

ex64v2

August 20, 2022

```
[ ]: # Second version. This version is more efficient than the previous one
# because the amount of joined data is reduced.
from pyspark.streaming import StreamingContext

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

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

[ ]: # 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)

[ ]: # 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()

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

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

[ ]: # Compute max and min for each stockId of each input batch
    stockIdPriceDStream = pricesDStream.map(extractStockIdPricePrice)\
        .reduceByKey(lambda v1, v2: ( max(v1[0],v2[0]), min(v1[1],v2[1]) ) )
```

```
[ ]: # Join stockIdPriceDStream with stockIdPriceHistoricalMaxMinRDD
# 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 stocks with stream max price > maximum historical price
# or stream min price < minimum historical price
def anomalyValue(pair):
    stockBatchMaxPrice = pair[1][0][0]
    stockBatchMinPrice = pair[1][0][1]

    stockHistoricalMaxPrice = pair[1][1][0]
    stockHistoricalMinPrice = pair[1][1][1]

    if stockBatchMaxPrice>stockHistoricalMaxPrice or
↪stockBatchMinPrice<stockHistoricalMinPrice:
        return True
    else:
        return False

selectedStockPricesDStream = stockIdPriceMaxMinDStream\
.filter(anomalyValue)
```

```
[ ]: # Retrieve only the stockIDs of the selected stocks
# keys is not available for DStreams.
# transform must be used or map
selectStockIdsDStream = selectedStockPricesDStream\
.transform(lambda batchRDD: batchRDD.keys())
```

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

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

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

```
[ ]: ssc.stop(stopSparkContext=False)
```

```
[ ]:
```