Laboratorijska vežba 5 – Apache Spark

Cilj vežbe: Razvoj Apache Spark aplikacije

Potrebni alati

- Apache Spark klijent
- Apache Hadoop

Ovi alati podrazumevaju da je prethodno instalirana Java i podešena promenljiva okruženja JAVA_HOME.

Instalacija Apache Spark-a

- Preuzeti Apache Spark sa linka (treba odabrati verziju 3.2.4 i Pre-built for Apache Hadoop 2.7 tip paketa)
- Ekstraktovati arhivu na pogodnu lokaciju (npr. **C:\Spark**)
- Definisati promenljivu okruženja SPARK_HOME tako da ukazuje na direktorijum u kome se nalazi Spark, npr:
  C:\Spark

Apache Hadoop instalacija

Za potrebe ove laboratorijske vežbe dovoljno je instalirati **winutils.exe** za odgovarajuću verziju Hadoop-a.

- Preuzeti **winutils.exe**
- Kreirati **Hadoop** folder (npr. **C:\Hadoop**), a u njemu **bin** folder.
- Smestiti **winutils.exe** u bin folder
- Kreirati promenljivu okruženja HADOOP_HOME tako da ukazuje na lokaciju gde je ekstraktovan Hadoop (npr. **C:\Hadoop**):
- U Path promenljivu okruženja dodati putanje do Hadoop\bin foldera, Spark\bin foldera i Java bin foldera

Ostala podešavanja

- Instalirati PySpark: pip install pyspark==3.2.2

Apache Spark

Apache Spark je sistem za obradu podataka na klasteru. Aplikacija koja koristi ovaj sistem se sastoji od glavnog programa, menadžera klastera i čvorova-radnika. Menadžer klastera vrši alokaciju resursa na čvorovima u klasteru i pokreće izvršenja na čvorovima. Kada se sistem pokrene, Spark šalje kod aplikacije na čvorove, a zatim i konkretne zadatke i podatke koje treba obraditi. Uz sistem se mogu koristiti i biblioteke koje se pribavljaju zajedno sa njim: **SparkSQL**, **Spark Streaming**, **MLlib** i **GraphX**.

Osnovna apstrakcija podataka koju Spark koristi jeste *resilient distributed dataset* (RDD). RDD je kolekcija elemenata koji se mogu paralelno obrađivati. Kreiraju se paralelizacijom postojećih elemenata (npr. paralelizacijom listi) ili iz nekog drugog izvora (npr. HDFS, Cassandra, Hbase, lokalni fajl sistem).

Kolekcije je moguće obraditi transformacijama (npr. mapiranje, filtirianje, unija, presek i dr.) koje kreiraju nove podatke (novi RDD) ili akcijama (npr. reduce(), collect()) koje izvršavaju neko izračunavanje nad podacima.Transformacijama se definiše šta će se dogoditi sa podacima kada obrade počne, dok pozivi akcija započinju obradu podataka.

Za definisanje zadataka koji će se izvršiti nad čvorovima potrebno je definisati funkcije (kao lambda izraze ili implementacijom *Function* interfejsa u odgovarajućoj klasi).

Tok Apache Spark programa podrazumeva:

- Kreiranje sesije – definisanje naziva aplikacije, master čvora i drugih konfiguracionih parametara,
- Kreiranje SparkContext promenljive koja predstavlja ulaznu tačku za korišćenje različitih Spark funkcionalnosti – konekcija prema Spark klasteru, kreiranje RDD-ova itd,
- Kreiranje transformacija i akcija nad podacima.

Zadatak 1

Kreirati Apache Spark program koji filtrira brojeve manji od 10. Pronalazi kvadrate filtriranih brojeva i u konzoli prikazuje sumu dobivenih kvadrata.

```
from pyspark import SparkContext, SparkConf

conf = SparkConf().setAppName("test").setMaster("local")
sc = SparkContext(conf=conf)

data = [1, 2, 3, 4, 50, 61, 72, 8, 9, 19, 31, 42, 53, 6, 7, 23]
#data = [(1.5, 2.2, 3.1, 4.2, 50.1, 61.3, 72.8, 8.2, 9.5, 19.6, 31.7, 42.8,
53.3, 6.6, 7.4, 23.1]


rdd = sc.parallelize(data)


filteredRDD = rdd.filter(lambda x: x > 10)
transformedRDD = filteredRDD.map(lambda x: x*x)
sum = transformedRDD.reduce(lambda x, y: x+y)

print("Broj elemenata:", sum)
```

Zadatak 2

Kreirati Apache Spark program koji korišćenjem RDD-a broji i prikazuje koliko se puta svaka od reči nalazi u tekstu.

Rešenje 1 – ukoliko je tekst zadat kao lista rečenica

```
from pyspark.sql import SparkSession


spark = SparkSession\
        .builder\
        .appName("brojanje_reci_1")\
        .getOrCreate()

data = ["RDD Like Demo",     \
    "Abstract RDD Like Demo", \
    "RDD Demo", \
    "Double RDD Demo", \
    "Pair RDD Demo", \
    "Hadoop RDD Demo", \
    "New Hadoop RDD Demo" \
    ]

rdd = spark.sparkContext.parallelize(data)
word_counts = rdd.flatMap(lambda line: line.split()) \
                    .map(lambda word: (word, 1)) \
                    .reduceByKey(lambda a, b: a + b)

output = word_counts.collect()
for (word, count) in output:
    print("%s: %i" % (word, count))

sc.stop()
```

Rešenje 2 – ukoliko se tekst čita iz fajla

```
spark = SparkSession\
        .builder\
        .appName("brojanje_reci_2")\
        .getOrCreate()

lines = spark.read.text("input.txt").rdd.map(lambda r: r[0])
word_counts = lines.flatMap(lambda x: x.split(' ')) \
                  .map(lambda x: (x, 1)) \
                  .reduceByKey(lambda a, b: a + b)


word_counts.saveAsTextFile("output.txt")
sc.stop()
```

Zadatak 3

Kreirati Apache Spark program za klasifikaciju koji na osnovu zadatih atributa koji opisuju profil pacijanta određuje da li će rezultati testa na dijabetes biti pozitivni ili negativni. Profil pacijanta je opisan sa ukupno 8 numeričnih atributa, deveti atribut predstavlja klasu (0 – ako nema dijabetes, 1 – ako ima dijabetes).

Rešenje

```
from pyspark.sql import SparkSession
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.mllib.evaluation import MulticlassMetrics

spark = SparkSession.builder.appName('PrimerKlasifikacije').getOrCreate()

data = spark.read.csv('diabetes.csv', header=True, inferSchema=True)

assembler = VectorAssembler(inputCols=data.columns[:-1],
outputCol="features")
data = assembler.transform(data).select("features", "label")
data.show(5)

splits = data.randomSplit([0.7, 0.3], 1234)
train = splits[0]
test = splits[1]

nb = NaiveBayes(smoothing=1.0, modelType="multinomial")

model = nb.fit(train)

predictions = model.transform(test)

evaluator = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="accuracy")

accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))

predictionAndLabels = predictions.select("prediction",
"label").rdd.map(tuple)

metrics = MulticlassMetrics(predictionAndLabels)
confusionMatrix = metrics.confusionMatrix()
print("Confusion Matrix:")
print(confusionMatrix)

spark.stop()
```