Homework 4

Fubang ZHAO
fubang.zhao@polytechnique.edu

December 7, 2017

Question 1. Algorithm Design based on Anti-monotonicity

1.
   - Yes. If we define $p \subset p'$, we can get obviously$s(p, A_m = 1) > s(p', A_m = 1)$. Therefore, it satisfies the anti-monotone property.
   - With the anti-monotone algorithm, we can easily adapt the Apriori Algorithm for this problem.

2.
   - No.
   - Here is a couter-example:

A_1	A_2	class
0	0	1
0	1	0
1	0	0
1	1	1

   We define a answer set: $A = \{P | P \in 2^{\Sigma} \ \wedge \ r(P) > 2 \ \wedge \ \forall P' \subset P, r(P') < r(P)\}$.

$$r(A_1 = 0, A_2 = 0) = \frac{\frac{1}{4}}{\frac{1}{4}} \times \frac{\frac{3}{4}}{\frac{1}{4}} = 3$$

$$r(A_1 = 0) = \frac{\frac{1}{4}}{\frac{1}{2}} \times \frac{\frac{1}{2}}{\frac{1}{4}} = 1$$

$$r(A_2 = 0) = \frac{\frac{1}{4}}{\frac{1}{2}} \times \frac{\frac{1}{2}}{\frac{1}{4}} = 1$$

   With the result above, we can see that Pattern $(A_1 = 0, A_2 = 0)$ belongs to answer set A. However, its subsets $(A_1 = 0)$ and $(A_2 = 0)$ do not belong to this answer set. Therefore, it does not satisfy the anti-monotone property.

- Without the anti-monotone property, we cannot apply the Apriori Algorithm for this problem, which means we need to search all the possible patterns to get all the qualified patterns.
  So in the worst case, the complexity of the algorithm is 2^n.

3.

- No. Since $P = P_1 \wedge P_2$ and $s(P) = s(P_1)$, we can know that in each pattern P, there is always a pattern P_1. Besides it, there is no other P_1 outside pattern P, which means each P and P_1 has the same class(A_m). So,

$$s(P, A_m = 1) = s(P_1, A_m = 1)$$
$$s(\neg P, A_m = 1) = s(\neg P_1, A_m = 1)$$

  Therefore, we can get $r(P) = r(P_1)$. Obviously, for $\forall P \in \Phi$, P does not belong to the answer set A.

- No.
  - Since $P' \supset P$, we define $P' - P = A$, where A means all the items that belong to P' but not P.
  - Since $P = P_1 \wedge P_2$ and $s(P) = s(P_1)$, we know that in each pattern P, there is always a pattern P_1. Besides it, there is no other P_1 outside pattern P.
  - We define that $P_3 = A \wedge P_1$. Since the pattern P' needs both A and P_1 in the same time, and P_3 only happens in the pattern P'(because P_1 only happens in P, if there is a A in any pattern with P, it will be a pattern P'), so $P' = P_3 \wedge P_2$ and $s(P') = s(P_3)$, which means $P' \in \Phi$.
  - So the pattern P' does not belong to the answer set A.

- Since we have known that, for $\forall P \in \Phi$, P and its all super patterns does not belong to the answer set A, we can rule out all the patterns that do not belong to the answer set A. After we rule out all the wrong patterns, we can get the right ones.

Question 2. Spark Exercise using the DBLP Dataset

A. DBLP Data Analysis using Spark

A1). The code is:

```
val k = 5
val paperauths = sc.textFile("lab_spark_fzhao/dblp/dblp_tsv/paperauths.tsv")
val paperAuthsTokens = paperauths.map(_.split("\t"))
val authCounts = sc.parallelize(paperAuthsTokens.map(tokens => (tokens(1),
    1)).reduceByKey((a, b) => a + b).sortBy(_._2, false).take(k))
val auths = sc.textFile("lab_spark_fzhao/dblp/dblp_tsv/authors.tsv")
val authsTokens = auths.map(_.split("\t")).map(tokens =>
    (tokens(0),tokens(1)))
authsTokens.join(authCounts).map(_._2).map(_._1).collect.foreach(println)
```

A2). The code is:

```
import org.apache.spark.mllib.fpm.FPGrowth
import org.apache.spark.rdd.RDD
val paperauths = sc.textFile("lab_spark_fzhao/dblp/dblp_tsv/paperauths.tsv")
```

```
val paperAuthsTokens = paperauths.map(_.split("\t"))
val dataset: RDD[Array[String]] = paperAuthsTokens.map(tokens =>
    (tokens(0), tokens(1))).groupByKey().mapValues(_.toSet.toArray).map(_._2)
dataset.first()

val fpg = new FPGrowth().setMinSupport(0.0001)
val model = fpg.run(dataset)

model.freqItemsets.map(itemset => itemset.items.mkString("[", ",", "]") +
    ", " + itemset.freq).saveAsTextFile("lab_spark_fzhao/result")
```

B. Click Stream Analysis using Spark Streaming

B1). The code is:

```
"""

Retrieves the visited URLs during the last minute, and update the results
    every
10 seconds, such that the top k of the number of visits within the last
    minute with descending order.

 Usage: aggregation.py <directory>
   <directory> is the directory that Spark Streaming will use to find and
       read new text files (this directory must be in HDFS)

 To run this on Spark cluster, and to monitor directory ‘streamed_data‘ on
     HDFS, run this example
    $ python Q2_B1.py streamed_data <value of K>

 Then run the python script ’start_stream.py’ that creates different text
     files in ‘streamed_data‘ that corresponds to user clicks.
 Note: Replace
     ’hn0-sparkl.koydbiauu5yuthevx5bhblrehb.ax.internal.cloudapp.net’ with
     your primary namenode that you can get from
 calling this command: hdfs getconf -namenodes
 """

import sys
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: Q2_B1.py <directory> <Value of K>")
        exit(-1)
```

```
    sc = SparkContext(appName="myAppAggregation")
    sc.setCheckpointDir("hdfs://hn0-sbdacl.rwcq00v1gfyetmn4ktdcz0g53b.ax.internal.cloudapp
    ssc = StreamingContext(sc, 5)
    lines = ssc.textFileStream(sys.argv[1])
    counts = lines.map(lambda line: line.split(" ")).map(lambda x:
        (x[2],1)).\
    reduceByKeyAndWindow(lambda a,b:a+b, lambda x,y: x-y, 60, 10).\
    transform(lambda rdd: rdd.sortBy(lambda x: x[1], ascending=False))

    counts.pprint(int(sys.argv[2]))

    ssc.start()
    ssc.awaitTermination()
```

B2). The code is:

```
"""
Retrieves the visited URLs during the last minute, and update the results
    every
10 seconds, such that the top k of the number of visits within the last
    minute with descending order.

 Usage: aggregation.py <directory>
   <directory> is the directory that Spark Streaming will use to find and
       read new text files (this directory must be in HDFS)

 To run this on Spark cluster, and to monitor directory ‘streamed_data‘ on
     HDFS, run this example
    $ python Q2_B2.py streamed_data

 Then run the python script ’start_stream.py’ that creates different text
     files in ‘streamed_data‘ that corresponds to user clicks.
 Note: Replace
     ’hn0-sparkl.koydbiauu5yuthevx5bhblrehb.ax.internal.cloudapp.net’ with
     your primary namenode that you can get from
 calling this command: hdfs getconf -namenodes
 """
import sys
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: Q2_B2.py <directory>")
        exit(-1)

    sc = SparkContext(appName="myAppSessionization")
```

```
sc.setCheckpointDir("hdfs://hn0-sbdacl.rwcq00v1gfyetmn4ktdcz0g53b.ax.internal.cloudapp
ssc = StreamingContext(sc, 5)
lines = ssc.textFileStream(sys.argv[1])
counts = lines.map(lambda line: line.split(" ")).map(lambda x: (x[1],
    int(x[0])))

def updateFunction(newVs, state):
    '''
    newVs<list>: new values for each key in this batch
    state<tuple>: (last timestamp,
                   # click which is ready to output,
                   # click to pass to next iteration for continuing to
                       calculate,
                   # session,
                   output or not)

    '''
    if state is None:
        return (max(newVs), 0, len(newVs), 0, 0)
    else:
        if (len(newVs) == 0):
            return state[:4] + (0,) # avoid printing twice
        if min(newVs) - state[0] <= 30:
            return (max(newVs), 0, (len(newVs) + state[2]), state[3], 0)
        else:
            return (max(newVs), state[2], len(newVs), state[3] + 1, 1)

countuup = counts.updateStateByKey(updateFunction).\
           filter(lambda x: x[1][4] == 1).\
           map(lambda x: (x[0], x[1][1], x[1][3]))
countuup.pprint()
ssc.start()
ssc.awaitTermination()
```