big_ds

June 21, 2020

1 "Big" Data Science

1.0.1 Guest Lecture in Introduction to Data Science at TAU / Prof. Saharon Rosset

1.0.2 Giora Simchoni

1.0.3 June 22nd, 2020

2 What is Small? What is Big?

These definitions are constantly changing.

(1) "Everything processed in Excel is small data." (Rufus Pollock, The Guardian)

(2) "[Big Data] is data so large it does not fit in main memory" (Leskovec et al., Mining of Massive Datasets)

Or maybe we should define the size of our data according how easy it is to process and understand it?

(3) "[Small Data is] data that has small enough size for human comprehension." (`jWork.ORG`)

(4) "data sets that are too large or complex for traditional data-processing application software to adequately deal with" (Wikipedia)

We'll talk about solutions to (2) and (4) today.

3 When data is too big to fit in RAM...

3.0.1 You want Regression? Deep Learning? Try computing an average!

3.0.2 Distributed File System

- a new form of file system
- data is distributed over computing clusters (a collection of hundreds to thousands of "computer nodes")

3.0.3 MapReduce

- a new software methodology which knows how to "speak" to data on DFS
- from computing `SELECT Country, COUNT(*) FROM table_name GROUP BY Country` to Machine Learning

4 Distributed File System (DFS)

- Typically: HDFS (Hadoop Distributed File System), another open source project by Apache
- "Computers" (nodes) are stacked in racks of 8-64 size each.
- Nodes on the same rack are connected, and racks are connected by a network or "switch"

5 Probably the only time this image is called for

6 Fact: Computers Fail

- If a node's disk has 1 in a million chance of failing the next minute, what is the chance of (at least one) failure on a cluster of 10000 nodes?
- Solution: chunk your data into say 64MB-size chunks, and replicate on say 3 nodes, on different racks

7 MapReduce

- Typically: Hadoop by Apache
- Three main features:
  1. Parallel computations, exploiting the cluster
  2. Tolerant to hardware failures
  3. All you need is a Map function and a Reduce function

7.1 The Mapper

A function which will take one or more file chunks and sum them up to (key, value) mapping. The mappings from a few mappers are then grouped to (key, [values]) mapping, and sorted by key.

7.2 The Reducer

A function which will take one of those grouped mapping and aggregate them in some way. A reducer typically handles one key at a time.

8 The Classic MapReduce Example: Word Count

9 Word Count: The Mapper

10 Word Count: MapReduce then groups and sorts, regardless of input:

11 Word Count: The Reducer

12 MapReduce Functional View

13 MapReduce Architectural View

Question: why would we prefer a smaller number of Reducer workers?

14 MapReduce: Failure is an Option

What happens when...

- The Master is down?
- A Map worker is down?
- A Reduce worker is down?

15 MapReduce: In Action (almost)

[1]:
```
def mapper(text):
    list_of_key_values = []
    for word in text.split(' '):
        if word != '':
            list_of_key_values.append((word, 1))
    return list_of_key_values

def reducer(single_key_values):
    key, values = single_key_values
    return key, sum(values)
```

[2]:
```
mapper('O Captain! my Captain! our fearful trip is done,')
```

[2]:
```
[('O', 1),
 ('Captain!', 1),
 ('my', 1),
 ('Captain!', 1),
 ('our', 1),
 ('fearful', 1),
 ('trip', 1),
 ('is', 1),
```

```
 ('done,', 1)]
```

[3]:
```
mapper('The ship has weather'd every rack, the prize we sought is won,')
```

[3]:
```
[('The', 1),
 ('ship', 1),
 ('has', 1),
 ('weather'd', 1),
 ('every', 1),
 ('rack,', 1),
 ('the', 1),
 ('prize', 1),
 ('we', 1),
 ('sought', 1),
 ('is', 1),
 ('won,', 1)]
```

[4]:
```
mapper('The port is near, the bells I hear, the people all exulting')
```

[4]:
```
[('The', 1),
 ('port', 1),
 ('is', 1),
 ('near,', 1),
 ('the', 1),
 ('bells', 1),
 ('I', 1),
 ('hear,', 1),
 ('the', 1),
 ('people', 1),
 ('all', 1),
 ('exulting', 1)]
```

[5]:
```
# remember Hadoop will group map output for you!
# remember single key for single reducer, though single reducer can handle a␣
 ↪few keys
reducer(('Captain', [1, 1]))
```

[5]:
```
('Captain', 2)
```

16 Running a Hadoop Job - Live Demo

```
bash-4.1$ bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.7.1.jar \
-file /home/mapper.py -mapper /home/mapper.py \
-file /home/reducer.py -reducer /home/reducer.py \
-input shakespeare -output output -numReduceTasks 4

packageJobJar: [/home/mapper.py, /home/reducer.py, /tmp/hadoop-unjar2660950065718379808/] [] /t
```

```
19/05/30 10:50:09 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
19/05/30 10:50:10 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
19/05/30 10:50:11 INFO mapred.FileInputFormat: Total input paths to process : 10
19/05/30 10:50:11 INFO mapreduce.JobSubmitter: number of splits:10
19/05/30 10:50:12 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1559226271713_000
19/05/30 10:50:12 INFO impl.YarnClientImpl: Submitted application application_1559226271713_000
19/05/30 10:50:12 INFO mapreduce.Job: The url to track the job: http://2cd2e5b21407:8088/proxy/
19/05/30 10:50:12 INFO mapreduce.Job: Running job: job_1559226271713_0005
19/05/30 10:50:18 INFO mapreduce.Job: Job job_1559226271713_0005 running in uber mode : false
19/05/30 10:50:18 INFO mapreduce.Job:  map 0% reduce 0%
19/05/30 10:50:35 INFO mapreduce.Job:  map 60% reduce 0%
19/05/30 10:50:52 INFO mapreduce.Job:  map 90% reduce 0%
19/05/30 10:50:53 INFO mapreduce.Job:  map 90% reduce 15%
19/05/30 10:50:54 INFO mapreduce.Job:  map 90% reduce 23%
19/05/30 10:50:59 INFO mapreduce.Job:  map 100% reduce 32%
19/05/30 10:51:00 INFO mapreduce.Job:  map 100% reduce 50%
19/05/30 10:51:02 INFO mapreduce.Job:  map 100% reduce 100%
19/05/30 10:51:03 INFO mapreduce.Job: Job job_1559226271713_0005 completed successfully

bash-4.1$ bin/hadoop fs -ls output
Found 5 items
-rw-r--r--   1 root supergroup          0 2019-05-30 10:51 output3/_SUCCESS
-rw-r--r--   1 root supergroup      78211 2019-05-30 10:51 output3/part-00000
-rw-r--r--   1 root supergroup      77696 2019-05-30 10:51 output3/part-00001
-rw-r--r--   1 root supergroup      77111 2019-05-30 10:51 output3/part-00002
-rw-r--r--   1 root supergroup      78458 2019-05-30 10:51 output3/part-00003
```

17 MapReduce: Plausible Exercise

Facebook has 2.5 billion users.

Suppose (and we're REALLY simpifying things here) files in HDFS contain each a list of integers, which are the number of friends each user has.

Write a Mapper and a Reducer to compute the maximum number of friends a user has.

[6]:
```python
def mapper(list_of_ints):
    ### Your code here
    pass

def reducer(single_key_values):
    ### Your code here
    pass
```

18 One Solution

[7]:
```
def mapper(list_of_ints):
    list_of_key_values = [('whatever', max(list_of_ints))]
    return list_of_key_values

def reducer(single_key_values):
    key, values = single_key_values
    return key, max(values)
```

19 MapReduce: Another Plausible Exercise

Which of the following will NOT be "naturally-fitting" to the MapReduce methodology? More than 1 answer is correct.

a. Getting the average annual income by country from a huge table containing each adult person in the world and his/her annual income

b. Getting the median annual income by country from a huge table containing each adult person in the world and his/her annual income

c. Multiplying a distributed huge 1-trillion x 1-million matrix with a 1-million long vector

d. Multiplying a distributed huge 1-trillion x 1-trillion matrix with a 1-trillion long vector

e. Fitting a Random Forests model with 1000 trees to a medium-size dataset which can fit in a single node RAM

f. Fitting Gradient Boosted Trees model with 1000 trees to a medium-size dataset which can fit in a single node RAM

g. Getting the average income by settlement from a file containing each person in Israel, his/her city and income

20 But surely you don't need to write a Mapper and a Reducer each time you want a simple average, let alone fit a Random Forest model?...

21 Hadoop Ecosystem

22 Pig Script Example

```
A = LOAD 'data' AS (f1:int,f2:int,f3:int);

DUMP A;
(1,2,3)
(4,2,1)
(8,3,4)
(4,3,3)
```

```
(7,2,5)
(8,4,3)

B = GROUP A BY f1;

DUMP B;
(1,{(1,2,3)})
(4,{(4,2,1),(4,3,3)})
(7,{(7,2,5)})
(8,{(8,3,4),(8,4,3)})

X = FOREACH B GENERATE COUNT(A);

DUMP X;
(1L)
(2L)
(1L)
(2L)
```

23 Hive Script Example

```
SELECT COUNT(*) FROM table2;
```

24 Spark

- Spark by Apache is a "unified analytics engine for large-scale data processing."
- Write (single) programs in Java, Scala, Python (PySpark), R (SparkR), and SQL.
- Works not only with Hadoop data! Also as a standalone (you can install Spark on your laptop), on streaming data, Kubernetes and more
- Especially suitable for Machine Learning
- But for Deep Learning you're probably already using TensorFlow by Google or Keras on top of it
- And for parallel data manipulations utilizing a multicore CPU I'd also look at Dask by Anaconda

25 Spark: Greatest advantage - In-memory computation

- Hadoop: Tasks are distributed among the nodes of a cluster, which in turn load/save data on disk
- Spark: Saves and loads the data in and from the RAM rather than from the disk

26 Spark: RDD and DataFrame

Every framework has its basic data objects or structures:

- Basic R has: vector, list, data.frame, matrix, factor
- Basic Python has: list, set, dictionary, tuple

- Tidyverse (R) has: tibble
- Pandas (Python) has: Series, DataFrame
- Spark has: RDD, DataFrame, DataSet (only in Scala/Java)

27 RDD (Resilient Distributed Dataset)

- Collection of elements (lines of text, tuples (rows) of table), that can be divided across multiple nodes in a cluster to run parallel processing
- "Resilient" = can automatically recover from node failures
- Immutable - you can't change a RDD, you can only **transform** it into another RDD (Map) or perform an **action** on it in some way (Reduce)
- Lazy Evaluation - nothing happens when you transform a RDD! Not until you perform an action, in which case Spark remembers all the transformations which have led your data to where it is (in a DAG), and moves your data along these "pipes" in an optimized way

[8]:
```
from pyspark import SparkContext

sc = SparkContext()
rdd = sc.parallelize([1,2,3,4,5])
rdd
```

[8]:
```
ParallelCollectionRDD[0] at readRDDFromFile at PythonRDD.scala:262
```

[9]:
```
rdd.collect()
```

[9]:
```
[1, 2, 3, 4, 5]
```

28 PySpark: Basic Transformations

[10]:
```
# map
rdd.map(lambda x: x + 1).collect()
```

[10]:
```
[2, 3, 4, 5, 6]
```

[11]:
```
# flatMap
rdd.flatMap(lambda x: range(1, x)).collect()
```

[11]:
```
[1, 1, 2, 1, 2, 3, 1, 2, 3, 4]
```

[12]:
```
# filter
rdd.filter(lambda x: x < 4).collect()
```

[12]:
```
[1, 2, 3]
```

[13]:
```
# glom
rdd.glom().collect()
```

[13]:
```
[[], [1], [], [2], [3], [], [4], [5]]
```

[14]:
```
# sortByKey
rdd.sortBy(lambda x: x, ascending=False).collect()
```

[14]:
```
[5, 4, 3, 2, 1]
```

29 PySpark: Basic Actions

[15]:
```
# take
rdd.take(2)
```

[15]:
```
[1, 2]
```

[16]:
```
# first
rdd.first()
```

[16]:
```
1
```

[17]:
```
# reduce
rdd.reduce(lambda a, b: a + b)
```

[17]:
```
15
```

[18]:
```
# fold
rdd.fold(1000, lambda a, b: a + b)
```

[18]:
```
9015
```

[19]:
```
# count, sum, mean, min, max, stdev, variance...
rdd.count()
```

[19]:
```
5
```

30 PySpark: Functional Programming (Chaining)

[20]:
```
# increase each value by 10 and sum
rdd.map(lambda x: x + 10).sum()
```

[20]:
```
65
```

[21]:
```
# group by remainder of division by 3,
# sum each group's values,
# sort RDD by remainder
# and filter by remainder smaller than 2
```

```
rdd.groupBy(lambda x: x % 3) \
    .mapValues(sum) \
    .sortByKey() \
    .filter(lambda x: x[0] < 2) \
    .collect()
```

[21]:
```
[(0, 3), (1, 5)]
```

31 PySpark: Word Count

[22]:
```
shakespeare = sc.textFile('hadoop_example/shakespeare/*')
```

[23]:
```
shakespeare.count()
```

[23]:
```
44640
```

[24]:
```
wc = shakespeare \
    .flatMap(lambda line: line.split(' ')) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda a, b: a + b)
```

[25]:
```
wc.top(10, key = lambda t: t[1])
```

[25]:
```
[('the', 6788),
 ('and', 5113),
 ('I', 5036),
 ('to', 4298),
 ('of', 3930),
 ('a', 3160),
 ('my', 2829),
 ('in', 2532),
 ('you', 2490),
 ('is', 2101)]
```

32 DataFrames

- (inspired by Pandas which is inspired by R)
- Collection of `Rows` under named `Columns`
- Especially suitable for tabular data
- Syntax very similar to Pandas
- Like RDD: Resilient, Distirbuted, Lazy evaluated, Immutable

[26]:
```
from pyspark.sql import SparkSession

spark = SparkSession(sc)
```

```
df = spark.createDataFrame(data=[(1, 'A'), (2, 'B'), (3, 'C')], schema=['id',␣
↪'name'])
df
```

[26]:
```
DataFrame[id: bigint, name: string]
```

[27]:
```
df.show()
```

```
+---+----+
| id|name|
+---+----+
|  1|   A|
|  2|   B|
|  3|   C|
+---+----+
```

33 PySpark: Basic DataFrame Manipulations (I)

[28]:
```
saheart = spark.read.csv('SAheart.csv', header = True, inferSchema = True)
saheart.count()
```

[28]:
```
462
```

[29]:
```
saheart.columns
```

[29]:
```
['row.names',
 'sbp',
 'tobacco',
 'ldl',
 'adiposity',
 'famhist',
 'typea',
 'obesity',
 'alcohol',
 'age',
 'chd']
```

[30]:
```
saheart = saheart.drop('row.names')
```

[31]:
```
saheart.show(2)
```

```
+---+-------+----+---------+-------+-----+-------+-------+---+---+
|sbp|tobacco| ldl|adiposity|famhist|typea|obesity|alcohol|age|chd|
+---+-------+----+---------+-------+-----+-------+-------+---+---+
|160|   12.0|5.73|    23.11|Present|   49|   25.3|   97.2| 52|  1|
```

```
|144|   0.01|4.41|    28.61| Absent|   55|  28.87|   2.06| 63|  1|
+---+-------+----+---------+-------+-----+-------+-------+---+---+
only showing top 2 rows
```

34 PySpark: Basic DataFrame Manipulations (II)

[32]:
```
saheart.select('famhist').distinct().show()
```

```
+-------+
|famhist|
+-------+
|Present|
| Absent|
+-------+
```

[33]:
```
saheart.groupby('famhist').agg({'chd': 'avg'}).show()
```

```
+-------+------------------+
|famhist|          avg(chd)|
+-------+------------------+
|Present|               0.5|
| Absent|0.23703703703703705|
+-------+------------------+
```

[34]:
```
saheart.stat.corr('obesity', 'alcohol')
```

[34]:
```
0.051619568611913025
```

35 PySpark: Processing DataFrame for Classification

[35]:
```
train, test = saheart.randomSplit([0.8, 0.2])
print(train.count())
print(test.count())
```

```
362
100
```

[36]:
```
from pyspark.ml.feature import RFormula

formula = RFormula(formula='chd ~ .', featuresCol='features', labelCol='label')

fit = formula.fit(train)
train_df = fit.transform(train)
```

```
train_df.show(1) # notice the two extra columns!
```

```
+---+-------+----+---------+-------+-----+-------+-------+---+---+-------------
------+-----+
|sbp|tobacco| ldl|adiposity|famhist|typea|obesity|alcohol|age|chd|
features|label|
+---+-------+----+---------+-------+-----+-------+-------+---+---+-------------
------+-----+
|101|   0.48|7.26|     13.0| Absent|   50|  19.82|   5.19| 16|
0|[101.0,0.48,7.26,…|  0.0|
+---+-------+----+---------+-------+-----+-------+-------+---+---+-------------
------+-----+
only showing top 1 row
```

[37]:
```
test_df = fit.transform(test)
```

36 PySpark: Logistic Regression (I)

[38]:
```
from pyspark.ml.classification import LogisticRegression

lr = LogisticRegression(maxIter=100, regParam=0.0)

lrModel = lr.fit(train_df)

print("Coefficients: " + str(lrModel.coefficients))
print("Intercept: " + str(lrModel.intercept))
```

```
Coefficients: [0.00129902161382201,0.07268020148984834,0.2174340350187219,0.0021
612609181482826,-0.7996643787786133,0.03229365612393295,-0.05282631703787926,0.0
017741654718499843,0.06347927905214099]
Intercept: -5.0743059336551175
```

[39]:
```
pred_df = lrModel.transform(test_df)
pred_df.show(2) # notice the extra columns!
```

```
+---+-------+----+---------+-------+-----+-------+-------+---+---+-------------
------+-----+--------------------+--------------------+----------+
|sbp|tobacco| ldl|adiposity|famhist|typea|obesity|alcohol|age|chd|
features|label|       rawPrediction|         probability|prediction|
+---+-------+----+---------+-------+-----+-------+-------+---+---+-------------
------+-----+--------------------+--------------------+----------+
|103|   0.03|4.21|    18.96| Absent|   48|  22.94|   2.62| 18|
0|[103.0,0.03,4.21,…|  0.0|[3.29608076815384…|[0.96429411302597…|
0.0|
|106|   1.08|4.37|    26.08| Absent|   67|  24.07|  17.74| 28|
1|[106.0,1.08,4.37,…|  1.0|[1.95018796785011…|[0.87546713634656…|
```

```
0.0|
+---+-------+----+---------+-------+-----+-------+-------+---+---+--------------
------+-----+--------------------+--------------------+----------+
only showing top 2 rows
```

37 PySpark: Logistic Regression (II)

[40]:
```
# assuming data is small
import numpy as np
y_pred = np.array([int(row.prediction) for row in pred_df.select('prediction').
 ↪collect()])
y_true = np.array([int(row.label) for row in pred_df.select('label').collect()])
print('Test accuracy: %.2f' % np.mean(y_pred == y_true))
```

```
Test accuracy: 0.68
```

[41]:
```
# for AUC
from pyspark.ml.evaluation import BinaryClassificationEvaluator

evaluator = BinaryClassificationEvaluator(rawPredictionCol='rawPrediction')
auc = evaluator.evaluate(pred_df)
print('Test AUC: %.2f' % auc)
```

```
Test AUC: 0.73
```

38 PySpark: Random Forest

[42]:
```
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

rf = RandomForestClassifier(numTrees=1000)

rfModel = rf.fit(train_df)

pred_df = rfModel.transform(test_df)

# when data isn't so small, get accuracy with␣
 ↪MulticlassClassificationEvaluator, but beware of the automatic threshold
evaluator = MulticlassClassificationEvaluator(metricName='accuracy')
accuracy = evaluator.evaluate(pred_df)
print('Test accuracy: %.2f' % accuracy)
```

```
Test accuracy: 0.68
```

[43]:
```
evaluator = BinaryClassificationEvaluator(rawPredictionCol='rawPrediction')
auc = evaluator.evaluate(pred_df)
print('Test AUC: %.2f' % auc)
```

```
Test AUC: 0.71
```