Miloš Veljanovski 1559

Projekat 3 – Sistemi za analizu velike količine podataka

Korišćeni izvorni podaci

- Iskorišćen je set podataka o vožnji biciklama iz „oblasti zaliva" tj. San Francisko zaliva (uključujući Palo Alto i San Hoze)
- Dataset sadrži sledeće kolene podataka:
  - Vreme i datum početka i završetka vožnje
  - Podaci o početkoj stanici (ID, ime, geografska širina i dužina)
  - Podaci o završnoj stanici (ID, ime, geografska širina i dužina)
  - ID vožnje
  - Tip korisnika (vozača)
- Korišćena je Lite verzija dataseta sa podacima za septembar mesec 2022. godine

Opis projekta 3

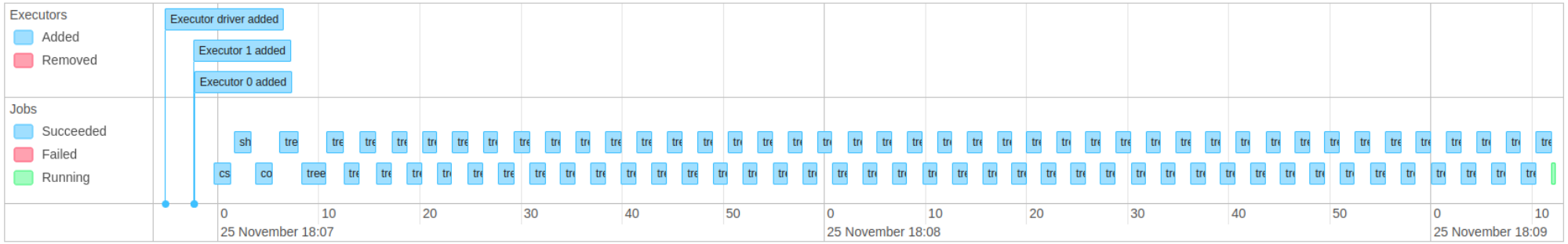
- PySpark aplikacije za treniranje i klasifikaciju (mašinsko učenje)
- Model se trenira iz postojećeg dataset-a i dobija se na osnovu feature-a: latituda i longituda početne stanice
- Identičan Kafka Producer iz Projekta 2 je iskorišćen za slanje redova na Kafka topic
- Rezultati predikcije upisuju se u bazi podataka InfluxDb
- Vizuelizacija podataka je postignuta kroz Grafana web aplikaciju

Treniranje na Spark clasteru

Spark Jobs (?)

User: root
Total Uptime: 2.3 min
Scheduling Mode: FIFO
Active Jobs: 1
Completed Jobs: 86

▾ Event Timeline
☐ Enable zooming



▾ Active Jobs (1)

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
86	treeAggregate at RDDLossFunction.scala:61 treeAggregate at RDDLossFunction.scala:61 (kill)	2023/11/25 18:09:11	0.1 s	0/1	0/1

Značajni delovi koda aplikacije za treniranje

app.py

```python
if __name__ == '__main__':
    load_dotenv()

    spark = SparkSession.builder.appName('PySparkApp - Training').getOrCreate()
    spark.sparkContext.setLogLevel("ERROR")

    HDFS_DATA = os.getenv('HDFS')
    MODEL = os.getenv('MODEL_LOCATION')

    dataFrame = spark.read.csv(HDFS_DATA, header=True)
    dataFrame = dataFrame.withColumn( colName: "duration", F.unix_timestamp("ended_at") - F.unix_timestamp("started_at"))
    dataFrame.show()

    columns = ['start_lat', 'start_lng']

    for column in columns:
        dataFrame = dataFrame.withColumn(column, F.col(column).cast(FloatType()))

    vectorAssembler = VectorAssembler().setInputCols(columns).setOutputCol('features').setHandleInvalid('skip')

    assembled = vectorAssembler.transform(dataFrame)

    stringIndexer = StringIndexer().setInputCol('duration').setOutputCol('label')
    indexedDataFrame = stringIndexer.fit(assembled).transform(assembled)

    train_split, test_split = indexedDataFrame.randomSplit([0.8, 0.2])

    print("Starting training")

    regressionModel = LogisticRegression(maxIter=100, regParam=0.02, elasticNetParam=0.8)

    pipeline = Pipeline(stages=[regressionModel])
    regressionModelPipe = pipeline.fit(train_split)

    prediction = regressionModelPipe.transform(test_split)

    evaluator = BinaryClassificationEvaluator(labelCol='label', rawPredictionCol='prediction',
                                              metricName='areaUnderPR')
    print("Starting evaluation")
    accuracy = evaluator.evaluate(prediction)

    print('Accuracy\'s value for logistic regression model is ' + str(accuracy))

    regressionModelPipe.write().overwrite().save(MODEL)
```

main

Delovi koda aplikacije za klasifikaciju

```
MODEL = os.getenv('MODEL')
print("Model Path:", MODEL)
influxDBConnection = connect_to_influx()

spark = SparkSession.builder \
    .appName("bigdata-project3-ml-classification") \
    .config("spark.sql.warehouse.dir", MODEL) \
    .config("spark.master", "local[*]") \
    .getOrCreate()

model = PipelineModel.load(MODEL)
spark.sparkContext.setLogLevel("INFO")
sampleDataframe = (
    spark.readStream.format("kafka")
    .option( key: "kafka.bootstrap.servers", kafka)
    .option( key: "subscribe", topic)
    .option( key: "startingOffsets",  value: "earliest")
    .load()
).selectExpr( *expr: "CAST(value as STRING)", "timestamp").select(
    from_json(col("value"), dataSchema).alias("sample"), "timestamp"
).select("sample.*")

sampleDataframe.writeStream \
    .foreachBatch(lambda df, epoch_id: analyze(df, epoch_id, model)) \
    .outputMode("update") \
    .trigger(processingTime="10 seconds") \
    .start().awaitTermination()
```

Upis podataka u bazu

```
1 usage
def connect_to_influx():
    return InfluxDBClient(db_host, db_port, db_user, db_password, db_name)


1 usage
def write_to_influx(count, predictions, accuracy):
    timestamp = datetime.utcnow().strftime('%Y-%m-%dT%H:%M:%SZ')
    measurement_data = [
        {
            "measurement": topic,
            "time": timestamp,
            "fields": {
                "number of rows": count,
                "predictions": count,
                "correct predictions": accuracy
            }
        }
    ]
    print(measurement_data)
    influxDBConnection.write_points(measurement_data, time_precision='ms')
    print("Count " + str(count))
    print("Accuracy " + str(accuracy))
```

Analiza

```
1 usage
def analyze(df, epoch, model):
    print("Epoch " + str(epoch))
    columns = ['start_lat', 'start_lng']

    for column in columns:
        df = df.withColumn(column, F.col(column).cast(FloatType()))

    vector_assembler = VectorAssembler().setInputCols(columns).setOutputCol('features').setHandleInvalid('skip')
    assembled = vector_assembler.transform(df)
    string_indexer = StringIndexer().setInputCol('member_casual').setOutputCol('label')
    indexed_data_frame = string_indexer.fit(assembled).transform(assembled)

    prediction = model.transform(indexed_data_frame)

    prediction.select('prediction', 'label')
    # prediction.show(truncate=False)

    predictions_made = prediction.count()
    correct_number = float(prediction.filter(prediction['label'] == prediction['prediction']).count())

    write_to_influx(df.count(), predictions_made, correct_number)
```

Vizuelizacija u Grafana web aplikaciji