PaytmChallenge

PayTM Challenge

Preparation

```
val spark = sparkSession
```

```
Took: 1 second 906 milliseconds, at 2017-3-4 21:29
```

Specify datasource path

```
val home = sys.env("TRAINING_HOME")
val datadir = s"$home/data/"
val logfile = "2015_07_22_mktplace_shop_web_log_sample.log"
```

```
Took: 1 second 770 milliseconds, at 2017-3-4 21:29
```

Load text into RDD

```
val rawRDD = sc.textFile(datadir + logfile)
rawRDD.count
```

```
1158500                     Took: 18 seconds 282 milliseconds, at 2017-3-4 21:29
```

Define function to convert ISO8601 into Epoch time

```
import java.time.Instant

def timeStrToLong(timeString:String):Long =
  Instant.parse(timeString).toEpochMilli();
```

```
Took: 1 second 423 milliseconds, at 2017-3-4 21:29
```

```
def v(df:org.apache.spark.sql.DataFrame) = DataFrameWidget.table(df, 25)
```

```
Took: 1 second 563 milliseconds, at 2017-3-4 21:30
```

Define regexp to extract time, ip, url from log lines

```
val lineRE = """(\S*)\s\S*\s(\S*):.*(http\S*).*""".r
```

```
Took: 1 second 808 milliseconds, at 2017-3-4 21:30
```

Case class to store parsed data

```
case class LogRecord(timestamp:Long, ip:String, url:String)
```

```
Took: 1 second 311 milliseconds, at 2017-3-4 21:30
```

Parse lines with regular expression

```
val parsedRDD = rawRDD.map {
  case lineRE(time,ip,url) =>
    LogRecord(timeStrToLong(time), ip, url)
  case badLine =>
    println(s"Unexpected line: $badLine")
    LogRecord(0L, "", "")
}

parsedRDD.cache()
```

```
MapPartitionsRDD[2] at map at <console>:80        Took: 2 seconds 87 milliseconds, at 2017-3-4 21:30
```

Check parsing results

```
parsedRDD.take(3).foreach(println)
```

```
Took: 41 seconds 843 milliseconds, at 2017-3-4 21:30
```

Create DataFrame for sessionization

```
val logDF = parsedRDD.toDF.cache()
logDF.createOrReplaceTempView("log")
```

```
Took: 4 seconds 550 milliseconds, at 2017-3-4 21:30
```

Define Spark UDFs we will use in sessionization

```
import org.apache.spark.sql.functions.udf

//Define Spark UDFs

def sessionFlag(timeout:Long)=udf((duration:Long) => if (duration > timeout) 1 else 0)

def sessionName = udf((ip:String, session:Long) => ip + "_" + session)
```

Took: 1 second 637 milliseconds, at 2017-3-4 21:30

1. Sessionize log data

We will use window functions of Spark to sessionize data

```
import org.apache.spark.sql.expressions.Window

val timeout = 900000L //15 minutes session timeout

//Define window settings
val window = Window.partitionBy("ip").orderBy("timestamp")

val lagCol = lag(col("timestamp"), 1).over(window)

val sessionDF = logDF.withColumn("prevtime", lagCol)
  .withColumn("ptime",when($"prevtime".isNull, $"timestamp").otherwise(lagCol))
  .withColumn("duration",$"timestamp"-$"ptime")
  .withColumn("newsession", sessionFlag(timeout)($"duration"))
  .withColumn("session", sum($"newsession").over(window))
  .withColumn("session_id",sessionName($"ip",$"session"))
  .cache()

sessionDF.createOrReplaceTempView("session")
sessionDF.count()
```

```
1158500
```

Took: 3 minutes 34 seconds 702 milliseconds, at 2017-3-4 21:34

Sanity check of sessionization result

```
v(sessionDF.filter("""ip in ("115.114.78.170","103.15.250.10")"""))
```

timestamp	ip	url
1437555630170	"115.114.78.170"	"https://paytm.com:443/shop/wallet/txnhistory?page_size=10&page_number=0&channel=web&version=2"
1437561157336	"115.114.78.170"	"https://paytm.com:443/shop?utm_source=affiliate&utm_medium=promocodeclub&utm_campaign=promoc generic&utm_term=pccptm"
1437561157916	"115.114.78.170"	"https://paytm.com:443/shop/v1/frequentorders?channel=web&version=2"
1437561158234	"115.114.78.170"	"https://paytm.com:443/shop/cart?channel=web&version=2"
1437561158564	"115.114.78.170"	"https://paytm.com:443/shop?utm_source=affiliate&utm_medium=promocodeclub&utm_campaign=promoc generic&utm_term=pccptm"
1437561160154	"115.114.78.170"	"https://paytm.com:443/"
1437561163085	"115.114.78.170"	"https://paytm.com:443/shop/v1/frequentorders?channel=web&version=2"
1437561163409	"115.114.78.170"	"https://paytm.com:443/shop/cart?channel=web&version=2"
1437561170629	"115.114.78.170"	"https://paytm.com:443/shop/wallet/txnhistory?page_size=10&page_number=0&channel=web&version=2"
1437561170632	"115.114.78.170"	"https://paytm.com:443/shop/wallet/balance?channel=web&version=2"
1437561226555	"115.114.78.170"	"https://paytm.com:443/papi/v1/expresscart/verify"
1437561235425	"115.114.78.170"	"https://paytm.com:443/papi/v1/expresscart/verify"
1437561277310	"115.114.78.170"	"https://paytm.com:443/papi/v1/expresscart/verify"
1437561335848	"115.114.78.170"	"https://paytm.com:443/shop/cart?channel=web&version=2"
1437548333680	"103.15.250.10"	"http://www.paytm.com:80/?utm_source=Affiliates&utm_medium=Paritycube&utm_campaign=Paritycube'
1437555812536	"103.15.250.10"	"https://paytm.com:443/bus-tickets/search/Bangalore/Pondicherry(Puducher 07-22/1"
1437555814264	"103.15.250.10"	"https://paytm.com:443/shop/cart?channel=web&version=2"
1437555816668	"103.15.250.10"	"https://paytm.com:443/favicon.ico"
1437561167528	"103.15.250.10"	"http://www.paytm.com:80/"
1437561468552	"103.15.250.10"	"https://paytm.com:443/shop?utm_source=Affiliates&utm_medium=Payoom&utm_campaign=Payoom"
1437561469306	"103.15.250.10"	"https://paytm.com:443/shop/v1/frequentorders?channel=web&version=2"
1437561469396	"103.15.250.10"	"https://paytm.com:443/shop/cart?channel=web&version=2"
1437561478691	"103.15.250.10"	"https://paytm.com:443/shop/logout?channel=web&version=2"
1437561478900	"103.15.250.10"	"https://paytm.com:443/shop"
1437561479030	"103.15.250.10"	"https://paytm.com:443/shop/cart?channel=web&version=2"

2. Determine the average session time

```
val sessionLenDF = spark.sql("""select max(timestamp)-min(timestamp) as sessionlen, ip, sessio
```

Took: 2 seconds 80 milliseconds, at 2017-3-4 21:34

```
sessionLenDF.limit(10)
```

1

Took: 4 seconds 213 milliseconds, at 2017-3-4 21:34

sessionlen	ip	session_id
8538	"1.186.143.37"	"1.186.143.37_0"
69110	"1.187.164.29"	"1.187.164.29_0"
62308	"1.22.41.76"	"1.22.41.76_0"
13482	"1.23.208.26"	"1.23.208.26_0"
0	"1.23.208.26"	"1.23.208.26_1"
14419	"1.23.36.184"	"1.23.36.184_0"
60599	"1.38.19.8"	"1.38.19.8_0"
191511	"1.38.20.34"	"1.38.20.34_0"
5100	"1.39.13.13"	"1.39.13.13_0"
56469	"1.39.32.249"	"1.39.32.249_0"

Average session time

```
sessionLenDF.select(avg($"sessionlen").alias("avg_session_time"))
```

1

Took: 3 seconds 324 milliseconds, at 2017-3-4 21:34

avg_session_time
100727.51039354735

3. Determine unique URL visits per session

```
val uniqueDF = spark.sql("select count(distinct(url)) uniqueurlcount, session_id from session
```

Took: 989 milliseconds, at 2017-3-4 21:59

```
uniqueDF.sort($"uniqueurlcount".desc).limit(10)
```

1

Took: 3 seconds 961 milliseconds, at 2017-3-4 21:59

uniqueurlcount	session_id
8016	"119.81.61.166_5"
4656	"106.186.23.95_9"
4595	"52.74.219.71_4"
3928	"119.81.61.166_7"
3637	"119.81.61.166_8"
3334	"119.81.61.166_0"
2841	"119.81.61.166_9"
2786	"119.81.61.166_6"
2731	"106.186.23.95_4"
2599	"106.51.132.54_0"

```
Search bots and web crawlers have maximum number of unique URLs per session
```

4. Find the most engaged users, ie the IPs with the longest session times

```
sessionLenDF.select($"ip", $"session_id", $"sessionlen").sort($"sessionlen".desc).limit(10)
```

1

Took: 2 seconds 477 milliseconds, at 2017-3-4 22:0

ip	session_id	sessionlen
"52.74.219.71"	"52.74.219.71_4"	2069162
"119.81.61.166"	"119.81.61.166_4"	2068849
"106.186.23.95"	"106.186.23.95_4"	2068756
"125.19.44.66"	"125.19.44.66_4"	2068713
"125.20.39.66"	"125.20.39.66_3"	2068320
"192.8.190.10"	"192.8.190.10_2"	2067235
"54.251.151.39"	"54.251.151.39_4"	2067023
"180.211.69.209"	"180.211.69.209_3"	2066961
"180.179.213.70"	"180.179.213.70_4"	2065638
"203.189.176.14"	"203.189.176.14_4"	2065594

Most engaged users are search bots and web crowlers

Took: 4 seconds 402 milliseconds, at 2017-3-4 21:49

Build: | **buildTime**-*Tue Jan 10 21:39:02 UTC 2017* | **formattedShaVersion**-*0.8.0-SNAPSHOT-136c6444bb4a40cd9f42ed2a7d203fc0ef93057d* | **sbtVersion**-*0.13.8* | **scalaVersion**-*2.11.8* | **sparkNotebookVersion**-*0.8.0-SNAPSHOT* | **hadoopVersion**-*2.7.2* | **jets3tVersion**-*0.7.1* | **jlineDef**-*(jline,2.12)* | **sparkVersion**-*2.1.0* | **withHive**-*false* |.