

# Mining massive Datasets WS 2017/18

## Problem Set 3

Rudolf Chrispens, Marvin, Daniela Schacherer

November 13, 2017

### Exercise 01

1. More information in Ex1.py. Implemented the pseudocode into a working .py algorithm. Could not increase the performance of the for loop. Couldn't understand how this would be done?
2. Solution for problemset "dataset-problemset3-ex1-2" can be found in 1\_2\_ProblemsetSolution.pdf the code used code was in Ex1.py.

```
numpyDataset = np.load("dataset-problemset3-ex1-2.npy")
print(numpyDataset)
```

3. Code is in Ex1.3.py. Time is recorded in seconds.

```
Time: 1.5244791507720947 N: 1
Time: 1.0207879543304443 N: 2
Time: 1.9531109333038330 N: 3
Time: 3.8049488067626953 N: 4
Time: 5.6916959285736080 N: 5
Time: 9.2186932563781740 N: 6
Time: 14.890535116195679 N: 7
Time: 22.376883983612060 N: 8
Time: 29.547118186950684 N: 9
Time: 44.000823020935060 N: 10
Time: 66.097368001937870 N: 11
Time: 73.463916063308720 N: 12
Time: 95.624380350112920 N: 13
Time: 122.46513080596924 N: 14
Time: 153.88598299026490 N: 15
Time: 191.58319568634033 N: 16
Time: 235.73992490768433 N: 17
Time: 286.40261006355286 N: 18
Time: 347.29994392395020 N: 19
```

## Exercise 02

### 1. MapReduce Pseudocode for k-means

The k-means algorithm consists of two steps. The first step computes for each mean  $\mu_i$  the set of points that are closest to it. In the second step new means are computed using the priorly determined sets. These two phases correspond to the Map- and the Reduce phase of MapReduce. The Map phase computes the squared distance to all means for each point  $x$  in the dataset and returns a key-value pair  $(i, (x,1))$  where  $i$  is the index of the mean with the smallest distance to point  $x$ . The Reduce phase then simply computes the sum of the vector points for each key.

Pseudocode:

- **Map** for every point  $x$ : return  $(\operatorname{argmin}_i(\|x - \mu_i\|), (x,1))$
- **Reduce** for every elements with key  $i$ : return  $(i, (x + y, s + t))$  with  $x$  and  $y$  being the data points and  $s$  and  $t$  being the counts.

### 2. MapReduce Pseudocode for Inverted Indexing

Inverted

- **Map**: for every keyword in the given list the Mapper should perform the following:  
if  $\text{keyword}$  in  $\text{text}_i$ : return  $(\text{keyword}, \text{doc}_i)$
- **Reduce**: for every keyword add the documents indices to a list and finally return  $(\text{keyword}, [\text{doc}_i, \text{doc}_j, \dots])$

The proposed pseudocode would probably not scale well with the number of keywords, because the mapper has to run a search algorithm for every keyword. If the number of mappers is limited, a mapper has to perform multiple searches and thus the time for the complete calculation will get larger.

### 3. MapReduce Pseudocode

When one dataset is small and every mapper has access to it the joining can already be part of the mapping phase. Let  $R$  with tuples  $(a,b)$  and  $S$  with tuples  $(b,c)$  be the datasets, and  $R$  is the smaller one. We want to join on  $b$ . Every mapper gets tuples from  $S$  in this form:  $(S,a,b)$

- **Map**: for every tuple of  $R$ :  
if  $b$  in  $(S,a,b)$ : return  $(a,b,c)$
- **Reduce**: in the reduce phase we now only have to collect all the joined tuples.

## Exercise 03

a)

In Spark the input of every Transformation (map) and Action (reduce) is a RDD. With a Transformation the output is again a RDD and with an action the output can have different types. The data in the RDD can be different, key-value pairs is not mandatory.

In MapReduce the output is not a RDD but a set of key-value pairs. The output is again a set of

key-value pairs. Depending on the map step the output has a different number of data. At the reduce step it is basically the same. Input and output are both key-value pairs.

**b)**

I wrote a spark application that counts the number of characters in a textfile. The Code is in the submitted *ex3problem3b.py* file that uses the textfile *lorem\_ipsum.txt*.

Program code:

```
from pyspark import SparkContext

if __name__ == "__main__":
    sc = SparkContext(appName="PythonKMeans")

    lines = sc.textFile("lorem_ipsum.txt")
    lineLengths = lines.map(lambda s: len(s))
    totalLength = lineLengths.reduce(lambda a, b: a + b)

    print(totalLength)

    sc.stop()
```

The output of the Program

591

## Exercise 04

**a)**

Let  $m$  be the size of the matrix and  $v$  the size of the vector. Both gets divided in  $n$  different parts. Each map-task gets  $m/n$  and  $v/n$  parts and this  $n$  times to multiply all of the matrix. Because of that we get a communication cost of  $O(n * (m/n + v/n))$ .

**b)**

Let  $n$  be the number of tuples. Then the answer is just  $O(n)$  because the only input are the tuples.