Mining massive databases

Diego Saez-Trumper

Data Science - UCU

About me

Currently:

- Senior Research Scientist at Wikimedia Foundation
- Visiting Scholar at Universitat Pompeu Fabra, Barcelona

Research at Universities

- UPF (Spain), Cambridge (UK), UFMG (Brazil)

Previous Industrial Experience:

- Yahoo, NTENT, QCRI, Eurecat

Conferences:

- KDD, CIKM, RecSys, IMC, Hypertext, COSN, WSDM, SIGIR, ICWSM

Mykola Trokhymovych

mykola.trokhymovych@upf.edu | trokhymovych.com

- PhD candidate @ Universitat Pompeu Fabra
    - Co-advised by Dr. Diego Saez-Trumper and Prof. Ricardo Baeza-Yates
    - Research topic: Knowledge Integrity in the Generative AI Era
- Head of AI and Data Science @ Theodora AI
- Research contractor @ Wikimedia Foundation

Previously:

- MSc. Data Science @ Ukrainian Catholic University
- Data scientist @ Ciklum, Jooble, Surprise

Conferences:ACL’24, KDD’23, CIKM’21

Which kind of problems we will work on in this course?

- How can you build a efficient SPAM filter?
  - Black list are to large keep in memory.
  - Classification needs to be done in real time
- How to identify a bot attack in Wikipedia?
  - Data streams are huge, you can’t keep in memory the information about each user.
  - Random sampling might lose relevant information.
- How to cluster a stream of new Tweets according to their topic in real time.
  - Topics are not known in advance.
  - It is not possible to see the full corpus everytime you receive a new tweet.

Syllabus

Chapter 1: Introduction to Big Data

- General overview of the course
- Hadoop and spark introduction
- Pig and Pyspark comparison
- Public (big) datasets and resources

Chapter 2: Processing Structured and Semi-structured large data

- Preprocessing User Generated Content and other semi-structured data.
- Introduction to PySpark.
- Scikit-learn at large scale
- Basics of machine learning with PySpark (mllib).

Syllabus

Chapter 3: Working with Data Streams

- Data Streams, how are they different from static data?
- Online vs Offline Algorithms.
- Applied ML in (large) streams.

Chapter 4: Graphs

- Introduction to graphs.
- Temporal graphs representation.
- Centrality measures for graphs (PageRank, Hits, etc).
- Communities / Clusters / Cliques in Graphs.
- Large graphs with NoSql databases.

Skills to be developed in this course

- Learn how to prepare large datasets for ML process.
- Learn the basics of ML mixing scikit-learn, spark and ML libraries.
- Text mining at large scale with PySpark.
- Graph mining with spark.
- Introduction to ML with Data Streams.

Evaluation

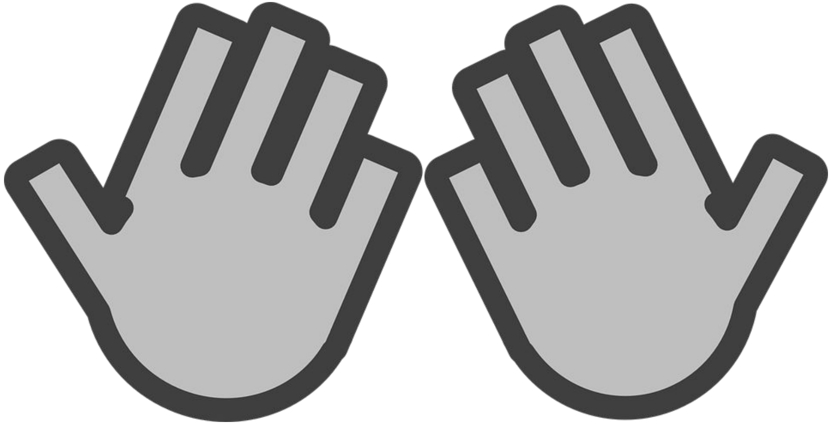
- 2 homeworks (groups of 3 to 4 students)
- Oral presentation

Course highlights

- Course is in **pySpark** (-> not in Scala)
- Open data
- Projects based evaluations
  - In the next session we will discuss about the projects.

Methodology

- ~~Move fast, break things~~
- ~~Don’t be evil~~
- **Deeply understand the basics - Play with Data!**
- **KISS**



Your responsibilities

- Ask questions!
- Deliver on time
- Understand what you do
    - Be able to explain your code
    - Justify your choices

Today

- ~~Course Description~~
- Background:
    - Quick introduction to ML and Massive Datasets
    - Overview of HDFS and Spark
- Hands on:
    - Hello world with PySpark
    - Basic data management with PySpark Dataframes
    - Processing text with PySpark

What is "Big" for an Algorithm?

- Concept of "Big Data" has been abused for commercial purposes.
- "Big data" projects can include datasets of couple of Gb, until Petabytes
- We need to chose the framework that we will use depending on the data characteristics

frameworks, algorithms implemented, community support, etc

Data size

Data Scientist ML tasks at companies

There are a set of different business cases that might include:

- Business and behavioral Analytics
- Recommender Systems
- Process optimization
- Etc.

From an ML perspective we can usually reduce our problems to:

- Classify
- Cluster
- Rank

Data mining approaches for large datasets

- Use distributed systems (e.g Hadoop, Spark) to pre-process and reduce the data, and later apply standard ML frameworks.
- Use distributed ML Frameworks (e.g ML-spark) to learn directly from Big Data.
- Use distributed systems to parallelize standard ML Frameworks (e.g. compare multiple scikit-learn models results)

Deep learning vs “shallow methods”

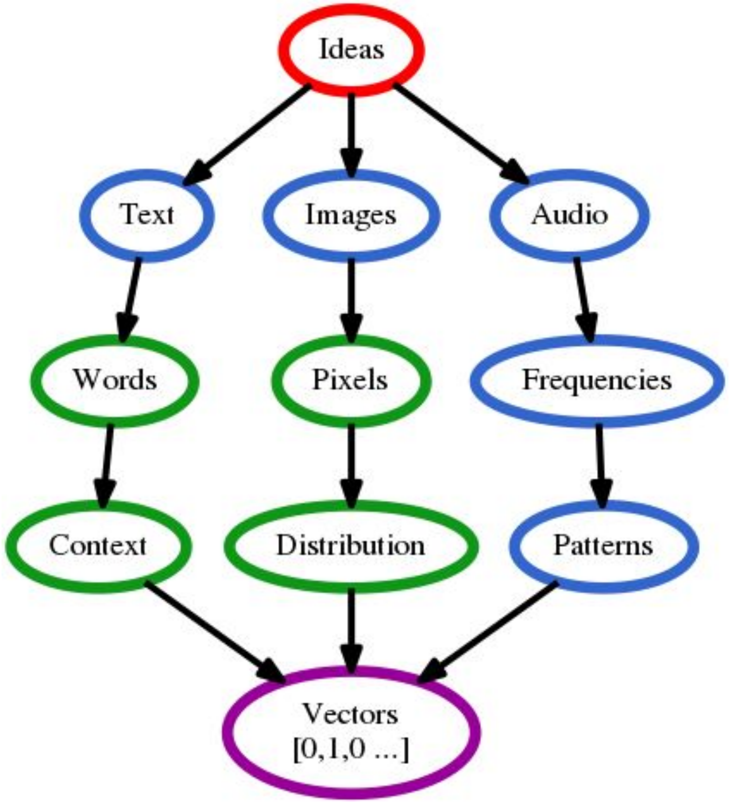
- Keep it simple.
- Depends on the needs and infrastructure you have.
- If the signal you are looking for is not in the data, don’t waste your time.

How “smart” is AI and ML?

- We shouldn’t expect an algorithm been very “smart”.
- Your algorithm is not likely to understand very complex abstractions
- The power of AI and ML is the amount of data that can be processed in a short span of time

Machines transform ‘reality’ in vectors

- AI takes a **simplified version of the “reality”.**
- Transforming text, images or audio to vectors, a lot of information is lost.
- As more abstract is the information, more complex is for the machine to understand.



Cloud computing providers

- Which infrastructure should I expect to use?
  - Depends on company size and culture

Data Cleaning / Munging / Wrangling

Raw data

After Data Munging

Data scientist expend most of their time in data cleaning

Collecting User Generated Content

- Facebook API: https://developers.facebook.com/
- Airbnb: http://insideairbnb.com/get-the-data.html
- Common Crawl: https://commoncrawl.org/

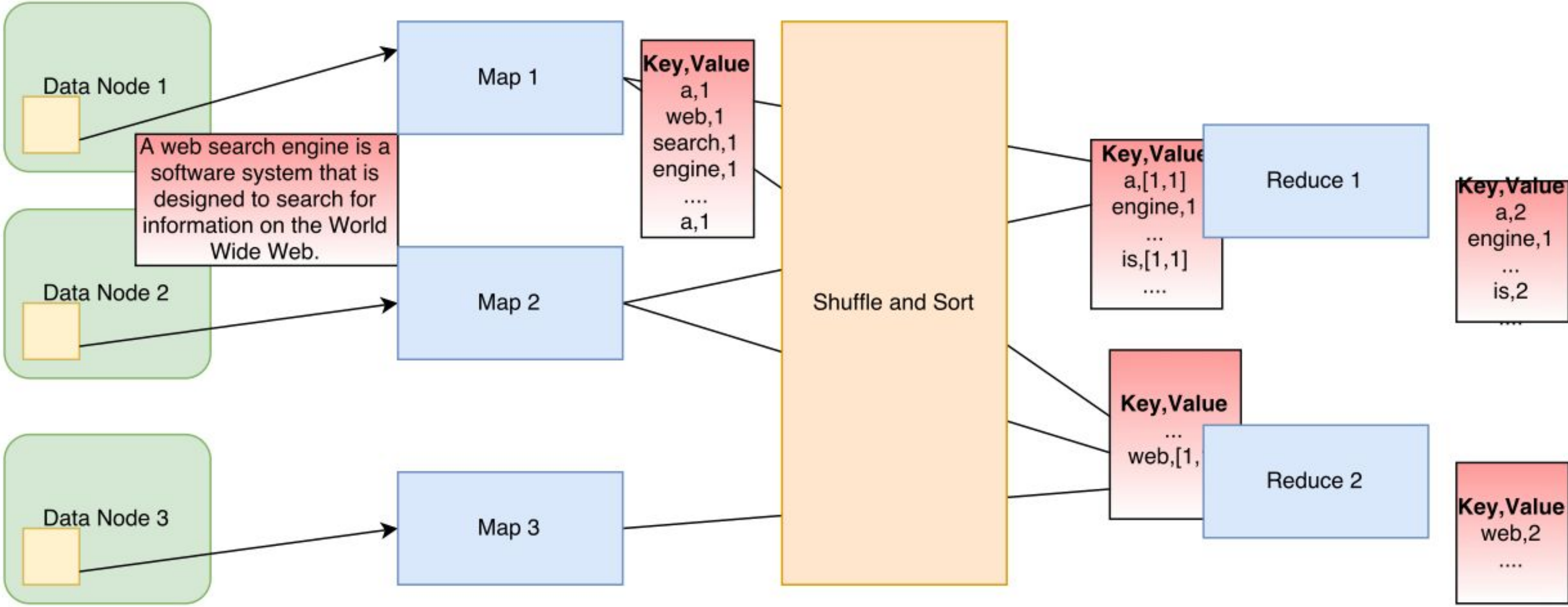
Open Source Communities

- Stackoverflow: https://www.kaggle.com/stackoverflow/stackoverflow
- SNAP: https://snap.stanford.edu/data/
- Wikimedia (nexts sessions)

Some Tools for preprocessing "big data"

- Spark (Scala)
- Pyspark: Python wrapper for Spark
- Hive: SQL-like language
- Kafka: Distributed event streaming platform
- Flink: A framework to work with Data Streams

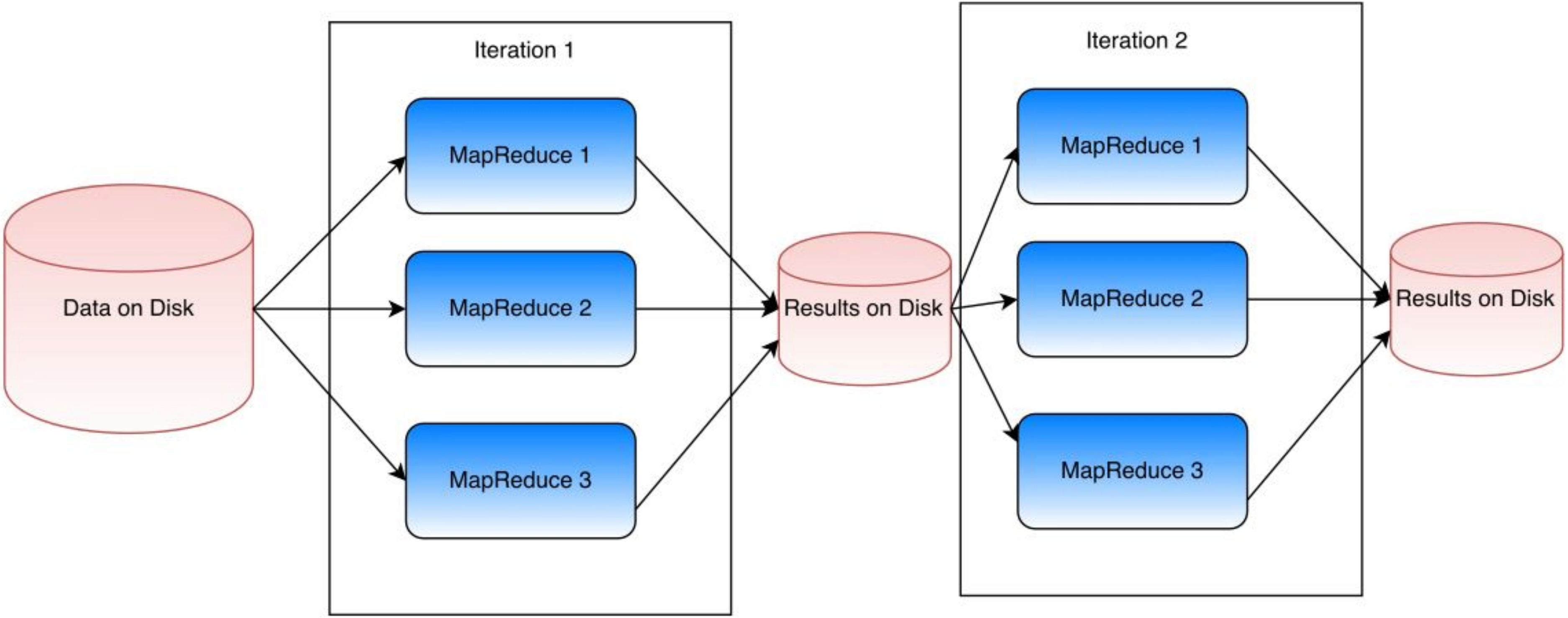
MapReduce – Reduce Stage



Traditional MapReduce is inefficient!

- The data sharing between the MapReduce jobs is to write it to an external stable storage system (Ex – HDFS).
- Most of the time is wasted in Disk I/O
- Hence, Interactive and Iterative jobs become inefficient.
- Spark provides a much simpler framework than MapReduce to do the processing.
- Resilient Distributed Datasets (RDD) are the key idea in Spark

Traditional MapReduce is inefficient



Resilient Distributed Datasets

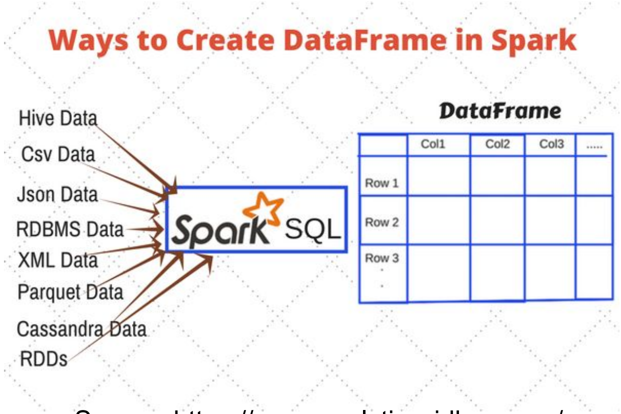
- The features of RDDs (decomposing the name):
  - **Resilient**, i.e. fault-tolerant with the help of RDD lineage graph and so able to recompute missing or damaged partitions due to node failures.
  - **Distributed** with data residing on multiple nodes in a cluster.
  - **Dataset** is a collection of partitioned data with primitive values or values of values, e.g. tuples or other objects (that represent records of the data you work with).
- Other features include: In-Memory, Immutable or Read-Only, Lazy evaluated and Location-Stickiness.

Resilient Distributed Datasets

- It supports in-memory processing computation.
- This means, it stores the state of memory as an object across the jobs and the object is sharable between those jobs.
- Data sharing in memory is 10 to 100 times faster than network and Disk.

And more!!!

- Spark can use dataframe to manipulate data. These dataframes are inspired in (python) Pandas dataframes.



Source: https://www.analyticsvidhya.com/

Pig Example

```
/* load a log file of user sessions. Filter for a specific date and count entries per item*/

f0 = LOAD 'logfile' using PigStorage('\t') AS (log_date:chararray, item_id:chararray, some_stuff:chararray);
f1 = FILTER f0 BY log_date == '20160515';
f2 = FOREACH f1 GENERATE item_id;
f3 = GROUP f2 BY item_id;
f4 = FOREACH f3 GENERATE group AS item_id, COUNT(f2) AS nb_entries;

/* add item name */

item1 = LOAD 'item' using PigStorage('\t') AS (item_id:chararray, item_name:chararray);
join1 = JOIN f4 BY item_id LEFT, item1 BY item_id;
result = FOREACH join1 GENERATE f4::item_id, item_name, nb_entries;
/

STORE result INTO 'result_file' USING PigStorage('\t');
```

Source: [mapr]

PySpark example

```
df.apply(lambda x: …)
```

Spark RDD Example

```
conf = SparkConf()
sc = SparkContext(conf=conf)

f0 = sc.textFile('logfile').map(lambda x: x.split('\t'))
f1 = f0.filter(lambda x: x[0] == '20160515')
f3 = f1.groupBy(lambda (log_date, item_id, some_stuff): item_id)
f4 = f3.map (lambda (item_id, iterable): (item_id, len(iterable)))

# add item name
item1 = sc.textFile('item').map(lambda x: x.split('\t'))

# no need to set the key item_id on both parts before performing the joi, It's already on first place on each part.

join1 = f4.leftOuterJoin(item1)
result = join1.map(lambda (item_id, (nb_entries, item_name)): (item_id, item_name, str(nb_entries)))
# creating a line of tab separated fields, and save it in the result file
result_to_store = result.map (lambda record : '\t'.join(record))
result_to_store.saveAsTextFile('result_file')
```

Spark Dataframe Example

```
conf = SparkConf()
sc = SparkContext(conf=conf)
sqlContext = SQLContext(sc)
f1 = spark.read.csv( "logfile", header=True, mode="DROPMALFORMED") ### Available from spark 2.0.0

f1_df = sqlContext.sql(  "SELECT log.item_id, count(*) AS nb_entries  FROM log   WHERE log_date = '20160515'   GROUP BY item_id" )
f1_df.registerTempTable('log_agg')


#item
df_item1 = spark.read.csv( "item", header=True, mode="DROPMALFORMED")
df_item1.registerTempTable('item')
result = sqlContext.sql(   'SELECT log_agg.item_id, item_name, format_number(nb_entries, 0), FROM log_agg    LEFT OUTER JOIN item ON
log_agg.item_id = item.item_id')

result_to_store = result.rdd.map (lambda record : '\t'.join(record))
result_to_store.saveAsTextFile('result_file')
```

Install PySpark

- **Install with Docker**
  - **$ docker pull jupyter/pyspark-notebook**

  - **https://medium.com/@suci/running-pyspark-on-jupyter-notebook-with-docker-602b18ac4494**
- **Or use a Colab installation:**
  - **https://colab.research.google.com/drive/1cgBHmUrz5bT7OiDZdB-yuYOo00oLwMxT?usp=sharing**

Challenge (45 mins)

- Download AirBnB data from your favorite (available) city from:
  - http://insideairbnb.com/get-the-data.html
- Get the reviews and listing summary, read with pySpark, create **SPARK dataframe** (don't use pandas or similar) and join both tables.
- Compute the average price of properties (rooms, flats, everything together).
- Create a new column that shows the price deviation from the average, such as: price_i / avg_price
- Add another column, *Price_bucket* , that is "H" if the price is higher than the average, and "L" if lower or equal.
- Export this result as CSV (output 1)
- Group by *neighbourhood* and compute the average price of each group
- Export this result as CSV (output 2)

TF-IDF

- We will use MLIB to compute TF-IDF representation of description (name) of each property/house.
- https://spark.apache.org/docs/3.0.0/mllib-feature-extraction.html#tf-idf
- Next, take the result to python (collect) and compute the cosine similarity between all possible pairs of properties.
- Represent the results as a graph (you can use networkx python lib) where the weight of the edge is the cosine similarity between names.

Questions?

diego.saez@upf.edu