podaci se šalju na ovaj topic iz koga čita jedan Consumer I upisuje u raw.csv fajl. Ovi podaci se mogu zatim koristiti za sledeće treniranje modela.

  2) *Predictions* – na ovaj topic se šalju predikcije zagađenja i klastera u kojem se vozilo nalazi. Sa ovog topica čita drugi consumer koji ove podatke upisuje u predictions.csv fajl za potrebe dalje vizuelizacije.

Streaming job - Pokretanje

- Za pokretanje streaming jop-a koristimo sledeću komandu

  docker run -d --name streaming_job -p 4040:4040 --network mobility_infr -e TIME_PERIOD_DURATION=45 -e NUM_PERIODS=4 streaming-job



Spark Jobs (?)

User: spark
Total Uptime: 54 s
Scheduling Mode: FIFO
Completed Jobs: 57

▸ Event Timeline

▾ **Completed Jobs (57)**

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id (Job Group) ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
56 (6aee328c-dc7e-4f0b-8cbe-d63ab389e82b)	time_period_3 id = 7e70dfed-e4e0-4f4d-860f-9e1e60e2099e runId = 6aee328c-dc7e-4f0b-8cbe-d63ab389e82b batch = 0 start at NativeMethodAccessorImpl.java:0	2024/04/10 14:01:56	18 ms	1/1	1/1
55 (54765461-a829-4736-ab05-6575655ff4d9)	time_period_2 id = 9025edab-41f2-4a30-958f-a9ba2e60010d runId = 54765461-a829-4736-ab05-6575655ff4d9 batch = 0 start at NativeMethodAccessorImpl.java:0	2024/04/10 14:01:55	23 ms	1/1	1/1
54 (c08cf602-6ac7-4fe6-b9b0-22fccd32d196)	time_period_1 id = 076d7771-61e4-4762-b371-545660f2b9b5 runId = c08cf602-6ac7-4fe6-b9b0-22fccd32d196 batch = 0 start at NativeMethodAccessorImpl.java:0	2024/04/10 14:01:55	21 ms	1/1	1/1
53 (4fe835f7-90a4-47f0-97c9-edffaf6bc556)	time_period_0 id = be9a143a-0e98-44c2-bc1e-647114cf6552 runId = 4fe835f7-90a4-47f0-97c9-edffaf6bc556 batch = 0 start at NativeMethodAccessorImpl.java:0	2024/04/10 14:01:55	41 ms	1/1	1/1
52 (d405b9c2-ac0e-457b-be73-f35da919ff0a)	raw_data_query id = bc6871f6-da31-4296-a89e-17961bac69f1 runId = d405b9c2-ac0e-457b-be73-f35da919ff0a batch = 0 start at NativeMethodAccessorImpl.java:0	2024/04/10 14:01:54	0.2 s	1/1	1/1
51	head at LinearRegression.scala:845 head at LinearRegression.scala:845	2024/04/10 14:01:53	64 ms	1/1	1/1
50	load at LinearRegression.scala:833	2024/04/10 14:01:52	25 ms	1/1	1/1

Streaming job - Pokretanje

- Napomena: Parametar TIME_PERIODS_DURATION mora da odgovara vrednosti ovog parametra koja je korišćena kod batch job-a, da bi rezulati imali smisla. Takođe parametar NUM_PERIODS odgovara broju perioda za koje su definisani modeli klasterizacije, tj. broju ovih modela.
- Pomoćnu aplikaciju za generisanje toka podataka pokrećemo komandom

  java -jar SUMODataProducer.jar input traffic-data

Vizuelizacija

- Consumer kontejneri su vezani za lokalni fajl sistem korišćenjem docker volume-a I tu upisuju svoje rezultate.
- Napomena: U realnom sistemu bi I obi fajlovi trebali da budu smešteni na hdfs, što ovde nije urađeno zbog jednostavnosti realizacije.