Rank Product

Mahmoud Parsian

mahmoud.parsian@yahoo.com

LinedIn: http://www.linkedin.com/in/mahmoudparsian

GitHub: https://github.com/mahmoudparsian/data-algorithms-book

August 2, 2015

1 Introduction

Let $\{A_1, ..., A_k\}$ be a set of (key-value) pairs where keys are unique per dataset. For example, a key can be an item, user, gene and a value can be number of items sold, number of friends for the user , and gene value (such as fold change or test expression). Then the ranked product of $\{A_1, ..., A_k\}$ is computed based on the ranks r_i for key i across all k datasets. Typically ranks are assigned based on the sorted values of datasets. We demonstrate the ranked product with a very simple example by using three data sets:

- Let $A_1 = \{(K_1, 30), (K_2, 60), (K_3, 10), (K_4, 80)\}$,
  we assign the ranks based on the descending sorted values of keys, then $Rank(A_1) = \{(K_1, 3), (K_2, 2), (K_3, 4), (K_4, 1)\}$ since $80 > 60 > 30 > 10$. Note that 1 is the highest rank (assigned to the largest value).
- Let $A_2 = \{(K_1, 90), (K_2, 70), (K_3, 40), (K_4, 50)\}$,
  we assign the ranks based on the descending sorted values of keys, then $Rank(A_2) = \{(K_1, 1), (K_2, 2), (K_3, 4), (K_4, 3)\}$ since $90 > 70 > 50 > 40$.
- Let $A_3 = \{(K_1, 4), (K_2, 8)\}$ we assign the ranks based on the descending sorted values of keys, then $Rank(A_3) = \{(K_1, 2), (K_2, 1)\}$ since $8 > 4$.

Finally, therefore, the rank product of $\{A_1, A_2, A_3\}$ is expressed as:

$$\{(K_1, \sqrt[3]{3 \times 1 \times 2}), (K_2, \sqrt[3]{2 \times 2 \times 1}), (K_3, \sqrt[2]{4 \times 4}), (K_4, \sqrt[2]{1 \times 3})\}$$

2 Application

The rank product (RP) is a biologically motivated test for the detection of differentially expressed genes in replicated micro-array experiments. It is a simple non-parametric statistical method based on ranks of fold changes. In addition to its use in expression profiling, it can be used to combine ranked lists in various application domains, including proteomics, metabolomics, statistical meta-analysis, and general feature selection (Source: Wikipedia). It is shown that Rank products (RP) a simple, yet powerful, new method to detect differentially regulated genes in replicated microarray experiments.

According to [4], "Rank Product, is relatively intuitive method that can be used to find differentially expressed genes as well as being used as a meta analysis method. It does not use any statistics, but rather scores genes on the bases of their ranks in multiple comparisons. The method is useful if you have very few replicates, or if you want to analyze how well the results from two studies agree."

3 Is This a Big Data Problem?

Consider 100 studies each with 100,000 assays and each assay with 60,000 records. This translates to $100 \times 100,000 \times 60,000 = 600,000,000,000$ records, which is a big data.

4 Calculation of the Rank Product

Given n genes and k replicates, let $e_{g,i}$ be the fold change and $r_{g,i}$ the rank of gene g in the i'th replicate.

Compute the rank product (RP) via the geometric mean:

$$RP(g) = \left(\prod_{i=1}^{k} r_{g,i} \right)^{1/k}$$

or

$$RP(g) = \sqrt[k]{\left(\prod_{i=1}^{k} r_{g,i} \right)}$$

5 Formalizing Rank Product

- Let $S = \{S_1, S_2, ..., S_k\}$ be a set of k studies, where $k > 0$ and each study represent a micro-array experiment
- Let $S_i (i = 1, 2, ..., k)$ be a study, which has an arbitrary number of assays identified by $\{A_{i1}, A_{i2}, ...\}$

- Let each assay (can be represented as a text file) be a set of arbitrary number of records in the following format:

  `<gene_id><,><gene_value_as_double_data-type>`

- Let `gene_id` be in $\{g_1, g_2, ..., g_n\}$ (we have n genes).

6 Example

To find rank product of all studies $S = \{S_1, S_2, ..., S_k\}$, first, find the mean of values per gene per study, then sort each study by their gene value and then assign a rank for each gene. For example, Let a single study to have 3 assays with the the following values

Single Study Assays		
Assay-1	Assay-2	Assay-3
g1,1.0	g1,2.0	g1,12.0
g2,3.0	g2,5.0	null
g3,4.0	null	g3,2.0
g4,1.0	g4,3.0	g4,15.0
...	...	...

The first step is to find the mean of values for each gene (per study):

```
g1, 5.0
g2, 4.0
g3, 2.0
g4, 8.0
```

Sorting by value will generate:

```
g4, 8.0
g1, 5.0
g2, 4.0
g3, 2.0
```

Next, we do assign a rank based on sorted values (means) of genes: the result will be (the last column is the ranked value):

```
g4, 8.0, 1
g1, 5.0, 2
g2, 4.0, 3
g3, 2.0, 4
```

The last step will be to find the rank product for each gene per study:

$$S_1 = \{(g_1, r_{11}), (g_2, r_{12}), ...\}$$

$$S_2 = \{(g_1, r_{21}), (g_2, r_{22}), ...\}$$

$$...$$

$$S_k = \{(g_1, r_{k1}), (g_2, r_{k2}), ...\}$$

then Ranked Product of g_j =

$$RP(g_j) = \left(\prod_{i=1}^{k} r_{i,j} \right)^{1/k}$$

or

$$RP(g_j) = \sqrt[k]{\left(\prod_{i=1}^{k} r_{i,j} \right)}$$

7 Spark Solution

The Spark solution will accept k input paths (each path represents a study, which may have any number of assay files). We use the following steps to implement the Rank Product for all genes used in these studies.

1. Find the mean per gene per study (in some situations , you may apply other functions such as median or COPA score)
2. Sort the genes by value per study and then assign rank values (rank values will be 1, 2, ..., N where 1 is assigned to the highest value and N is assigned to the lowest). To sort the dataset by value, we will swap the key with value and then perform the sort. Next we use `JavaPairRDD.zipWithIndex()`, which zips the RDD with its element indices (these indices will be the ranks). Since Spark indices will start from zero, we will add 1 when computing the ranked product.

3. Finally compute the Rank Product per gene for all studies. This can be accomplished by grouping all ranks by the key (we may use `JavaPairRDD.groupByKey()` or `JavaPairRDD.combineByKey()` – note that, in general, `JavaPairRDD.combineByKey()` is more efficient than `JavaPairRDD.groupByKey()`). To understand `groupByKey()` and `combineByKey()` in details, see references [5, 6].

Two solutions are provided using Spark-1.4.0:

- `SparkRankProductUsingGroupByKey` (uses `groupByKey()`)
- `SparkRankProductUsingCombineByKey` (uses `combineByKey()`)

7.1 Input Data Format

Each assay (can be represented as a text file) is a set of arbitrary number of records in the following format:

```
<gene_id><,><gene_value_as_double_data-type>
```

where `gene_id` is a key, which has an associated double data type value.

7.2 Output Data Format

We generate output in the following format:

```
<gene_id><,><ranked-product-among-all-input-data><,><N>

where N = <number-of-values-participated-in-computing-rank-product>
```

8 Sample Run using groupByKey()

Not that the Spark solution using `combineByKey()` is more efficient than `groupByKey()` solution. When using `combineByKey()`, intermediate values are reduced by local workers before being sent for the final reduction, but by using `groupByKey()`, there is no local reduction at all: all values are sent to one location for further processing.

8.1 The Script

```
$ cat ./run_rank_product_using_group_by_key_spark_yarn.sh
# define the installation dir for hadoop
export HADOOP_HOME=/Users/mparsian/zmp/zs/hadoop-2.6.0
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export HADOOP_HOME_WARN_SUPPRESS=true
#
export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.7.0_60.jdk/Contents/Home
```

```
export BOOK_HOME=/Users/mparsian/zmp/github/data-algorithms-book
export LIB=$BOOK_HOME/lib
export SPARK_HOME=/Users/mparsian/spark-1.4.0
export SPARK_JAR=$BOOK_HOME/lib/spark-assembly-1.4.0-hadoop2.6.0.jar
export APP_JAR=$BOOK_HOME/dist/data_algorithms_book.jar
# defines some environment for hadoop
source $HADOOP_CONF_DIR/hadoop-env.sh
#
# build all other dependent jars in OTHER_JARS
JARS=`find $LIB -name '*.jar'  ! -name '*spark-assembly-1.4.0-hadoop2.6.0.jar' `
OTHER_JARS=""
for J in $JARS ; do
   OTHER_JARS=$J,$OTHER_JARS
done
#
# define input/output for Hadoop/HDFS
#// args[0] = output path
#// args[1] = number of studies (K)
#// args[2] =   input path for study 1
#// args[3] =   input path for study 2
#// ...
#// args[K+1] = input path for study K
OUTPUT=/rankproduct/output
NUM_OF_INPUT=3
INPUT1=/rankproduct/input1
INPUT2=/rankproduct/input2
INPUT3=/rankproduct/input3

# remove all files under input
$HADOOP_HOME/bin/hadoop fs -rmr $OUTPUT
#
# remove all files under output
driver=org.dataalgorithms.bonus.rankproduct.spark.SparkRankProductUsingGroupByKey
$SPARK_HOME/bin/spark-submit --class $driver \
    --master yarn-cluster \
    --jars $OTHER_JARS \
    --conf "spark.yarn.jar=$SPARK_JAR" \
    $APP_JAR $OUTPUT $NUM_OF_INPUT $INPUT1  $INPUT2  $INPUT3
```

8.2 Input

```
# hadoop fs -cat /rankproduct/input1/rp1.txt
K_1,30.0
K_2,60.0
K_3,10.0
```

```
K_4,80.0

# hadoop fs -cat /rankproduct/input2/rp2.txt
K_1,90.0
K_2,70.0
K_3,40.0
K_4,50.0

# hadoop fs -cat /rankproduct/input3/rp3.txt
K_1,4.0
K_2,8.0
```

8.3 Output

```
# hadoop fs -cat /rankproduct/output/part*
(K_2,(1.5874010519681994,3))
(K_3,(4.0,2))
(K_1,(1.8171205928321397,3))
(K_4,(1.7320508075688772,2))
```

9 Sample Run using combineByKey()

Not that the Spark solution using `combineByKey()` is more efficient than `groupByKey()` solution.

9.1 The Script

```
$ cat ./run_rank_product_using_combine_by_key_spark_yarn.sh
# define the installation dir for hadoop
export HADOOP_HOME=/Users/mparsian/zmp/zs/hadoop-2.6.0
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export HADOOP_HOME_WARN_SUPPRESS=true
#
export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.7.0_60.jdk/Contents/Home
export BOOK_HOME=/Users/mparsian/zmp/github/data-algorithms-book
export LIB=$BOOK_HOME/lib
export SPARK_HOME=/Users/mparsian/spark-1.4.0
export SPARK_JAR=$BOOK_HOME/lib/spark-assembly-1.4.0-hadoop2.6.0.jar
export APP_JAR=$BOOK_HOME/dist/data_algorithms_book.jar
# defines some environment for hadoop
source $HADOOP_CONF_DIR/hadoop-env.sh
#
# build all other dependent jars in OTHER_JARS
```

```
JARS=‘find $LIB -name ’*.jar’  ! -name ’*spark-assembly-1.4.0-hadoop2.6.0.jar’ ‘
OTHER_JARS=""
for J in $JARS ; do
   OTHER_JARS=$J,$OTHER_JARS
done
#
# define input/output for Hadoop/HDFS
#// args[0] = output path
#// args[1] = number of studies (K)
#// args[2] =   input path for study 1
#// args[3] =   input path for study 2
#// ...
#// args[K+1] = input path for study K
OUTPUT=/rankproduct/output
NUM_OF_INPUT=3
INPUT1=/rankproduct/input1
INPUT2=/rankproduct/input2
INPUT3=/rankproduct/input3

# remove all files under input
$HADOOP_HOME/bin/hadoop fs -rmr $OUTPUT
#
# remove all files under output
driver=org.dataalgorithms.bonus.rankproduct.spark.SparkRankProductUsingCombineByKey
$SPARK_HOME/bin/spark-submit --class $driver \
    --master yarn-cluster \
    --jars $OTHER_JARS \
    --conf "spark.yarn.jar=$SPARK_JAR" \
    $APP_JAR $OUTPUT $NUM_OF_INPUT $INPUT1  $INPUT2  $INPUT3
```

9.2 Input

```
# hadoop fs -cat /rankproduct/input1/rp1.txt
K_1,30.0
K_2,60.0
K_3,10.0
K_4,80.0

# hadoop fs -cat /rankproduct/input2/rp2.txt
K_1,90.0
K_2,70.0
K_3,40.0
K_4,50.0

# hadoop fs -cat /rankproduct/input3/rp3.txt
```

```
K_1,4.0
K_2,8.0
```

9.3 Output

```
# hadoop fs -cat /rankproduct/output/part*
(K_2,(1.5874010519681994,3))
(K_1,(1.8171205928321397,3))
(K_4,(1.7320508075688772,2))
(K_3,(4.0,2))
```

10 References

1. Rank Product (https://en.wikipedia.org/wiki/Rank_product)
2. Geometric mean (https://en.wikipedia.org/wiki/Geometric_mean)
3. Rank products: a simple, yet powerful, new method to detect differentially regulated genes in replicated microarray experiments. By Rainer Breitling, Patrick Armengaud, Anna Amtmann, Pawel Herzyk, 11 August 2004
4. Rank Product: http://www.molmine.com/magma/analysis/rank_product.html
5. Avoid groupByKey: `http://databricks.gitbooks.io/databricks-spark-knowledge-base/content/best_practices/prefer_reducebykey_over_groupbykey.html`
6. Fast Data Processing with Spark - Second Edition, By Krishna Sankar and Holden Karau