ex64

August 20, 2022

[ ]:
```python
from pyspark.streaming import StreamingContext
```

[ ]:
```python
historicalInputFile = "data/Ex64/data/historicalData.txt"
```

[ ]:
```python
# Read the historical data and compute the maximum and minimum price for each␣
 ↪stock
# Non-streaming RDD
historicalDataRDD = sc.textFile(historicalInputFile)
```

[ ]:
```python
# Return one pair (stockId, (price, price) )  for each input record
def extractStockIdPricePrice(line):
    fields = line.split(",")

    stockId = fields[1]
    price = fields[2]

    return (stockId, (float(price), float(price)) )




stockIdPriceHistoricalRDD = historicalDataRDD.map(extractStockIdPricePrice)
```

[ ]:
```python
# Compute max and min for each stockId based on the historical data
stockIdPriceHistoricalMaxMinRDD = stockIdPriceHistoricalRDD\
.reduceByKey(lambda v1, v2: ( max(v1[0],v2[0]), min(v1[1],v2[1]) ) ).cache()
```

[ ]:
```python
# Create a Spark Streaming Context object
#ssc = StreamingContext(sc, 60)
ssc = StreamingContext(sc, 10)
```

[ ]:
```python
# Create a (Receiver) DStream that will connect to localhost:9999
pricesDStream = ssc.socketTextStream("localhost", 9999)
```

[ ]:
```python
# Join on the stockid each input record of the input stream with the
# content of stockIdPriceHistoricalMaxMinRDD to retrieve
# the historical maximum-minimum range of the stock
```

```
[ ]: # Return one pair (stockId,price) for each input record
     stockIdPriceDStream = pricesDStream.map(lambda record: ( record.split(",")[1] ,␣
      ↪float(record.split(",")[2])) )
```

```
[ ]: # Join the RDD associated with the content of the current batch and
     # the non-streaming RDD stockIdPriceHistoricalMaxMinRDD
     stockIdPriceMaxMinDStream = stockIdPriceDStream\
     .transform(lambda batchRDD: batchRDD.join(stockIdPriceHistoricalMaxMinRDD))
```

```
[ ]: # Select only lines with price > maximum historical price
     # or price < minimum historical price
     def anomalyValue(pair):
         currentPrice = pair[1][0]
         stockHistoricalMaxPrice = pair[1][1][0]
         stockHistoricalMinPrice = pair[1][1][1]

         if currentPrice>stockHistoricalMaxPrice or␣
      ↪currentPrice<stockHistoricalMinPrice:
             return True
         else:
             return False



     selectedStockPricesDStream = stockIdPriceMaxMinDStream.filter(anomalyValue)
```

```
[ ]: # Retrieve only the stockIDs and apply distinct to remove duplicates
     # keys and distinct are not available for DStreams.
     # transform must be used
     selectStockIdsDStream = selectedStockPricesDStream\
     .transform(lambda batchRDD: batchRDD.keys().distinct())
```

```
[ ]: selectStockIdsDStream.pprint()
```

```
[ ]: #Start the computation
     ssc.start()
```

```
[ ]: # Run this application for 90 seconds
     ssc.awaitTerminationOrTimeout(90)
     ssc.stop(stopSparkContext=False)
```

```
[ ]:
```