SOLUTIONS

Exercises on Block1:
Map-Reduce
Retrieval Evaluation
Clustering

Advanced Search Techniques for Large Scale Data Analytics

Pavel Zezula and Jan Sedmidubsky
Masaryk University
http://disa.fi.muni.cz

Map-Reduce (1) – Assignment

- Suppose our input data to a map-reduce system are integer values (the keys are not important)
  - The map function takes an integer *i* and produces pairs (*p, i*) such that *p* is a prime divisor of *i*
    - Example: *map*('any_key', 12) = [(2,12), (3,12)]
  - The reduce function is addition
    - Example: *reduce*(*p*, [*i* , *i* , ..., *i*]) is (*p*, *i* + *i* + ... + *i*)
- Compute the output, if the input is the set of integers 15, 21, 24, 30, 49

Map-Reduce (1) – Recap

- **Map-reduce input:** a set of key-value pairs
- Programmer specifies two methods:
  - **Map(k, v)** → <k’, v’>*
    - Takes a key-value pair and outputs a set of key-value pairs
      - E.g., key is the filename, value is a single line in the file
    - There is one Map call for every *(k,v)* pair
  - **Reduce(k’, <v’>*)** → <k’, v’’>*
    - **All values *v’* with same key *k’* are reduced together and processed in *v’* order**
    - There is **one** Reduce function call per unique key *k’*

Map-Reduce (1) – Solution

- Map functions:
  - *map*('any_key', 15) = [(3, 15), (5, 15)]
  - *map*('any_key', 21) = [(3, 21), (7, 21)]
  - *map*('any_key', 24) = [(2, 24), (3, 24)]
  - *map*('any_key', 30) = [(2, 30), (3, 30), (5, 30)]
  - *map*('any_key', 49) = [(7, 49)]

- Reduce functions:
  - *reduce*(2, [24, 30]) = (2, 54)
  - *reduce*(3, [15, 21, 24, 30]) = (3, 90)
  - *reduce*(5, [15, 30]) = (5, 45)
  - *reduce*(7, [21, 49]) = (7, 70)

- **Output**: (2, 54), (3, 90), (5, 45), (7, 70)

Map-Reduce (2) – Assignment

- Suppose we have the following relations R, S:

R

A	B
0	1
1	2
2	3

S

B	C
0	1
1	2
2	3

- Apply the natural join algorithm
  - Apply the Map function to the tuples of relations
  - Construct the elements that are input to the Reduce function

Map-Reduce (2) – Recap

- Natural-join algorithm
  - Finding tuples that agree on common attributes, i.e., only the attribute B is in both relations R and S
  - Description of natural-join algorithm is in textbook in Section 2.3.7

Map-Reduce (2) – Solution

- Map functions – for each tuple (a, b) of R, the key-value pair (b, (R, a)) is produced and, analogically, for each tuple (b, c) of S, the pair (b, (S, c)) is created:
  - **R**: **S**:
  - *map*(R, (0, 1)) = (1, (R, 0)) *map*(S, (0, 1)) = (0, (S, 1))
  - *map*(R, (1, 2)) = (2, (R, 1)) *map*(S, (1, 2)) = (1, (S, 2))
  - *map*(R, (2, 3)) = (3, (R, 2)) *map*(S, (2, 3)) = (2, (S, 3))

- Based on the 4 different keys as the result of all the *map* calls, the following elements are input to the 4 reduce functions:
  - (0, [(S, 1)]) *reduce*(0, [(S, 1)]) = {}
  - (1, [(R, 0), (S, 2)]) *reduce*(1, [(R, 0), (S, 2)]) = {(0, 1, 2)}
  - (2, [(R, 1), (S, 3)]) *reduce*(2, [(R, 1), (S, 3)]) = {(1, 2, 3)}
  - (3, [(R, 2)]) *reduce*(3, [(R, 2)]) = {}

Map-Reduce (3) – Assignment

- Design MapReduce algorithms that take a very large file of integers and produce as output:
  1) The largest integer;
  2) The average of all the integers;
  3) The same set of integers, but with each integer appearing only once;
  4) The count of the number of distinct integers in the input.
- Suppose that the file is divided into parts that can be read in parallel by map functions

Map-Reduce (3) – Recap

- Example of algorithm for counting words

```
map(key, value):
// key: document name; value: text of the document
   for each word w in value:
      emit(w, 1)

reduce(key, values):
// key: a word; value: an iterator over counts
      result = 0
      for each count v in values:
            result += v
      emit(key, result)
```

Map-Reduce (3) – Solution 1/4

- 1) The largest integer
  - The idea is to compute a local maximum independently within each map function and then compute the global maximum within a **single** reducer – ensured by using the same “max” key within all map-function calls

```
map(file_id, iterator_over_numbers)
     max_local = MIN_INTEGER
     for each number n in interator_over_numbers
              if (n > max_local)
                       max_local = n
     emit(‘max’, max_local)

reduce(key, iterator_over_all_max_values)
     max_total = MIN_INTEGER
     for each number n in iterator_over_all_max_values
              if (n > max_total)
                       max_total = n
     emit(‘max’, max_total)
```

Map-Reduce (3) – Solution 2/4

- 2) The average of all the integers
  - The idea is to compute a local sum and count independently within each map function and then compute the global average within a **single** reducer – ensured by using the same “avg” key within all map-function calls

```
map(file_id, iterator_over_numbers)
      sum_local = 0
      count_local = 0
      for each number n in interator_over_numbers
                sum_local += n
                count_local += 1
      emit(‘avg’, (sum_local, count_local))
reduce(key, iterator_over_sum_count_pairs)
      sum_total = 0
      count_total = 0
      for each pair (sum_local, count_local) in iterator_over_sum_count_pairs
                sum_total += sum_local
                count_total += count_local
      emit(‘avg’, sum_total/count_total)
```

Map-Reduce (3) – Solution 3/4

- 3) The same set of integers, but with each integer appearing only once
  - The idea is to send each specific number to a single reducer, thus guaranteeing that each reducer emits the given value only once

```
map(file_id, iterator_over_numbers)
    for each number n in interator_over_numbers
            emit(n, 1)

reduce(key, iterator_over_numbers)
    emit(key, 1)
```

Map-Reduce (3) – Solution 4/4

- 4) The count of the number of distinct integers in the input
  - The idea is to send all the different numbers to a single reducer that eliminates duplicates using the union operation and counts the values

```
map(file_id, iterator_over_numbers)
     number_set = {}
     for each number n in interator_over_numbers
             number_set = number_set ∪ {n}
     emit('count', number_set)

reduce(key, iterator_over_number_sets)
     total_number_set = {}
     for each number_set in iterator_over_number_sets
              total_number_set = total_number_set ∪ number_set
     emit('count', |total_number_set|)
```

Retrieval Evaluation (1) – Assignment

- The algorithm retrieves the six most convenient documents for each query. We focus on the first relevant document retrieved.
  1) Determine a convenient measure for this task
  2) Compute the measure on the following four query rankings with relevant/irrelevant objects:
     - $R_1 = \{d_7, d_5, d_3, d_8, d_1\}$
     - $R_2 = \{d_5, d_6, d_3, d_2, d_4\}$
     - $R_3 = \{d_9, d_3, d_4, d_8, d_5\}$
     - $R_4 = \{d_9, d_3, d_1, d_7, d_5\}$
  3) How can be the result value interpreted?

Retrieval Evaluation (1) – Recap

- Mean Reciprocal Rank (MRR)
  - A good metric for those cases in which we are interested in the first correct answer
  - MRR = an average over reciprocal rankings *RR*
  - Definition of *RR*:
    - R_i: ranking relative to a query q_i
    - $S_{correct(R_i)}$: position of the first correct answer in R_i
    - S_h: threshold for ranking position
    - Then, the reciprocal rank $RR(R_i)$ for query q_i is:

$$RR(\mathcal{R}_i) = \begin{cases} \frac{1}{S_{correct}(\mathcal{R}_i)} & \text{if } S_{correct}(\mathcal{R}_i) \leq S_h \\ 0 & \text{otherwise} \end{cases}$$

Retrieval Evaluation (1) – Solution

1) The Mean Reciprocal Rank (MRR) is the most convenient measure for this task
2) Results for individual rankings (RR_i):
   - $RR_1 = 0.25$
   - $RR_2 = 0.5$
   - $RR_3 = 0.33$
   - $RR_4 = 0$

   $MRR = 0.27$
3) The first correct answer is at the 3.7-th position within an algorithm ranking (1/0.27 = 3.7) on average

Retrieval Evaluation (2) – Assignment

- Assume the following two rankings of documents (for some query):
  - $R_1 = \{d_7, d_5, d_3, d_8, d_1\}$
  - $R_2 = \{d_5, d_8, d_3, d_1, d_7\}$
- Based on these rankings compute:
  - Spearman rank correlation coefficient
  - Kendall Tau coefficient

Retrieval Evaluation (2) – Recap

- The Spearman coefficient
  - The mostly used rank correlation metric
  - Based on the differences between the positions of the same document in two rankings
  - Definition:
    - $s_{1,j}$ be the position of a document d_j in ranking R_1
    - $s_{2,j}$ be the position of d_j in ranking R_2
    - K indicates the size of the ranked sets
    - $S(R_1, R_2)$ is the Spearman rank correlation coefficient

$$\boxed{S(\mathcal{R}_1, \mathcal{R}_2) = 1 - \frac{6 \times \sum_{j=1}^{K} (s_{1,j} - s_{2,j})^2}{K \times (K^2 - 1)}}$$

Retrieval Evaluation (2) – Solution 1

- $(s_{1,d_7} - s_{2,\,d_7})^2 = 16$
- $(s_{1,d_5} - s_{2,\,d_5})^2 = 1$
- $(s_{1,d_3} - s_{2,\,d_3})^2 = 0$
- $(s_{1,d_8} - s_{2,\,d_8})^2 = 4$
- $(s_{1,d_1} - s_{2,\,d_1})^2 = 1$

- Spearman coefficient:
  - 1 – [6 * (16 + 1 + 0 + 4 + 1) / 120] = **–0.1**

Retrieval Evaluation (2) – Recap

- The Kendall Tau coefficient
  - When we think of rank correlations, we think of how two rankings tend to vary in similar ways
  - Consider two documents d_j and d_k and their positions in rankings R_1 and R_2
  - Further, consider the differences in rank positions for these two documents in each ranking, i.e.,
    - $s_{1,k} - s_{1,j}$
    - $s_{2,k} - s_{2,j}$
  - If these differences have the same sign, we say that the document pair (d_k, d_j) is **concordant** (**C**) in both rankings; if they have different signs, it is **discordant** (**D**)

Retrieval Evaluation (2) – Recap

- The Kendall Tau coefficient
  - Definition:
    - $\Delta(R_1, R_2)$: number of discordant document pairs in R_1 and R_2
    - K: the size of the ranked sets

$$\tau(R_1, R_2) = 1 - \frac{2 \times \Delta(R_1, R_2)}{K(K-1)}$$

Retrieval Evaluation (2) – Solution 2

- R_1:
  - $(d_7, d_5), (d_7, d_3), (d_7, d_8), (d_7, d_1)$ => D D D D
  - $(d_5, d_3), (d_5, d_8), (d_5, d_1)$ => C C C
  - $(d_3, d_8), (d_3, d_1)$ => D C
  - (d_8, d_1) => C
- R_2:
  - $(d_5, d_8), (d_5, d_3), (d_5, d_1), (d_5, d_7)$ => C C C D
  - $(d_8, d_3), (d_8, d_1), (d_8, d_7)$ => D C D
  - $(d_3, d_1), (d_3, d_7)$ => C D
  - (d_1, d_7) => D

- $\Delta(R_1, R_2) = 10$
- Kendall Tau coefficient:
  - 1 – [(2 * 10) / 20] = **0**

Clustering (1) – Assignment

- The Sum Squared Error (SSE) is a common measure of the quality of a cluster
  - Sum of the squares of the distances between each of the points of the cluster and the centroid
- Sometimes, we decide to split a cluster in order to reduce the SSE
  - Suppose a cluster consists of the following three points: (9,5), (2,2) and (4,8)
  - Calculate the reduction in the SSE if we partition the cluster optimally into two clusters

Clustering (1) – Solution

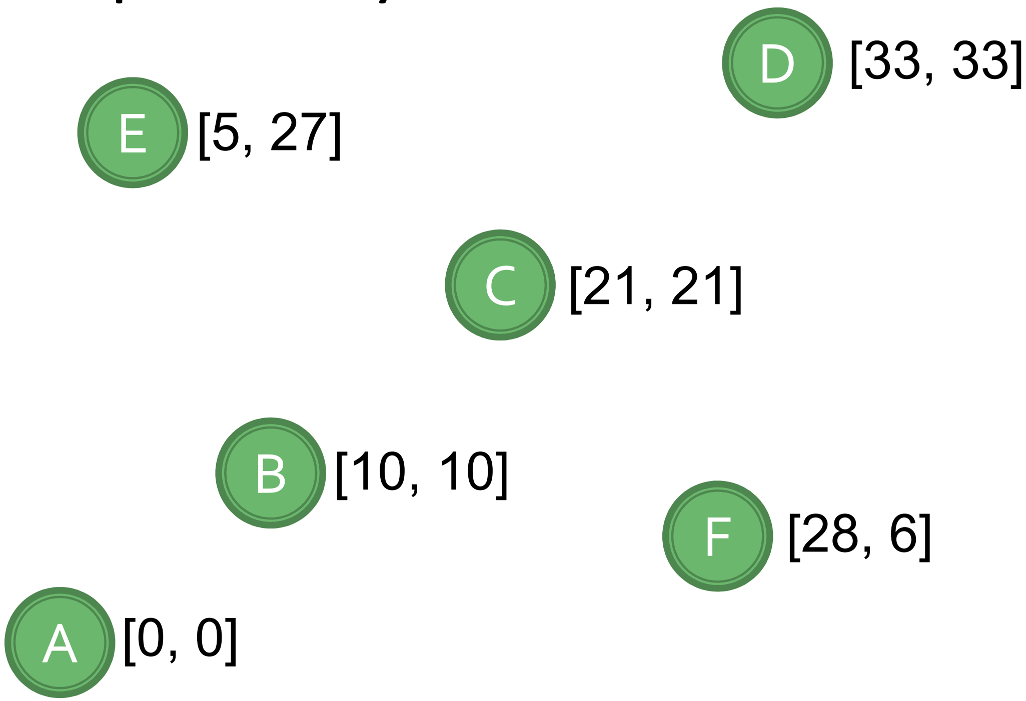
- Centroid of points is detemined by averaging the values in each dimension independently => centroid of that three points: (5,5)
  - $[(9,5) - (5,5)]^2 = 16$ $[(4,8) - (5,5)]^2 = 10$ $[(2,2) - (5,5)]^2 = 18$
  - => $SSE = 16 + 10 + 18 = 44$

- Then, we group the closest two points, i.e., points (9,5) and (4,8), to one cluster and compute its centroid: (6.5,6.5)
  - $[(9,5) - (6.5,6.5)]^2 = 8.5$ $[(4,8) - (6.5,6.5)]^2 = 8.5$ => $SSE_1 = 8.5 + 8.5 = 17$
- The second cluster has only one point, which is also centroid
  - $[(2,2) - (2,2)]^2 = 0$ => $SSE_2 = 0$
- => $SSE = SSE_1 + SSE_2 = 17 + 0 = 17$

- The reduction in the SSE:
  - 44 – 17 = **27**

Clustering (2) – Assignment

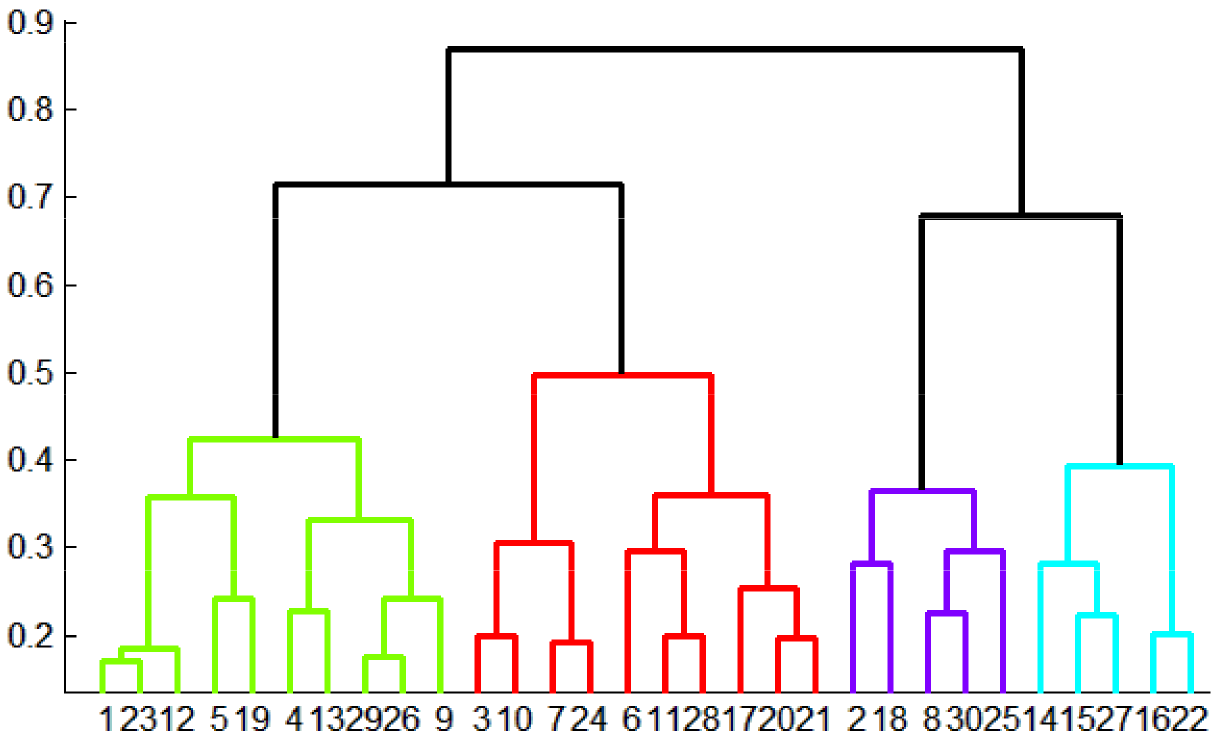
- Perform a hierarchical clustering on points A–F
  - Using the centroid proximity measure



  - There is a tie for which pair of clusters is closest. Follow both choices and identify the clusters.

Clustering (2) – Recap

- Hierarchical clustering
  - Key operation – repeatedly combine two nearest clusters



Clustering (2) – Solution

- Centroid proximity measure – distance between two clusters is the distance between their centroids

1) {A, B} with centroid at (5,5)
2) {C, F} with centroid at (24.5,13.5)
3) Tie:
    - {A, B, C, F} with centroid at (14.75,9.25) => {A, B, C, E, F}, {D}
    - {C, D, F} with centroid at (27.33,20) => {A, B, E}, {C, D, F}