Big Data
CS522 - MUM

By: Professor Premchand Nair

Final Project

Mo Nuaimat

Hadoop Pair Approach - Pseudo code

Driver:

1. Read input file
2. Call mappers
3. Call reducers

Mapper:

```
map(Text inputLine)
For each word in input line do:
        Neighbors <- findNeighbors(word)
        For each Word n in Neighbors do:
                emit(Pair(word, n), 1)
                // for counting relative
                emit(Pair(word, “*”), 1)
```

Reducer:

```
Init:
        keyTotal <- new Sequence
reduce(Pair p(k1, k2), List v[1, 1, ….])
        Sum = 0
        for(int v1 in v[1, 1, …]){
                Sum += v1
        }
        if(k2 = “*”){
                keyTotal[k1] <- sum; // occurnces count
        } else {
                relativeFreq = sum/keyTotal[k1]
                Pair p <- (k1,k2)
                emit(new Pair(p, relativeFreq))
        }
```

Hadoop Pair Approach - Java code - Driver

Driver:

1. Read input file
2. Call mappers
3. Call reducers

```
Job job = new Job(getConf());
job.setJarByClass(RelativeFreqPairsDriver.class);
job.setJobName(this.getClass().getName());

FileInputFormat.setInputPaths(job, new Path(input));
FileOutputFormat.setOutputPath(job, new Path(output));

job.setMapperClass(RelativeFreqPairsMapper.class);
job.setReducerClass(RelativeFreqPairsReducer.class);

job.setMapOutputKeyClass(TextPair.class);
job.setMapOutputValueClass(IntWritable.class);

job.setOutputKeyClass(TextPair.class);
job.setOutputValueClass(IntWritable.class);

boolean success = job.waitForCompletion(true);
return success ? 0 : 1;
```

Hadoop Pair Approach - Java code - Mapper

Mapper:

map(Text inputLine)
For each word in input line do:
    Neighbors <- findNeighbors()
    For each Word n in Neighbors do:
        emit(Pair(word, n), 1)
        // for counting relative
        emit(Pair(word, “*”), 1)

```
public void map(LongWritable key, Text value, Context context) throws I
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);
    ArrayList<Text> words = new ArrayList<>();
    while (tokenizer.hasMoreTokens()) {
        words.add(new Text(tokenizer.nextToken()));
    }

    ArrayList<TextPair> neighboursList =
            findNeighbourPairs(words.remove(0), words);

    System.out.println("Mapper output: ");
    for(TextPair gp:neighboursList){
        TextPair counter = new TextPair(gp.getKey(), new Text("*"));
        context.write(gp, one);
        context.write(counter, one);
    }
}
```

Hadoop Pair Approach - Java code - Mapper

Mapper:

map(Text inputLine)
For each word in input line do:
    Neighbors <- **findNeighbors()**
    For each Word n in Neighbors do:
        emit(Pair(word, n), 1)
        // for counting relative
        emit(Pair(word, "*"), 1)

```
public static ArrayList<TextPair> findNeighbourPairs(Text w, ArrayList<Tex
    if(list.size() == 1){
        TextPair gp = new TextPair( new Text(w) , new Text(list.remove(0))
        return new ArrayList<>(Arrays.asList(gp));
    }

    ArrayList<TextPair> ret = new ArrayList<>();
        for(Text w2:list){
            if(w2.equals(w)){
                break;
            }
            ret.add(new TextPair(w, w2));
        }

    ret.addAll( findNeighbourPairs(list.remove(0), list) );
    return ret;
}
```

Hadoop Pair Approach - Java code - Mapper

TextPair Class:

All key Data types in Hadoop should
All writable and comparable

```
public class TextPair implements WritableComparable<TextPair> {
    GenericPair<Text, Text> pair = new GenericPair<>();
    public TextPair(){
        this(new Text(), new Text());
    }
    public TextPair(Text a,Text b){
        pair.setKey(a);
        pair.setVal(b);
    }
    @Override
    public void write(DataOutput out) throws IOException {
        this.pair.getKey().write(out);
        this.pair.getVal().write(out);
    }
    @Override
    public void readFields(DataInput in) throws IOException {
        this.pair.key.readFields(in);
        this.pair.val.readFields(in);
    }

    @Override
    public int compareTo(TextPair o) {
        return this.pair.compareTo(o.pair);
    }
```

Hadoop Pair Approach - Java code - Reducer

Reducer:

Init:
keyTotal <- new Sequence
reduce(Pair p(k1, k2), List v[1, 1, ….])
Sum = 0
for(int v1 in v[1, 1, …]){
Sum += v1
}
if(k2 = “*”){
keyTotal[k1] <- sum;
} else {
relativeFreq = sum/keyTotal[k1]
Pair p <- (k1,k2)
emit(new Pair(p, relativeFreq))
}

```
@Override
protected void reduce(
        TextPair key,
        Iterable<IntWritable> values,
        Context context)
        throws IOException, InterruptedException {

    int sum = 0;
    for (IntWritable val : values) {
        sum += val.get();
    }

    Text right = key.getVal();
    Text left = key.getKey();
    if(right.toString().equals("*")){
        termsFreqTotal.put(left, sum );
        return;
    } else {
        int count = termsFreqTotal.get(left);
        float relFreq = (1.f * sum) / (1.f * count);
        DecimalFormat df = new DecimalFormat("#.00");
        relFreq = Float.parseFloat(df.format(relFreq));
        System.out.println(new GenericPair<TextPair, Float>(key, relFreq));
        context.write(key, new FloatWritable(relFreq));
    }
}
```

Hadoop Pair Approach - Result (Reducer Output)

< A10 , B12 > 0.5
< A10 , D76 > 0.5
< A12 , A10 > 0.08
< A12 , B12 > 0.25
< A12 , B76 > 0.08
< A12 , C31 > 0.17
< A12 , D76 > 0.42
< B12 , A10 > 0.07
< B12 , A12 > 0.27
< B12 , B76 > 0.07
< B12 , C31 > 0.2
< B12 , D76 > 0.4
< B76 , A10 > 0.17
< B76 , B12 > 0.33
< B76 , C31 > 0.17
< B76 , D76 > 0.33

< C31 , A10 > 0.06
< C31 , A12 > 0.24
< C31 , B12 > 0.24
< C31 , B76 > 0.06
< C31 , D76 > 0.41
< D76 , A10 > 0.08
< D76 , A12 > 0.33
< D76 , B12 > 0.33
< D76 , B76 > 0.08
< D76 , C31 > 0.17

Hadoop Stripe Approach - Pseudo code

Driver:

1. Read input file
2. Call mappers
3. Call reducers

Mapper:

```
map(Text inputLine)
For each word in input line do:
        Neighbors <- new List of Assoc Arrays
        Neighbors <- findNeighbors(word)
        For each AssocArray a in Neighbors do:
                emit(word, a)
```

Reducer:

```
reduce(Word w, List of Assoc Arrays v[v1, v2, ....])
        AggArray <- new AssocArray
        Int sum = 0;
        for(AssocArray a in v[v1, v2, ...]){
                // key-wise sum
                AggArray = AggArray + a;
                foreach(int v in a){
                        sum += a;
                }
        }
        for(Pair(w2,c) in AggArray)
                relativeFreq = c/sum
                Pair p <- (w,w2)
                emit(new Pair(p, relativeFreq))
```

Hadoop Stripe Approach - Java code - Mapper

Mapper:

map(Text inputLine)
For each word in input line do:
    Neighbors <- new List of Assoc Arrays
    Neighbors <- findNeighbors(word)
    For each AssocArray a in Neighbors do:
        emit(word, a)

```
public class RelativeFreqStripesMapper extends
        Mapper<LongWritable, Text, Text, MapWritable> {

    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        ArrayList<Text> words = new ArrayList<>();
        while (tokenizer.hasMoreTokens()) {
            words.add(new Text(tokenizer.nextToken()));
        }

        ArrayList<GenericPair<Text, MapWritable>> neighboursList = findNeighbourHashMap(
                words.remove(0), words);

        System.out.println("Mapper Output:");
        for (GenericPair<Text, MapWritable> gp : neighboursList) {
            System.out.println(gp);
            context.write(gp.getKey(), gp.getVal());
        }
    }
```

Hadoop Stripe Approach - Java code - Mapper

Mapper:

map(Text inputLine)
For each word in input line do:
    Neighbors <- new List of Assoc Arrays
    Neighbors <- **findNeighbors**(word)
    For each AssocArray a in Neighbors do:
        emit(word, a)

```
private static ArrayList<GenericPair<Text, MapWritable>> findNeighbourHashMap(
        Text w, ArrayList<Text> list) {
    if (list.size() == 1) {
        MapWritable hm = new MapWritable();
        hm.put(list.remove(0), new IntWritable(1));
        GenericPair<Text, MapWritable> gp = new GenericPair<>(w, hm);
        return new ArrayList<>(Arrays.asList(gp));
    }
    ArrayList<GenericPair<Text, MapWritable>> ret = new ArrayList<>();

    HashMap<String, Integer> hm = new HashMap<>();
    for (Text w2 : list) {
        if (w2.equals(w)) {
            break;
        }
        if(hm.containsKey(w2.toString())){
            int oldValue = hm.get(w2.toString());
            hm.put(w2.toString(), oldValue + 1);
        } else {
            hm.put(w2.toString(), 1);
        }
    }
    MapWritable mw = new MapWritable();
    for(String s:hm.keySet()){
        mw.put(new Text(s), new IntWritable(hm.get(s)));
    }
    ret.add(new GenericPair<Text, MapWritable>(w, mw));
    ret.addAll(findNeighbourHashMap(list.remove(0), list));
    return ret;
}
```

Hadoop Stripe Approach - Java code - Reducer

Reducer:

```
reduce(Word w, List of Assoc Arrays v[v1, v2, ….])
        AggArray <- new AssocArray
        Int sum = 0;
        for(AssocArray a in v[v1, v2, …]){
                // key-wise sum
                AggArray = AggArray + a;
                foreach(int v in a){
                        sum += a;
                }
        }
        for(Pair(w2,c) in AggArray)
                relativeFreq = c/sum
                Pair p <- (w,w2)
                emit(new Pair(p, relativeFreq))
```

```java
@Override
protected void reduce(Text key,
        Iterable<MapWritable> values,
        Context context)
        throws IOException, InterruptedException {
    List<MapWritable> cache = new ArrayList<MapWritable>();

    CustomHashMap agg = new CustomHashMap();
    int count = 0;
    for (MapWritable hm : values) {
        cache.add(hm);
        for(Writable k:hm.keySet()){
            Text t = (Text) k;
            count += ((IntWritable) hm.get(k)).get();
            if(agg.containsKey(t)){
                int old = agg.get(t).get();
                old += ((IntWritable) hm.get(k)).get();
                agg.put(t, new IntWritable(old));
            } else {
                agg.put(t, new IntWritable(((IntWritable) hm.get(k)).get()));
            }
        }
    }
```

Hadoop Stripe Approach - Java code - Reducer

```java
@Override
protected void reduce(Text key,
        Iterable<MapWritable> values,
        Context context)
        throws IOException, InterruptedException {
    List<MapWritable> cache = new ArrayList<MapWritable>();

    CustomHashMap agg = new CustomHashMap();
    int count = 0;
    for (MapWritable hm : values) {
        cache.add(hm);
        for(Writable k:hm.keySet()){
            Text t = (Text) k;
            count += ((IntWritable) hm.get(k)).get();
            if(agg.containsKey(t)){
                int old = agg.get(t).get();
                old += ((IntWritable) hm.get(k)).get();
                agg.put(t, new IntWritable(old));
            } else {
                agg.put(t, new IntWritable(((IntWritable) hm.get(k)).get()));
            }
        }
    }
    for(Text k2:agg.keySet()){
        int sum = agg.get(k2).get();

        float relFreq = (1.f * sum) / (1.f * count);

        DecimalFormat df = new DecimalFormat("#.00");
        relFreq = Float.parseFloat(df.format(relFreq));
        TextPair tp = new TextPair(key, k2);
        System.out.println(new GenericPair<TextPair, Float>(tp,relFreq));
        context.write(tp, new FloatWritable(relFreq));
    }
    agg = null;
```

Hadoop Stripe Approach - Result (Reducer Output)

< A10 , B12 > 0.5
< A10 , D76 > 0.5
< A12 , B76 > 0.08
< A12 , A10 > 0.08
< A12 , B12 > 0.25
< A12 , D76 > 0.42
< A12 , C31 > 0.17
< B12 , B76 > 0.07
< B12 , A10 > 0.07
< B12 , A12 > 0.27
< B12 , D76 > 0.4
< B12 , C31 > 0.2
< B76 , A10 > 0.17
< B76 , B12 > 0.33
< B76 , D76 > 0.33
< B76 , C31 > 0.17

< C31 , B76 > 0.06
< C31 , A12 > 0.24
< C31 , A10 > 0.06
< C31 , B12 > 0.24
< C31 , D76 > 0.41
< D76 , B76 > 0.08
< D76 , A10 > 0.08
< D76 , A12 > 0.33
< D76 , B12 > 0.33
< D76 , C31 > 0.17

Hadoop Hybrid Approach - Pseudo code

Driver:

1. Read input file
2. Call mappers
3. Call reducers

Mapper:

```
map(Text inputLine)
For each word in input line do:
        Neighbors <- findNeighbors(word)
        For each Word n in Neighbors do:
                emit(Pair(word, n), 1)
```

Reducer:

```
Init:
        buffer <- new Sequence
        lastKey <- String
reduce(Pair p(k1, k2), List values[1, 1, ....]):
        if(k1 != lastKey)
                flushBuffer(lastKey)
                lastKey = k1

        for(int v in values[1, 1, ...]){
                buffer{k2}.add ( v )
        }
}

Close:
        flushBuffer(lastKey)
```

Hadoop Hybrid Approach - Java code - Mapper

Mapper:

map(Text inputLine)
For each word in input line do:
    Neighbors <- findNeighbors(word)
    For each Word n in Neighbors do:
        emit(Pair(word, n), 1)

```
private final static IntWritable one = new IntWritable(1);

public void map(LongWritable key, Text value, Context context)
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);
    ArrayList<Text> words = new ArrayList<>();
    while (tokenizer.hasMoreTokens()) {
        words.add(new Text(tokenizer.nextToken()));
    }

    ArrayList<TextPair> neighboursList =
            findNeighbourPairs(words.remove(0), words);

    for(TextPair gp:neighboursList){
        context.write(gp, one);
    }
}
```

Hadoop Hybrid Approach - Java code - Mapper

Mapper:

map(Text inputLine)
For each word in input line do:
    Neighbors <- **findNeighbors**(word)
    For each Word n in Neighbors do:
        emit(Pair(word, n), 1)

```
public static ArrayList<TextPair> findNeighbourPairs(Text w, ArrayList<Text> list){
    if(list.size() == 1){
        TextPair gp = new TextPair( new Text(w) , new Text(list.remove(0)) );
        return new ArrayList<>(Arrays.asList(gp));
    }

    ArrayList<TextPair> ret = new ArrayList<>();
        for(Text w2:list){
            if(w2.equals(w)){
                break;
            }
            ret.add(new TextPair(w, w2));
        }

    ret.addAll( findNeighbourPairs(list.remove(0), list) );
    return ret;
```

Hadoop Hybrid Approach - Java code - Reducer

Reducer:

```
Init:
        buffer <- new Sequence
        lastKey <- String
reduce(Pair p(k1, k2), List values[1, 1, ....]):
        if(k1 != lastKey)
                flushBuffer(lastKey)
                lastKey = k1

        for(int v in values[1, 1, ...]){
                buffer{k2}.add ( v )
        }
}

Close:
        flushBuffer(lastKey)
```

```java
private static TreeMap<String, Integer> buffer = new TreeMap<>();
private static String lastKey = null;

@Override
protected void reduce(
        TextPair key,
        Iterable<IntWritable> values,
        Context context)
        throws IOException, InterruptedException {

    String thisKey = key.getKey().toString();
    String thisVal = key.getVal().toString();

    if(!thisKey.equals(lastKey)){
        flushBuffer(lastKey, context);
        lastKey = new String(thisKey);
    }
    Iterator<IntWritable> it = values.iterator();
    while(it.hasNext()){
    IntWritable i = it.next();
        if(!buffer.containsKey(thisVal)){
            buffer.put(thisVal, i.get());
        } else {
            Integer oldValue  = buffer.get(thisVal);
            buffer.put(thisVal, new Integer(oldValue +  i.get()));
        }
    }
}
```

Hadoop Hybrid Approach - Java code - Reducer

Reducer:

Init:
    buffer <- new Sequence
    lastKey <- String
reduce(Pair p(k1, k2), List values[1, 1, ….]):
    if(k1 != lastKey)
        flushBuffer(lastKey)
        lastKey = k1

    for(int v in values[1, 1, …]){
        buffer{k2}.add (v)
    }
}

Close:
    flushBuffer(lastKey)

```
private void flushBuffer(
        String thisKey,
        Context context) throws IOException, InterruptedException {

    int hmSum = 0;
    for(String k:buffer.keySet()){
        hmSum += buffer.get(k);
    }

    for(String k:buffer.keySet()){
        float val = 1.f * buffer.get(k);
        float relFreq = val / (hmSum*1.f);
        DecimalFormat df = new DecimalFormat("#.00");
        relFreq = Float.parseFloat(df.format(relFreq));
        TextPair gp = new TextPair(new Text(thisKey), new Text(k));
        context.write(gp, new FloatWritable(relFreq) );
    }

    buffer = new TreeMap<>();
}
```

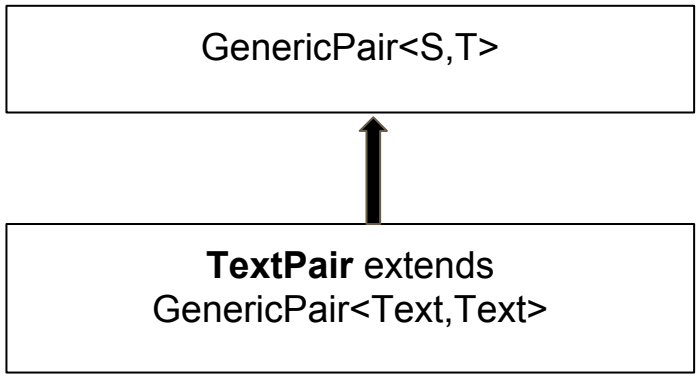
Hadoop Hybrid Approach - Result (Reducer Output)

< A10 , B12 > 0.5
< A10 , D76 > 0.5
< A12 , A10 > 0.08
< A12 , B12 > 0.25
< A12 , B76 > 0.08
< A12 , C31 > 0.17
< A12 , D76 > 0.42
< B12 , A10 > 0.07
< B12 , A12 > 0.27
< B12 , B76 > 0.07
< B12 , C31 > 0.2
< B12 , D76 > 0.4
< B76 , A10 > 0.17
< B76 , B12 > 0.33
< B76 , C31 > 0.17
< B76 , D76 > 0.33

< C31 , A10 > 0.06
< C31 , A12 > 0.24
< C31 , B12 > 0.24
< C31 , B76 > 0.06
< C31 , D76 > 0.41
< D76 , A10 > 0.08
< D76 , A12 > 0.33
< D76 , B12 > 0.33
< D76 , B76 > 0.08
< D76 , C31 > 0.17

Hadoop - Comparators

For TextPair Data type that is used as a key to Mapper output




```
@Override
public int compareTo(Object o) {
     if(! (o instanceof GenericPair) ) {
            throw new RuntimeException("Can't compare to " + o);
        }
        GenericPair gp = (GenericPair) o;
        if (this.getKey() instanceof Comparable) {
            int n = ((Comparable) this.key).compareTo((Comparable) gp.key);
            if(n != 0){
                return n;
            }
            return ((Comparable) this.val).compareTo((Comparable) gp.val);
        }

        return 0;
}
```

Hadoop Project - Demo

- I setup my project using maven
  mvn clean package
- I created a shell script to help me:
  a. Generate jar file using mvn
  b. Copy sample input files to HDFS
  c. Run the generated JAR file and write results to a new folder with current datetime as part of the directory name
  d. Print out the output directory name
  e. Fetch and Print output from HDFS

start.sh

```
#!/bin/bash

JAR_FILE="target/RelativeFreqPairs-1.0-SNAPSHOT.jar"
CLASS_NAME="edu.mum.bigdata.mo.RelativeFreqPairsDriver"
HADOOP_OUTPUT_FOLDER="/user/hive/warehouse/chd_pairs_`date +'%Y%m%d%H%M

INPUT_FILES="cust*.txt"
HADOOP_INPUT_DEST="/user/hive/warehouse/custHistData"

mvn clean package
sudo -u hdfs hadoop fs -copyFromLocal $INPUT_FILES $HADOOP_INPUT_DEST
sudo hadoop jar $JAR_FILE $CLASS_NAME $HADOOP_INPUT_DEST $HADOOP_OUTPUT

echo "Output written to $HADOOP_OUTPUT_FOLDER"
echo "Output was: "
hadoop fs -cat $HADOOP_OUTPUT_FOLDER/part-r-00000
```

Spark Project - Problem Statement

- CDH provides a month-long worth of apache log file for DataCo
- DataCo is an online store that sells products to users.
- I will parse this log file and came up with co-occurence matrix for the products that cross-sale together, so, If users who buy product X usually buy product Y, I want my report to show top 50 items that goes together.
- For the purpose of this project, we are going to assume each IP address represents a different user.

```
) Chrome/35.0.1916.153 Safari/537.36"
op/category/camping%20&%20hiking/product/Diamondback%20
.1; rv:30.0) Gecko/20100101 Firefox/30.0"
op/category/indoor/outdoor%20games/product/O'Brien%20M
0.0) Gecko/20100101 Firefox/30.0"
'category/trade-in/product/Glove%20It%20Imperial%20Golf
ML, like Gecko) Chrome/36.0.1985.125 Safari/537.36"
op/category/fishing/product/Field%20&%20Stream%20Sport
pleWebKit/537.76.4 (KHTML, like Gecko) Version/7.0.4 S
category/baseball%20&%20softball/product/adidas%20Kids'
 Gecko/20100101 Firefox/30.0"
category/fitness%20accessories/product/Under%20Armour%2
 10_9_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
'category/trade-in/product/Glove%20It%20Urban%20Brick%2
6 (KHTML, like Gecko) Chrome/35.0.1916.153 Safari/537.
op/category/indoor/outdoor%20games/product/O'Brien%20M
leWebKit/537.77.4 (KHTML, like Gecko) Version/7.0.5 Sa
'category/electronics/product/Under%20Armour%20Kids'%20
37.36 (KHTML, like Gecko) Chrome/35.0.1916.153 Safari/
ry/shop%20by%20sport/product/Under%20Armour%20Girls'%2
t/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153 Safa
```

Spark Project - DataCo Co-occurenceM Pseudo code

```
Log <- load log file
ipProductPairs <- new Array
For each entry in log do:
        Pair p <- parse(entry)[“ip”, “product”]
                filter(only with “add_to_cart”)
                cleanText
        ipProductsPair[] = p;

Pairs neighbours <- new Array

For each ip in inProductPairs
        itemsForUser = inProductPairs[ip]
        Neigh <- findN(itemsForUser)
        For each Pair p in Neigh
                Neighbours[] = Neigh
                Neighbours[] = new Pair(prod,’*’)
```

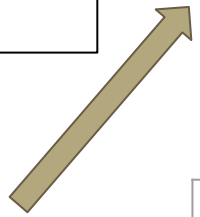
```
neighboursSorted <- sortByKey(neighbours)
productsCount  = find count of each product
freqSortedRatio <- RelCoValues(neighboursSorted)

Save freqSortedRatio into HDFS
Top50 <- sortByValue(freqSortedRatio).take(50)
Print Top50
```

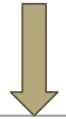
Spark Project - Problem Statement

(219.281.28.142, Product A)
(219.281.28.142, Product B)
(219.281.28.142, Product C)
(155.123.128.255, Product B)
(155.123.128.255, Product A)
(155.123.128.255, Product B)

(219.281.28.142, [Product A, B, C])
(155.123.128.255, [Product B, A, B])



For user 1:	For user 2:
((A, B), 1)	((B,A), 1)
((A, C), 1)	((B, *), 1)
((A, *), 2)	((A,B), 1)
((B, C), 1)	((A, *), 1)
((B, *), 1)	



((A, B), [1 + 1] / [2 + 1])	=	((A, B), ⅔)
((A, C), [1] / [2 + 1])	=	((A, C), ⅓)
((B, A), [1] / [1+1])	=	((B,A), ½)
((B, C), [1] / [1+1])	=	((B,C), ½)

Spark Project - DataCo Co-occurenceM Java code

Parsing Apache Log entries

```
JavaPairRDD<String, List<String>> ipProductPairs = sc
        .textFile(args[0])
        .map(new Function<String, Tuple2<String, String>>() {
            public Tuple2<String, String> call(String line)
                    throws Exception {
                ApacheAccessLog parser = ApacheAccessLog.parseFromLogLine(line);

                String endpoint = parser.getEndpoint();
                endpoint = URLDecoder.decode(endpoint, "UTF-8");
                //log.error("endpoint is : " + endpoint);
                return new Tuple2<String, String>(
                        parser.getIpAddress(), endpoint.toLowerCase());
            }
        })
        .filter(new Function<Tuple2<String, String>, Boolean>() {
            @Override
            public Boolean call(Tuple2<String, String> item)
                    throws Exception {
                return item._2.contains("/product/")
                        && item._2.contains("/add_to_cart");
            }

        })
```

Spark Project - DataCo Co-occurenceM Java code

Reduce by Key (IP Address)
Finding Neighbour pairs and
Mapping to:
- **Pair(Product, product)**
- **Pair(Product, *)**

```
JavaPairRDD<Tuple2<String, String>, Integer> neighbours = ipProductPairs.reduceByKey(
new Function2<List<String>, List<String>, List<String>>(){
    @Override
    public List<String> call(List<String> v1, List<String> v2)
            throws Exception {
        List<String> prod = new ArrayList<String>();
        prod.addAll(v1);
        prod.addAll(v2);
        return prod;
    }

}).map(new Function<Tuple2<String, List<String>>, List<Tuple3<String,  String, Integer>>>(){
    @Override
    public List<Tuple3<String, String, Integer>> call(
            Tuple2<String, List<String>> record) throws Exception {
        ArrayList<Tuple2<String, String>> neighbors = findNeighbourPairs(record._2().remove(0
        List<Tuple3<String,  String, Integer>> ret = new ArrayList<Tuple3<String,  String, In
        for(Tuple2<String, String> gp:neighbors){
            ret.add(new Tuple3<String, String,Integer>(gp._1(), gp._2(), 1));
            ret.add(new Tuple3<String, String,Integer>(gp._1(), "*", 1));
        }
        return ret;
    }}).flatMap(new FlatMapFunction<List<Tuple3<String, String, Integer>>, Tuple3<String, Str
```

Spark Project - DataCo Co-occurenceM Java code

Find count of each product

Use that count to calculate Relative co-occurrence

Remove count of each rows

```
JavaRDD<Tuple2<Tuple2<String, String>, Float>> freqSortedRatio = neighboursSorted
  .map(new Function<Tuple2<Tuple2<String, String>, Integer>, Tuple2<Tuple2<String, String>, Floa
  @Override
  public Tuple2<Tuple2<String, String>, Float> call(
      |   Tuple2<Tuple2<String, String>, Integer> t)
          throws Exception {
      if(t._1()._2().equals("*")){
          CountHolder.lastCounter = t._2();
          return new Tuple2<Tuple2<String, String>, Float>(t._1(), 0.f);
      }
      float ratio = (1.f * t._2())/( 1.f * CountHolder.lastCounter );
      DecimalFormat df = new DecimalFormat("#.00");
      ratio = Float.parseFloat(df.format(ratio));
      System.out.println(t._1() + " : " + ratio);

      return new Tuple2<Tuple2<String, String>, Float>(t._1(), ratio);
  }

}).filter(new Function<Tuple2<Tuple2<String, String>, Float>, Boolean>(){

  @Override
  public Boolean call(Tuple2<Tuple2<String, String>, Float> v1)
          throws Exception {
      return !v1._1()._2().equals("*");
  }});

freqSortedRatio.saveAsTextFile("hdfs:///user/hive/warehouse/spark_prod_relfreq_" + System.curren
```

Spark Project - DataCo Co-occurenceM Java code

Find top 50 products that Sells together

```
List<Tuple2<Tuple2<String, String>, Float>> top50byVal = freqSortedRatio
        .takeOrdered(50, new ValComparator());
System.out.println("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-");
System.out.println("TOP Relations: ");
System.out.println("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-");
for(Tuple2<Tuple2<String, String>, Float> pair:top50byVal) {
    System.out.println(pair._1() + " -> " + pair._2());
}
```

Spark Project - DataCo Co-occurenceM Java code

Comparator for key

```
public class KeyComparator implements Comparator<Tuple2<String, String>>, Serializable

        @Override
        public int compare(Tuple2<String, String> o1,
                Tuple2<String, String> o2) {
            int c1 = o1._1().compareTo(o2._1());
            if(c1 != 0)
                return c1;
            return o1._2().compareTo(o2._2());
        }
}
```

Comparator for values

```
public class ValComparator implements
        Serializable, Comparator<Tuple2<Tuple2<String, String>, Float>> {

    @Override
    public int compare(Tuple2<Tuple2<String, String>, Float> o1,
            Tuple2<Tuple2<String, String>, Float> o2) {
        return -1*Float.compare(o1._2(), o2._2());
    }

}
```

Spark Project - DataCo Co-occurenceM Output

-=-
TOP Relations:
-=-

(bushnell pro x7 jolt slope rangefinder,
nike men's cj elite 2 td football cleat) -> 0.19

(clicgear rovic cooler bag,
nike men's cj elite 2 td football cleat) -> 0.17

(polar ft4 heart rate monitor,
nike men's dri-fit victory golf polo) -> 0.13

(under armour kids' mercenary slide,
adidas kids' rg iii mid football cleat) -> 0.13

(bridgestone e6 straight distance nfl san dieg,nike
men's cj elite 2 td football cleat) -> 0.12

(clicgear rovic cooler bag,
perfect fitness perfect rip deck) -> 0.11

(garmin forerunner 910xt gps watch,
nike men's fingertrap max training shoe) -> 0.11

(glove it urban brick golf towel,
nike men's dri-fit victory golf polo) -> 0.11

(hirzl men's hybrid golf glove,
nike men's cj elite 2 td football cleat) -> 0.11

(bag boy beverage holder,
perfect fitness perfect rip deck) -> 0.11

(garmin approach s4 golf gps watch,
nike men's dri-fit victory golf polo) -> 0.1

Spark Project - DataCo Co-occurenceM

DeMo

:)