CRYSTALBALL PROJECT

Professor: Prem Nair
Student Name: Bao Pham
Date: September 19th, 2015

PSEUDO CODE FOR PAIR APPROACH

```
class Mapper
        method Initialize
                H = new AssociativeArray()

        method Map(docid a; doc d)
                for all term w in doc d do
                        for all term u in Neighbors(w) do
                                H{pair(w; u)} ← H{pair(w; u)} + 1
                                H{pair(w; *)} ← H{pair(w; *)} + 1

method Close
        for all pair p in H do
                Emit(pair p; count H{pair p})
```

```
class Reducer
        method Initialize
                marginal = 0

        method Reduce(pair p; counts[c1; c2; ...])
                sum = 0;
                relativeFrequency = 0.0;
                for all count c in counts[c1; c2; ...] do
                        if (pair p == (w; *))
                                marginal = marginal + c
                else
                                sum = sum + c
        relativeFrequency = sum / marginal
        Emit(pair p; double relativeFrequency)
```

JAVA CODE FOR PAIR APPROACH

```
public void map(LongWritable offset, Text lineText, Context context) throws IOException, InterruptedException {
		LOG.info("Starting mapping ");
		if (lineText != null) {
			String[] listTerm = lineText.toString().split("\\s+");
			if (listTerm != null) {
				for (int i = 0; i < listTerm.length - 1; i++) {
					String currentTerm = listTerm[i];
					Pair totalPair = new Pair(currentTerm, "*");
					for (int j = i + 1; j < listTerm.length; j++) {
						if (currentTerm.equals(listTerm[j]))
							break;
						Pair pair = new Pair(currentTerm, listTerm[j]);
						if (pairMap.containsKey(pair)) {
							LOG.info("Pair " + pair + " has already existed");
							int counter = pairMap.get(pair);
							counter++;
							pairMap.put(pair, counter);
						} else {
							pairMap.put(pair, one);
						}

						if (pairMap.containsKey(totalPair)) {
							int counter = pairMap.get(totalPair);
							counter++;
							pairMap.put(totalPair, counter);
						} else {
							pairMap.put(totalPair, one);
						}

					}
				}
			}
		}
	}

	public void cleanup(Context context) throws IOException, InterruptedException {
		Enumeration<Pair> enumerator = pairMap.keys();
		while (enumerator.hasMoreElements()) {
			Pair p = enumerator.nextElement();
			LOG.info("<Pair, value> = (" + p.getTerm1() + ", " + p.getTerm2() + "), " + pairMap.get(p));
			context.write(p, new IntWritable(pairMap.get(p)));
		}
	}
```

```
public void reduce(Pair key, Iterable<IntWritable> counts,
Context context)
			throws IOException, InterruptedException {
		int sum = 0;
		double relativeFrequency = 0.0;
		if (currentTerm.equals("")) {
			currentTerm = key.term1;
		} else if (!currentTerm.equals(key.term1)) {
			marginal = 0;
			currentTerm = key.term1;
		}
		LOG.info("Current Pair" + key);
		Iterator<IntWritable> iterator = counts.iterator();
		while (iterator.hasNext()) {
			int val = iterator.next().get();
			LOG.info("Value " + val);
			if (key.term2.equals("*")) {
				marginal += val;
			} else {
				sum += val;
			}
		}

		LOG.info("Current pair (" + key);
		LOG.info("Marginal " + marginal);
		LOG.info("Sum " + sum);
		if (!key.term2.equals("*")) {
			relativeFrequency = (double) sum / marginal;
			relativeFrequency =
Double.parseDouble(formatDouble(relativeFrequency));
			LOG.info("Frequency " + relativeFrequency);
			context.write(key, new
DoubleWritable(relativeFrequency));
		}

	}
```

RESULT OF PAIR APPROACH

(9, 46)	1.0
(10, 12)	0.5
(10, 34)	0.5
(12, 10)	0.09
(12, 18)	0.09
(12, 34)	0.36
(12, 56)	0.18
(12, 79)	0.09
(12, 92)	0.18
(16, 9)	0.25
(16, 28)	0.25
(16, 46)	0.25
(16, 56)	0.25
(18, 12)	0.25
(18, 29)	0.12
(18, 34)	0.25
(18, 56)	0.12
(18, 79)	0.12
(18, 92)	0.12
(26, 9)	0.11
(26, 16)	0.11
(26, 28)	0.22
(26, 39)	0.11
(26, 46)	0.22
(26, 56)	0.22
(28, 9)	0.17
(28, 16)	0.17
(28, 39)	0.17
(28, 46)	0.33
(28, 56)	0.17
(29, 10)	0.07
(29, 12)	0.27
(29, 18)	0.07
(29, 34)	0.27
(29, 56)	0.13
(29, 79)	0.07
(29, 92)	0.13
(34, 10)	0.08
(34, 12)	0.25
(34, 18)	0.08
(34, 29)	0.08
(34, 56)	0.25
(34, 79)	0.08
(34, 92)	0.17
(39, 9)	0.17
(39, 16)	0.17
(39, 28)	0.17
(39, 46)	0.33
(39, 56)	0.17
(46, 9)	0.25
(46, 16)	0.25
(46, 28)	0.25
(46, 56)	0.25
(56, 9)	0.06
(56, 10)	0.06
(56, 12)	0.18
(56, 16)	0.06
(56, 28)	0.12
(56, 29)	0.06
(56, 34)	0.18
(56, 39)	0.06
(56, 46)	0.12
(56, 92)	0.12
(79, 12)	0.2
(79, 18)	0.2
(79, 34)	0.2
(79, 56)	0.2
(79, 92)	0.2
(92, 10)	0.33
(92, 12)	0.33
(92, 34)	0.33

PSEUDO CODE FOR STRIPE APPROACH

```
Class Mapper
        Method Map(docid a; doc d)
                for all term w in doc d do
                        H = new AssociativeArray()
                        for all term u in Neighbors(w) do
                                H{u} = H{u} + 1
                        Emit(term w; stripe H)
```

```
Class Reducer
        Method Reduce(term w; stripes[H1;H2;H3; …])
                Hf = new AssociativeArray()
                marginal = 0
                for all stripe H in stripes [H1;H2;H3; …]
do
                        for all term u in H do
                                Hf{u} = Hf{u} + H{u}
                                marginal = marginal + H{u}

for all term u in Hf do
        Hf{u} ← Hf{u} / marginal
Emit(term w; stripe Hf)
```

JAVA CODE FOR STRIPE APPROACH

```
public void map(LongWritable offset, Text lineText, Context context)
throws IOException, InterruptedException {
            LOG.info("Starting mapping");
            if (lineText != null) {
                  String[] listTerm = lineText.toString().split("\\s+");
                  if (listTerm != null) {
                        for (int i = 0; i < listTerm.length - 1; i++) {
                              String currentTerm = listTerm[i];
                              MapWritable stripes = new MapWritable();
                              for (int j = i+1; j < listTerm.length; j++)
{
                                    if (currentTerm.equals(listTerm[j]))
                                          break;
                                    Text curNeighbor = new
Text(listTerm[j]);
                                    if (stripes.containsKey(curNeighbor))
{
                                          int counter =
((IntWritable)stripes.get(curNeighbor)).get();
                                          counter++;
                                          stripes.put(curNeighbor, new
IntWritable(counter));
                                    } else {
                                          stripes.put(curNeighbor, one);
                                    }
                              }
                              LOG.info("<Term, stripes> = (" + currentTerm
+ ", " + Utilities.mapWritableToText(stripes) + ")");
                              context.write(new
CrystalBallText(currentTerm), stripes);
                        }
                  }
            }
      }
```

```
public void reduce(Text term, Iterable<MapWritable> stripesList, Context context)
                  throws IOException, InterruptedException {
            MapWritable listTermNeighbor = new MapWritable();
            Iterator<MapWritable> listStripes = stripesList.iterator();
            double stripeTotal = 0.0;
            while (listStripes.hasNext()) {
                  MapWritable stripe = listStripes.next();

                  for (Entry<Writable, Writable> entry : stripe.entrySet()) {
                        Text curNeighbor = (Text)entry.getKey();
                        if (listTermNeighbor.containsKey(curNeighbor)) {

                              int val1 = ((IntWritable)entry.getValue()).get();
                              double val2 =
((DoubleWritable)listTermNeighbor.get(curNeighbor)).get();
                              stripeTotal += val1;
                              double val = val1 + val2;
                              listTermNeighbor.put(curNeighbor, new DoubleWritable(val));
                        } else {
                              int curVal = ((IntWritable)entry.getValue()).get();
                              listTermNeighbor.put(curNeighbor, new DoubleWritable(curVal));
                              stripeTotal += curVal;
                        }
                  }
            }

            for (Entry<Writable,Writable> entry : listTermNeighbor.entrySet()) {
                  double curVal = ((DoubleWritable)entry.getValue()).get();
                  double frequencies = curVal/stripeTotal;
                  LOG.info("CurVal/StripeTotal = " + curVal + "/" + stripeTotal + " =>
Frequency = " + frequencies);
                  frequencies = Double.parseDouble(Utilities.formatDouble(frequencies));
                  entry.setValue(new DoubleWritable(frequencies));
            }

            LOG.info("<Term, listTerm> = (" + term + ", " +
Utilities.mapWritableToText(listTermNeighbor) + ")");
            context.write(term, Utilities.mapWritableToText(listTermNeighbor));
      }
```

RESULT OF STRIPE APPROACH

9	[(46, 1.0)]
10	[(34, 0.5), (12, 0.5)]
12	[(56, 0.18), (92, 0.18), (34, 0.36), (18, 0.09), (79, 0.09), (10, 0.09)]
16	[(56, 0.25), (46, 0.25), (28, 0.25), (9, 0.25)]
18	[(56, 0.12), (92, 0.12), (34, 0.25), (79, 0.12), (29, 0.12), (12, 0.25)]
26	[(56, 0.22), (39, 0.11), (46, 0.22), (28, 0.22), (16, 0.11), (9, 0.11)]
28	[(56, 0.17), (39, 0.17), (46, 0.33), (16, 0.17), (9, 0.17)]
29	[(56, 0.13), (92, 0.13), (34, 0.27), (18, 0.07), (79, 0.07), (10, 0.07), (12, 0.27)]
34	[(56, 0.25), (92, 0.17), (18, 0.08), (79, 0.08), (29, 0.08), (10, 0.08), (12, 0.25)]
39	[(56, 0.17), (28, 0.17), (46, 0.33), (16, 0.17), (9, 0.17)]
46	[(56, 0.25), (28, 0.25), (16, 0.25), (9, 0.25)]
56	[(39, 0.06), (92, 0.12), (46, 0.12), (28, 0.12), (34, 0.18), (16, 0.06), (29, 0.06), (10, 0.06), (9, 0.06), (12, 0.18)]
79	[(56, 0.2), (92, 0.2), (34, 0.2), (18, 0.2), (12, 0.2)]
92	[(34, 0.33), (10, 0.33), (12, 0.33)]

PSEUDO CODE FOR HYBRID APPROACH

```
Class Mapper
        method Initialize
                H = new AssociativeArray()

        method Map(docid a; doc d)
                for all term w in doc d do
                        for all term u in neighbor(w) do
                                H{pair(w; u)} ← H{pair(w;
u)} + 1

        method Close
                for all pair p in H do
                        Emit(pair p; count H{pair p})
```

```
Class Reducer
        method inititalize
                marginal = 0;
                H = new AssociativeArray ()  //key is term u, value is count c
                currentTerm= null;

        //this method will be called multiple times
        method Reduce(pair(w; u); counts[c1;c2; …])
                if (currentTerm == null) then
                        currentTerm = w;

        //when new term w encountered
        else if (currentTerm != w) then
                for all term u in H do
                        H{u} = H{u} / marginal
                Emit(term currentTerm, stripe H)
                //reset for new term
                marginal = 0;
                H = new AssociativeArray()
                currentTerm = w;

        for all count c in counts[c1;c2; …] do
                        H{u} = H{u} + c
                        marginal = marginal + c

method Close //for the last term w
        for all term u in H do
                H{u} = H{u} / marginal
        Emit(term currentTerm, stripe H)
```

JAVA CODE FOR HYBRID APPROACH

```
public void map(LongWritable offset, Text lineText, Context context) throws
IOException, InterruptedException {
            LOG.info("starting mapping");
            if (lineText != null) {
                  String[] listTerm = lineText.toString().split("\\s+");
                  if (listTerm != null) {
                        for (int i = 0; i < listTerm.length; i++) {
                              String currentTerm = listTerm[i];
                              for (int j = i + 1; j < listTerm.length; j++)
{
                                    if (currentTerm.equals(listTerm[j]))
                                          break;
                                    Pair pair = new Pair(currentTerm,
listTerm[j]);

                                    if (pairMap.containsKey(pair)) {
                                          int counter = pairMap.get(pair);
                                          counter++;
                                          pairMap.put(pair, counter);
                                    } else {
                                          pairMap.put(pair, one);
                                    }
                              }
                        }
                  }
            }

      }

public void cleanup(Context context) throws IOException, InterruptedException {
                  Enumeration<Pair> enumerator = pairMap.keys();
                  while (enumerator.hasMoreElements()) {
                        Pair p = enumerator.nextElement();
                        LOG.info("<Pair, value> = (" + p.getTerm1() + ", " + p.getTerm2() + ") " +
pairMap.get(p));
                        context.write(p, new IntWritable(pairMap.get(p)));
      }
      }
```

```
@Override
public void reduce(Pair key, Iterable<IntWritable> counts, Context context)
                        throws IOException, InterruptedException {
                  if (currentTerm == null)
                        currentTerm = key.getTerm1();
                  else if (!currentTerm.equals(key.getTerm1())) {
                        for (Entry<Writable, Writable> entry : listNeighbor.entrySet()) {
                              double value = ((DoubleWritable) entry.getValue()).get();
                              double frequency = value / marginal;
                              frequency = Double.parseDouble(Utilities.formatDouble(frequency));
                              LOG.info("Pair = " + key + " value/marginal = " + value + "/" + marginal + " =>
frequency = " + frequency);
                              entry.setValue(new DoubleWritable(frequency));
                        }
                        context.write(new Text(currentTerm), Utilities.mapWritableToText(listNeighbor));
                        // reset for the new term
                        marginal = 0;
                        currentTerm = key.getTerm1();
                        listNeighbor = new MapWritable();
                  }
                  Iterator<IntWritable> listCount = counts.iterator();
                  Text curNeighbor = new Text(key.term2);
                  while (listCount.hasNext()) {
                        int count = listCount.next().get();
                        marginal += count;
                        if (listNeighbor.containsKey(curNeighbor)) {
                              double curVal = ((DoubleWritable)listNeighbor.get(curNeighbor)).get();
                              double newVal = curVal + count;
                              listNeighbor.put(curNeighbor, new DoubleWritable(newVal));
                        } else {
                              listNeighbor.put(curNeighbor, new DoubleWritable(count));
                        }
                  }
      }

@Override
public void cleanup(Context context) throws IOException, InterruptedException {
                  // NOTHING
                  LOG.info("Closing at Reducer");
                  for (Entry<Writable, Writable> entry : listNeighbor.entrySet()) {
                        double value = ((DoubleWritable) entry.getValue()).get();
                        double frequency = value / marginal;
                        frequency = Double.parseDouble(Utilities.formatDouble(frequency));
                        LOG.info("Pair = " + entry.getKey() + " value/marginal = " + value + "/" + marginal + " =>
frequency = " + frequency);
                        entry.setValue(new DoubleWritable(frequency));
                  }
                  LOG.info("<Term, Stripes> = (" + currentTerm + ", " + Utilities.mapWritableToText(listNeighbor) +
```

RESULT OF HYBRID APPROACH

9	[(46, 1.0)]
10	[(34, 0.5), (12, 0.5)]
12	[(56, 0.18), (92, 0.18), (34, 0.36), (18, 0.09), (79, 0.09), (10, 0.09)]
16	[(56, 0.25), (46, 0.25), (28, 0.25), (9, 0.25)]
18	[(56, 0.12), (92, 0.12), (34, 0.25), (79, 0.12), (29, 0.12), (12, 0.25)]
26	[(56, 0.22), (39, 0.11), (46, 0.22), (28, 0.22), (16, 0.11), (9, 0.11)]
28	[(56, 0.17), (39, 0.17), (46, 0.33), (16, 0.17), (9, 0.17)]
29	[(56, 0.13), (92, 0.13), (34, 0.27), (18, 0.07), (79, 0.07), (10, 0.07), (12, 0.27)]
34	[(56, 0.25), (92, 0.17), (18, 0.08), (79, 0.08), (29, 0.08), (10, 0.08), (12, 0.25)]
39	[(56, 0.17), (28, 0.17), (46, 0.33), (16, 0.17), (9, 0.17)]
46	[(56, 0.25), (28, 0.25), (16, 0.25), (9, 0.25)]
56	[(39, 0.06), (92, 0.12), (46, 0.12), (28, 0.12), (34, 0.18), (16, 0.06), (29, 0.06), (10, 0.06), (9, 0.06), (12, 0.18)]
79	[(56, 0.2), (92, 0.2), (34, 0.2), (18, 0.2), (12, 0.2)]
92	[(34, 0.33), (10, 0.33), (12, 0.33)]

COMPARISON

	Pair	Stripe	Hybrid
Map input records	3	3	3
Map output records	103	27	84
Map output bytes	1209	1055	1001
Time spent in maps (ms)	37753	44804	43440
Reduce input groups	90	14	71
Reduce Shuffle bytes	1433	1127	1187
Reduce input records	103	27	84
Reduce output records	71	14	14
Time spent in reduces (ms)	6078	5174	6757
Total time spent (ms)	**43831**	**49978**	**50197**

There is a will, there is a way!

Thank you!