TraditionalCollaborativeFiltering

Abstract

Collaborative filtering is an important personalized method in recommender systems in internet commerce. It is infeasible that traditional collaborative filtering is based on absolute rating for items since users are difficult to accurately make an absolute rating for items, and also different users give different rating distribution. In this tutorial, it shows that how to use a Hama to calculate TCF.

Implementation

Build a user by item matrix, Set entries from the raw data

...

Get the pairs of all row key combinations w/o repetition

We don't have to recalculate the same value pair with reversed order.

ex) similar(UserA, UserB) == similar(UserB, UserA).

In this case, it is going to return `1, 2}, {1, 3}, {2, 3` by discarding `2, 1}, {3, 1}, {3, 2` from the full possible combination. Since there will be mC2 combinations (m : num keys), one can optimize it to have mC2 / N values per reducer (N : num-reducers). Something like :

```
partition(index i, key key_j, int N) { // N is num reducers
 // find the data per reducer
 int dataPerRed = mC2 / N; // assuming m is known
 int prev_sum = 0;
 // calculate the total combinations contributed by previous indexes
 for (k=1; k < i; k++) {
  prev_sum += m - k + 1; // this adds the number of combinations contributed by kth index
 }
 prev_sum += j - i + 1 // self contribution
 return prev_sum % dataPerRed
}
```

Calculate |a|·|b|cos(q) on Map/Reduce

...

Collect the similarity result of the two users

...

Pseudo code for TCF with Hama

```
import java.math.BigInteger;

import org.apache.hama.HamaConfiguration;
import org.apache.hama.Matrix;
import org.apache.hama.Vector;

public class TraditionalCF {
  public static double[][] data = {
    { 2, 5, 1, 4 },
    { 4, 1, 3, 3 },
    { 3, 4, 2, 4 }
```

```
};

public static void main(String[] args) {
  HamaConfiguration conf = new HamaConfiguration();

  // 1. Build a user by item matrix, Set entries from the raw data
  Matrix userByItem = new Matrix(conf, "userByItem");
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 4; j++) {
      userByItem.set(i, j, data[i][j]);
    }
  }

  // 2. Get the pair set of all row key combinations
  Combination x = new Combination(data.length, 2);

  // 3. |a|·|b|cos(q) calculation
  while (x.hasMore()) {
    int[] pair = x.getNext();
    System.out.print("Similarity: (" + pair[0] + ", " + pair[1] + ") = ");

    Vector v1 = userByItem.getRow(pair[0]);
    Vector v2 = userByItem.getRow(pair[1]);
    double similarity = v1.dot(v2);

    // 4. Collect the similarity result of the two users
    System.out.println(similarity);
  }

  // Each part can be applied to large-scale job scheduling using Map/Reduce.
}

static class Combination {

  private int[] a;
  private int n;
  private int r;
  private BigInteger numLeft;
  private BigInteger total;

  public Combination(int n, int r) {
    if (r > n) {
      throw new IllegalArgumentException();
    }
    if (n < 1) {
      throw new IllegalArgumentException();
    }
    this.n = n;
    this.r = r;
    a = new int[r];
    BigInteger nFact = getFactorial(n);
    BigInteger rFact = getFactorial(r);
    BigInteger nminusrFact = getFactorial(n - r);
    total = nFact.divide(rFact.multiply(nminusrFact));
    reset();
  }

  public void reset() {
    for (int i = 0; i < a.length; i++) {
      a[i] = i;
    }
    numLeft = total;
  }

  public BigInteger getNumLeft() {
    return numLeft;
  }

  public boolean hasMore() {
    return numLeft.compareTo(BigInteger.ZERO) > 0;
  }
```

```
    public BigInteger getTotal() {
      return total;
    }

    private BigInteger getFactorial(int n) {
      BigInteger fact = BigInteger.ONE;
      for (int i = n; i > 1; i--) {
        fact = fact.multiply(BigInteger.valueOf(i));
      }
      return fact;
    }

    public int[] getNext() {
      if (numLeft.equals(total)) {
        numLeft = numLeft.subtract(BigInteger.ONE);
        return a;
      }

      int i = r - 1;
      while (a[i] == n - r + i) {
        i--;
      }
      a[i]++;
      for (int j = i + 1; j < r; j++) {
        a[j] = a[i] + j - i;
      }

      numLeft = numLeft.subtract(BigInteger.ONE);
      return a;
    }
  }
}
```