Lecture Notes for
Machine Learning in Python

Professor Eric Larson

Week 11 Dealing with Larger Data

Class logistics

- A2 is over!
    - thanks for all your work on this
- Remainder of semester:
    - A3 (no resubmits)
    - final in-class-assignment
- Grading is coming
    - resubmissions will be manipulated if I take too much time…

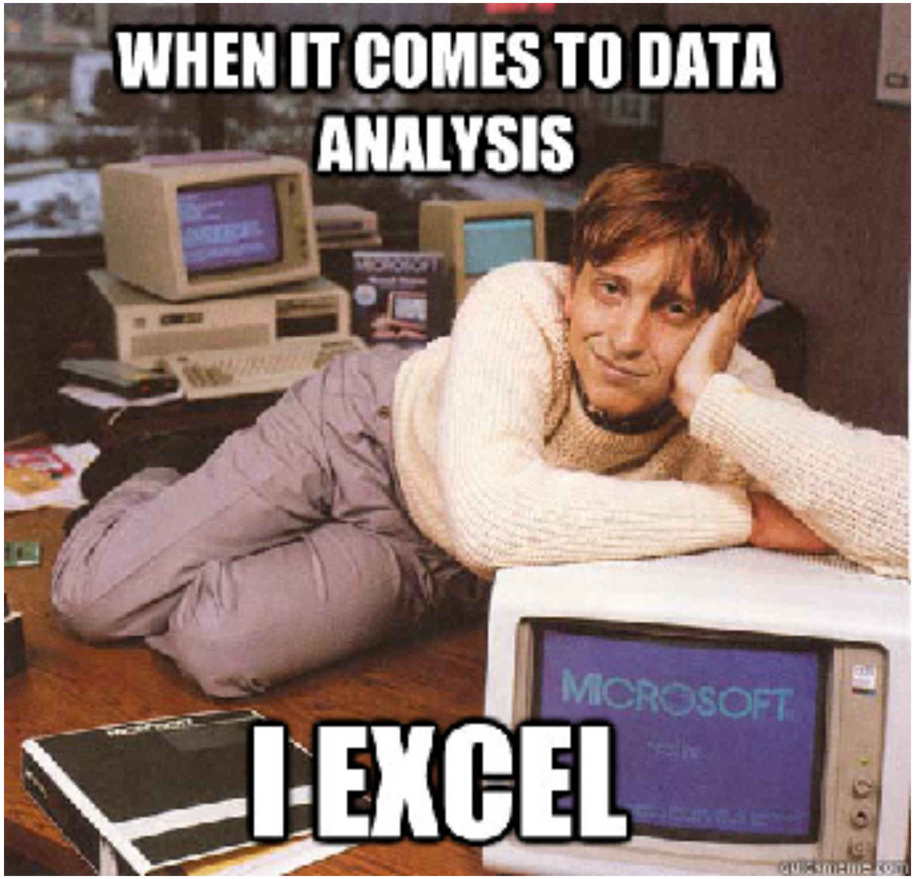
A2 Retrospective

- What problems did you run into?
- How did you solve them?

Two Lecture Agenda

- Today:
    - Defining larger data
    - Data Parallel: Grid Search
    - Out of core processing
    - Smarter processing

Big Data, Massive Data, Medium Data, and Maybe Massive-ish Data



Big Data

- Refers to the exponential size of data creation

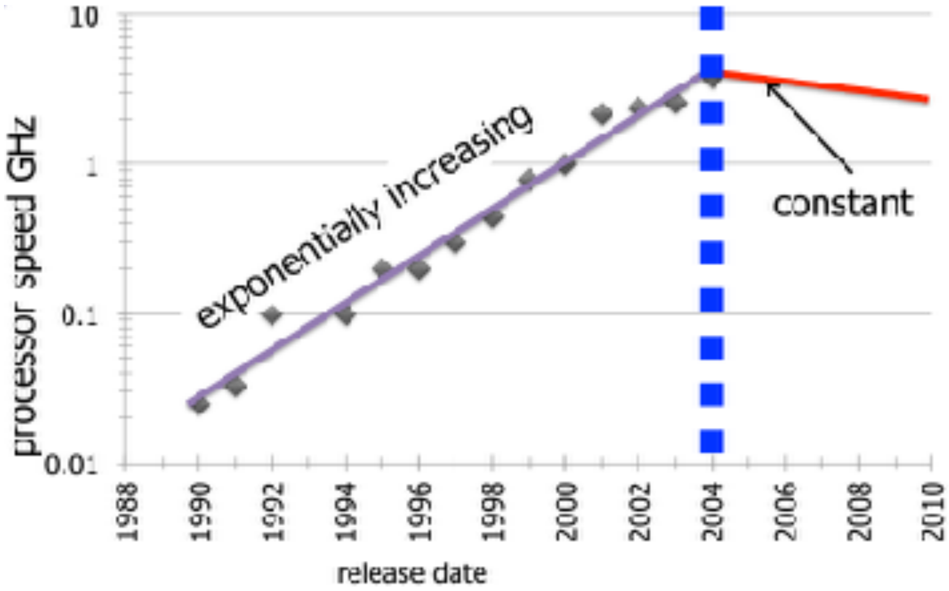
6 Billion
Flickr Photos

28 Million
Wikipedia Pages

1 Billion
Facebook Users

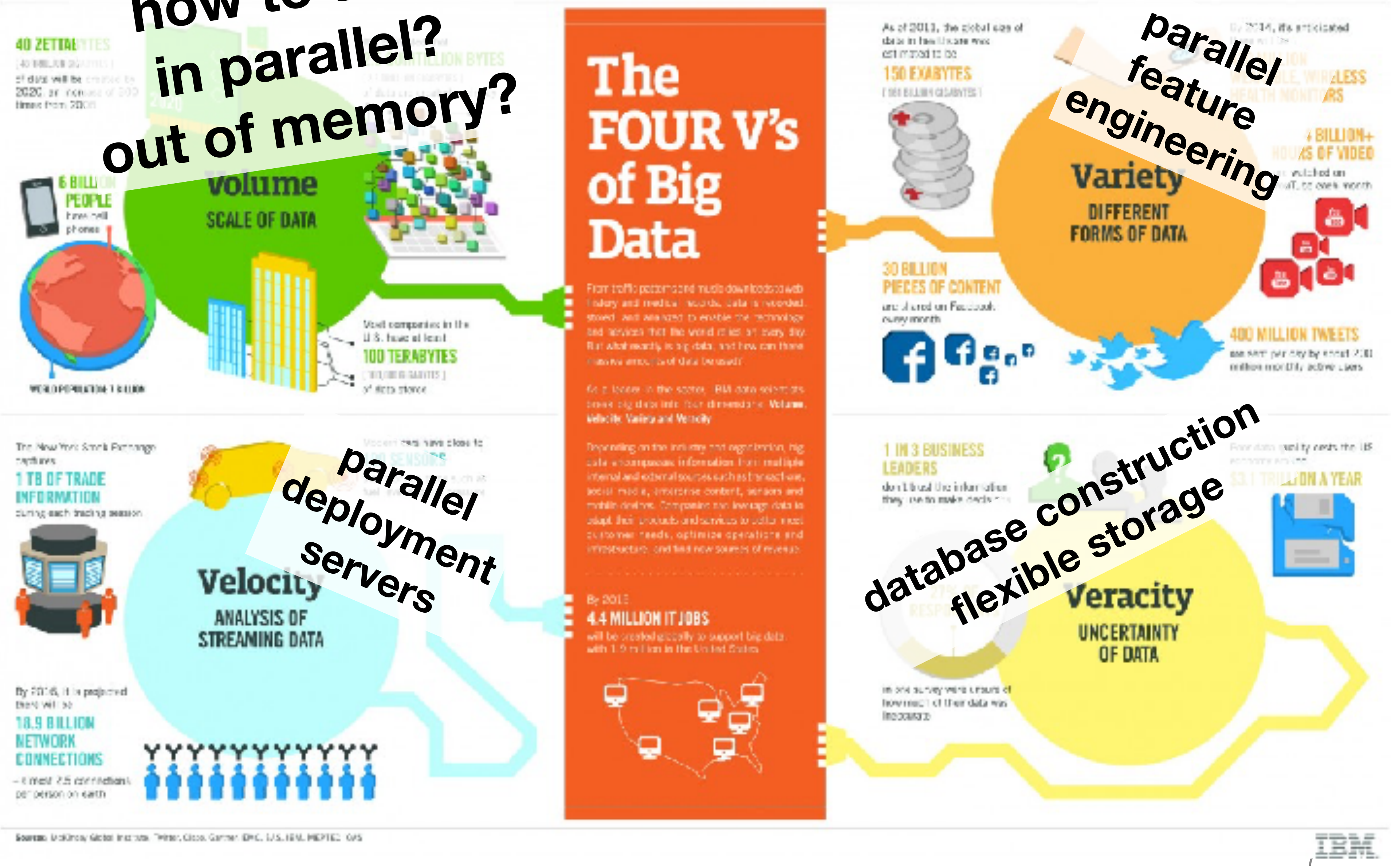
72 Hours a Minute
YouTube

- But is motivated by stagnant processor speed



You will see many definitions use the 4Vs

- **volume**: huge amount of data
- **variety**: types of data
- **velocity**: streaming data
- **veracity**: uncertainty of data



how to train in parallel? out of memory?

parallel feature engineering

parallel deployment servers

database construction flexible storage

http://www.ibmbigdatahub.com/infographic/four-vs-big-data

Dealing with Larger Data

- Massive is in the **eye of the beholder**
  - some breakdown of data is still too massive to use traditional, in-memory, processing methods
- **out-of-core**: cannot fit into RAM on single machine or requires too much paging to keep in memory efficiently
- **parallelism**: can be broken into smaller subproblems.
  - Usually this is “embarrassingly parallel” that is exploited
  - **distributed**: clusters where code is brought to the data for processing (map-reduce).
- **sub-sampling**: don’t ever rule this out!

“Maybe Massive” Examples: out of core

- **Out-of-core** on a single machine is great for exploratory analysis (10’s of GB up to 1TB)
    - **use case**: want to use similar statistical, visualization, and aggregation tools as you are accustomed
    - **objective**: simplify the data in something manageable
    - **example**: visualizing airline data or other tabular data in a massive relational database
    - typically emulates distributed architecture, used for debugging, **this will be our focus**!

“Massive” Examples: cannot store

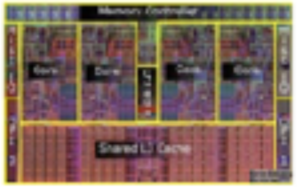
- Distributed
- **forced parallelism**: cannot fit on one hard drive
  - one server cannot store the data
  - streamed from many different sources
- implementing queries or aggregating over **truly massive** amounts of data—example: Google search, facebook, twitter
  - few companies work with data of this size (but not rare)
  - you need to define your problem as a map-reduce step and be conscious of data location or aggregate the data into a graph, then perform analysis

Data on different architectures

- scalable learning is hard because of
  - iterative algorithms
  - programability
  - failures
  - distributing data to parallel processes

abstracting parallel programming is just not there yet

GPUs



Multicore

Clusters

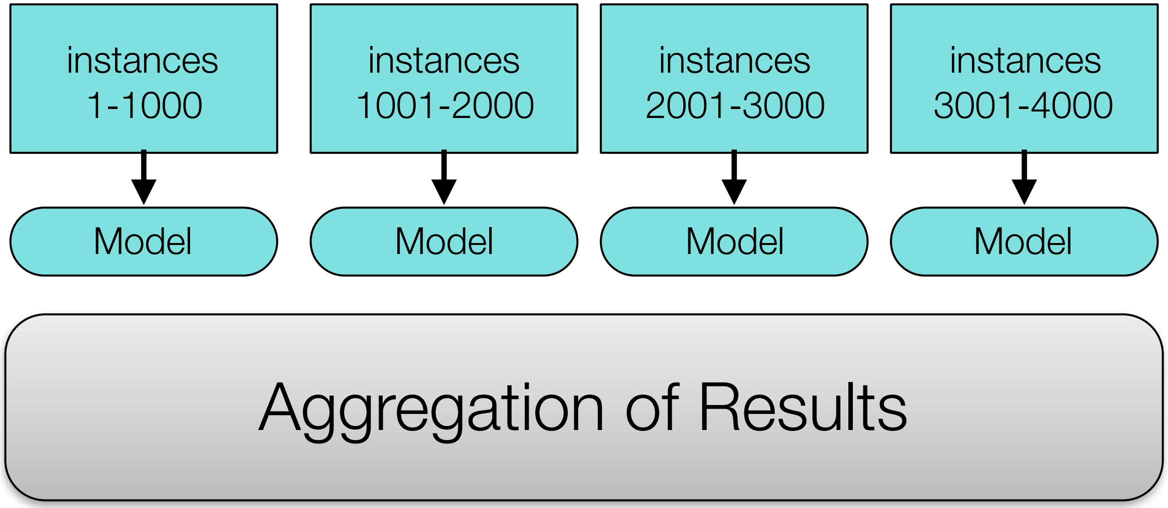
amazon
webservices™

Clouds

Supercomputers

Data Parallel (Embarrassingly Parallel)

- simplest type of parallelism, easy to exploit
  - problems can be broken into smaller, independent sub-problems
  - like running prediction on a data set of independent instances



Data Parallel Map Reduce

Map

- Data parallel over the elements (like images)
- Emit key/values (like image features)

input -> list(key, value)

Reduce

- Aggregate over the keys
- Must be commutative and associative operations
- Data parallel over keys
- Emit reduced output

list(key, value) -> output

Data Parallel Map Reduce

Map Reduce

Self Test 11.1:

We are performing Logistic regression on a large dataset located upon a cluster of machines (each cluster has a portion of the data). Can map-reduce help our computation?

(A) Yes, we can run gradient estimation separately on each cluster

(B) No, gradient estimation is iterative and the overhead of transferring gradient data negates the speed up in distributed computation

(C) Yes, we can train a full Logistic regression model on each machine and combine the outputs

(D) No, its better to sub-sample from each cluster and train a model on a single machine

Map Reduce in Machine Learning

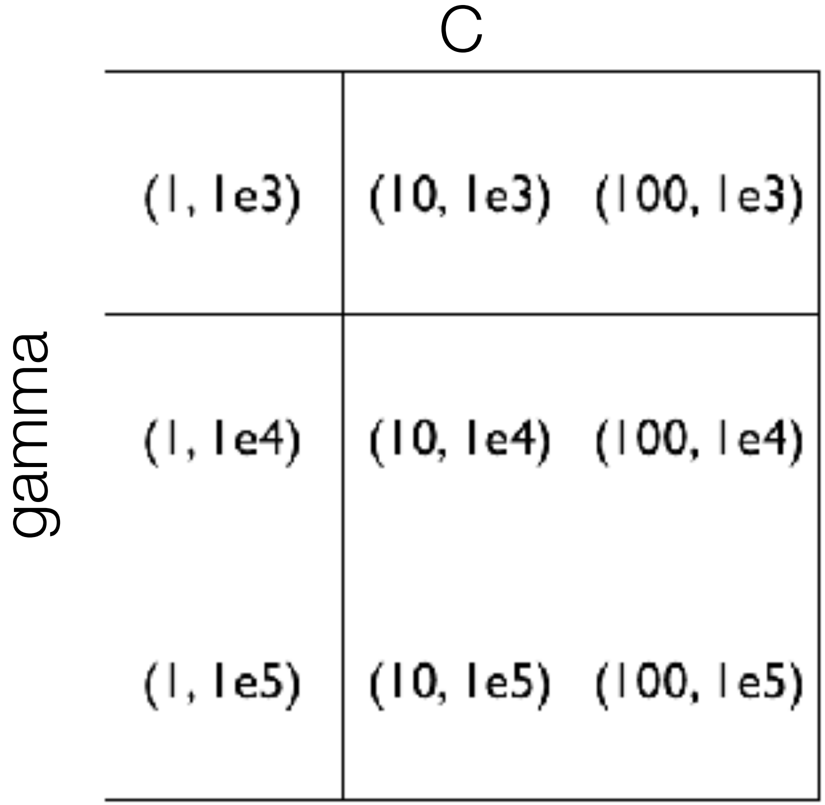
- Much of learning is iterative
- Though not always
  - ensembles
  - multi-class training (for large classes)
- Data gets moved around your cluster, possibly replicated, overhead might eliminate benefits
  - keep data near computation (or on HDFS)
- Mostly, Map Reduce is great for embarrassingly parallel
  - feature extraction
  - grid searching
  - cross validation
  - computing statistics

lets look at these!!

a good tutorial on map reduce in data mining: http://videolectures.net/kdd2010_papadimitriou_sun_yan_lsdm/

Grid Searching

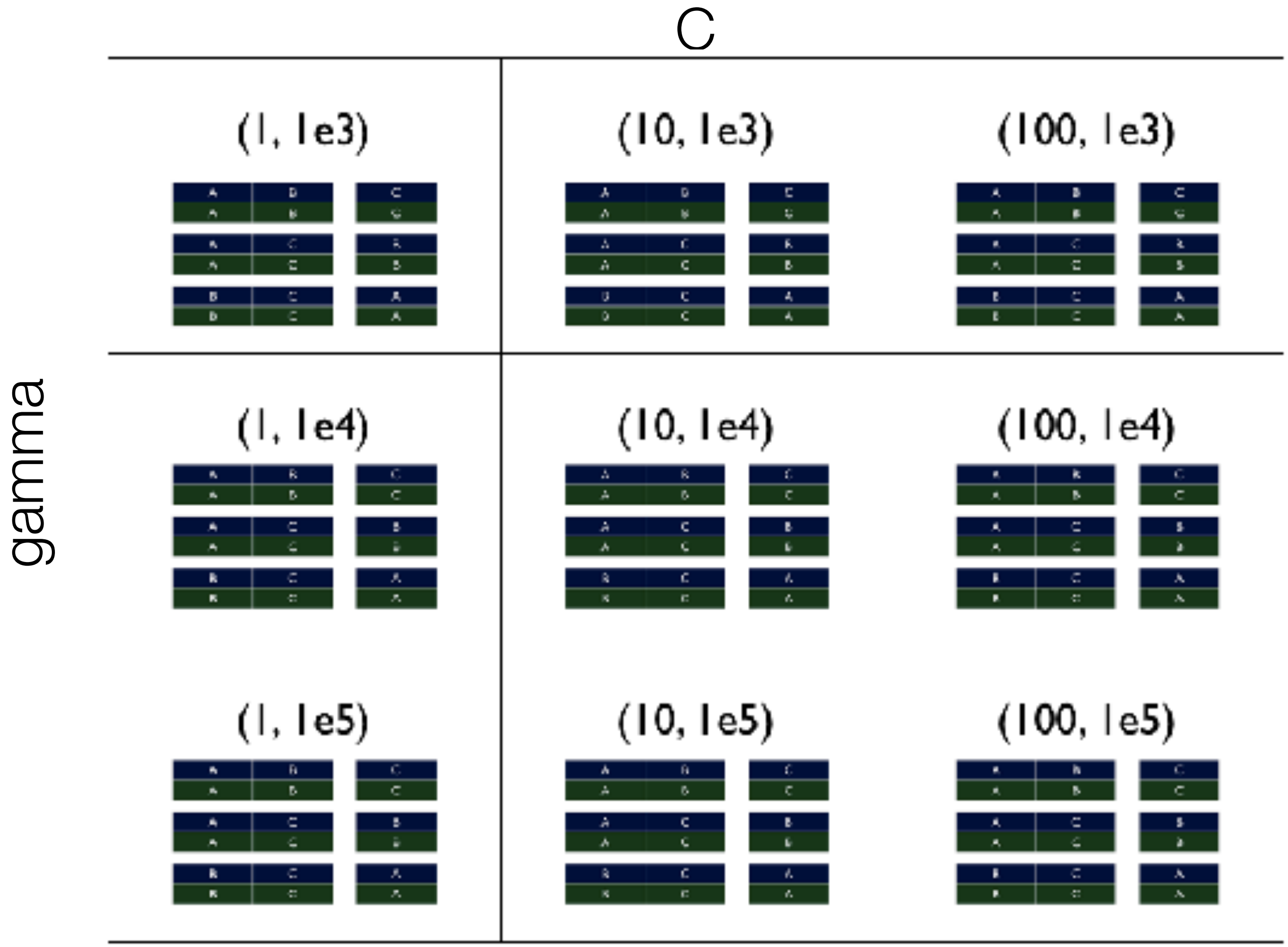
- Trying to find the best parameters
  - `SVM: C=[1, 10, 100] gamma=[1e3,1e4,1e5]`



	C		
gamma	(1, 1e3)	(10, 1e3)	(100, 1e3)
	(1, 1e4)	(10, 1e4)	(100, 1e4)
	(1, 1e5)	(10, 1e5)	(100, 1e5)

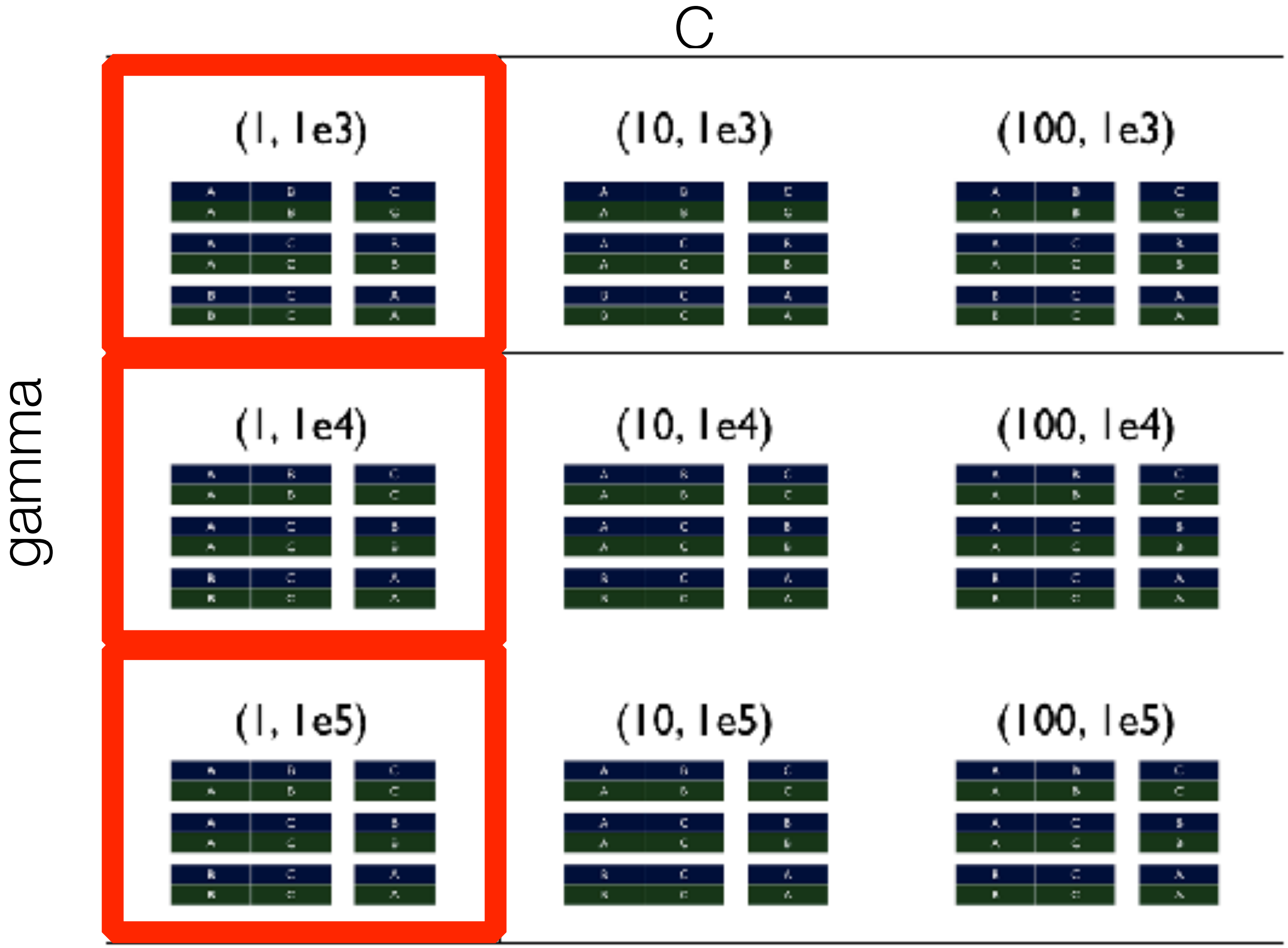
Grid Searching

- For each value, want to run cross validation…



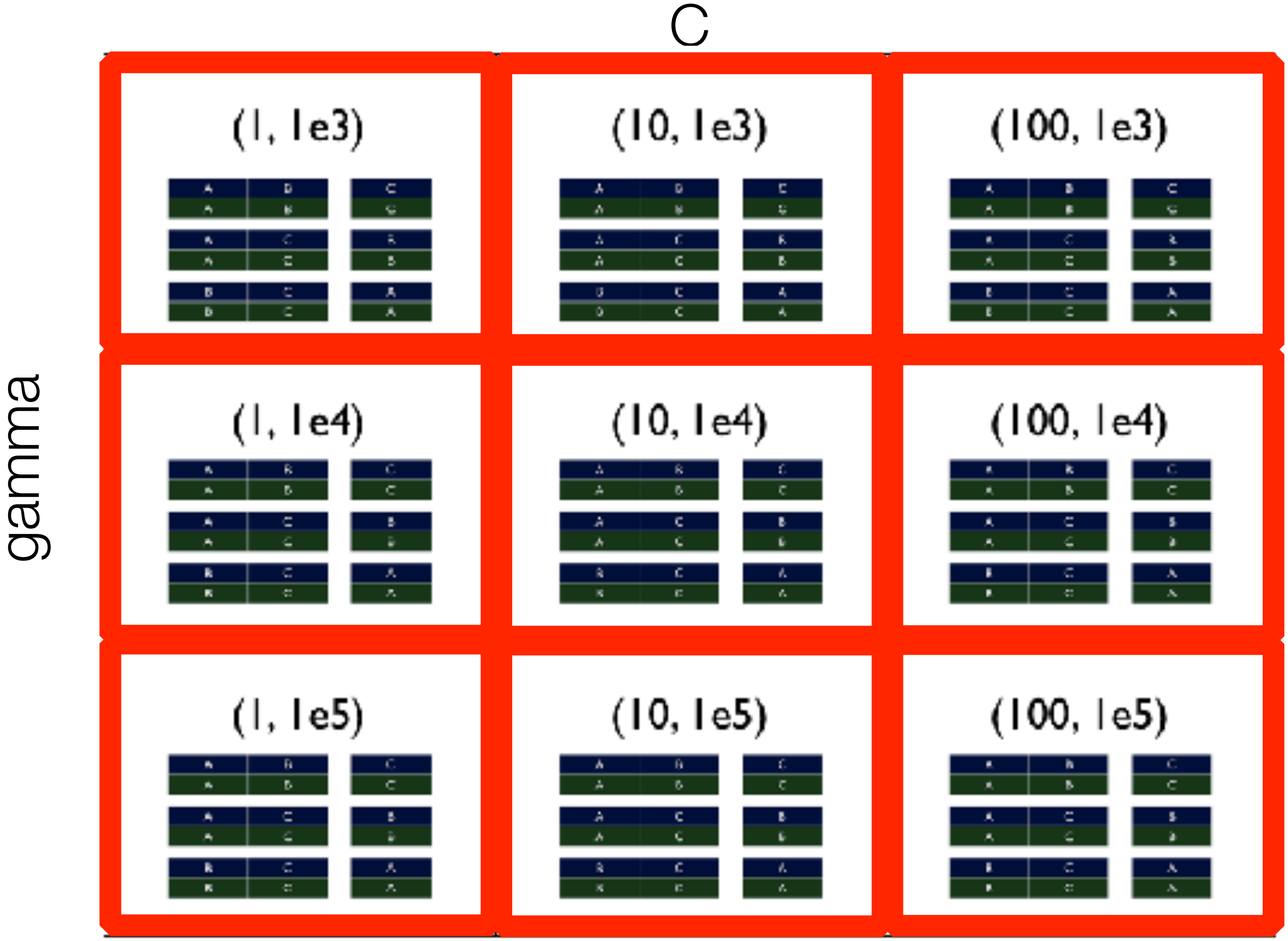
Grid Searching

- Could perform iteratively



Grid Searching

- or at random…



Demo

Parallel Parameter Tuning in sklearn

notebook 10

Other tutorials:

Olivier Grisel’s Tutorial:

https://www.youtube.com/watch?v=iFkRt3BCctg

~3 hours

https://github.com/ogrisel/parallel_ml_tutorial/blob/master/rendered_notebooks/06%20-%20Distributed%20Model%20Selection%20and%20Assessment.ipynb

Lecture Notes for
Machine Learning in Python

Professor Eric Larson

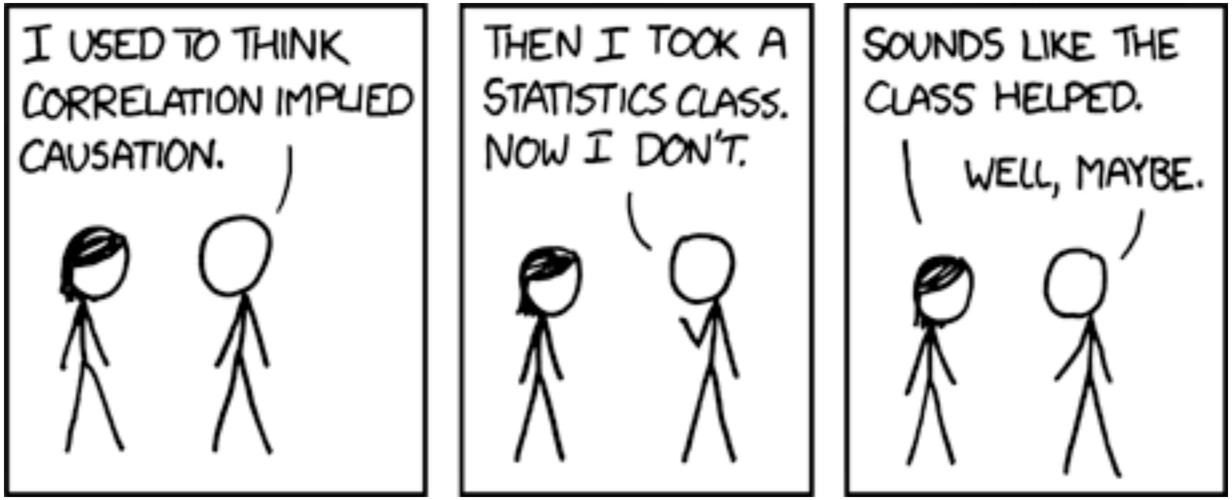
Week 11 Dealing with Larger Data

Two Lecture Agenda

- *Last Lecture*
  - *Defining larger data*
  - *Data Parallel: Grid Search*

- Today:
  - Out of core processing
  - Smarter processing

Managing Memory


I USED TO THINK CORRELATION IMPLIED CAUSATION.
THEN I TOOK A STATISTICS CLASS. NOW I DON'T.
SOUNDS LIKE THE CLASS HELPED.
WELL, MAYBE.

Out of Core Memory

- **Data cannot fit into RAM**
    - Classic **space-time** tradeoff
    - Load from disk in **chunks**
- Each chunk: **memory mapping**
    - map on disk instead of RAM
    - works, but is **slow**
    - need to optimize for **sequential access**

Out of Core Memory

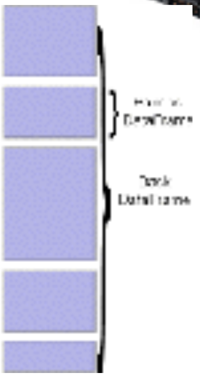
- **Data Format** is paramount: disk I/O and processing power
  - **Formatted file**: like a CSV, slow to read and requires parsing. Avoid this at all costs! But, this is a likely format to start with…
  - **Database**: optimized access, we want to use or emulate this type of memory access
  - **Uncompressed Raw Data**: once read, no further processing is needed, stored exactly as it is in memory
  - **Compressed**: Less to read from the disk, but requires further decoding

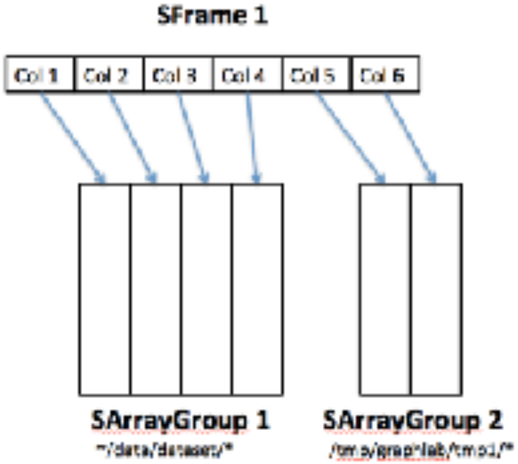
To Compress or not to Compress

- **Uncompressed**
  - fast indexing of **single datatype** arrays (like accessing struct in c++)
  - faster to know **where to read from file**, only need to read memory that is needed (fseek, fread)
  - quick sequential access, but **expensive** to add/delete data
- **Compressed** (or slightly compressed)
  - if compression is **hardware optimized**, then very little overhead for the decoding operation
  - need to read chunks of **possibly irrelevant data** to decompress what you are interested in
  - much **less** hard disk **memory footprint** required, could be as much as 80-90% savings

Out of Core: Representation

- **single file**: all data stored back to back in one file exactly like dumping a c-struct to file (raw bytes)
    - **tough to add columns** — need to read and write file completely to keep continuity
    - typically only **support one data type** or only numeric data
    - **examples**: numpy memmap and bigmemory in R
- **multiple files**:
    - separated by row chunks
        - each file has identical format, multiple data types
        - **example**: Dask (many pandas chunks)
    - separated by columns
        - trivial to **add columns** of new data
        - string and categorical data are easier to support
        - for lots of columns, can overwhelm the file system
        - **example**: GraphLab SFrames





Out of Core: Access

- Usually **subsumed to the programmer** via APIs
  - Numpy's memory map which is simply using c-file pointers rather than in-memory stack pointers
    - set up memory and never load all of it
  - easy to share memory across processes as long as file locking implemented appropriately (much easier to lock on the file system)
  - file system caches sequential access (huge speed up!!)
  - "*easy*" to use in `scikit-learn` with `partial_fit`

Out of Core: Access

- **Self Test 11.2**: A memory mapped file with 1 billion float32 values will be of what size on disk:
  - (A) 32 GB
  - (B) 4 GB
  - (C) 1 GB
  - (D) it depends

Demo

Out-of-core Machine Learning

Partial fitting

notebook 11

Other tutorials:

- https://github.com/rasbt/pattern_classification/blob/master/machine_learning/scikit-learn/outofcore_modelpersistence.ipynb

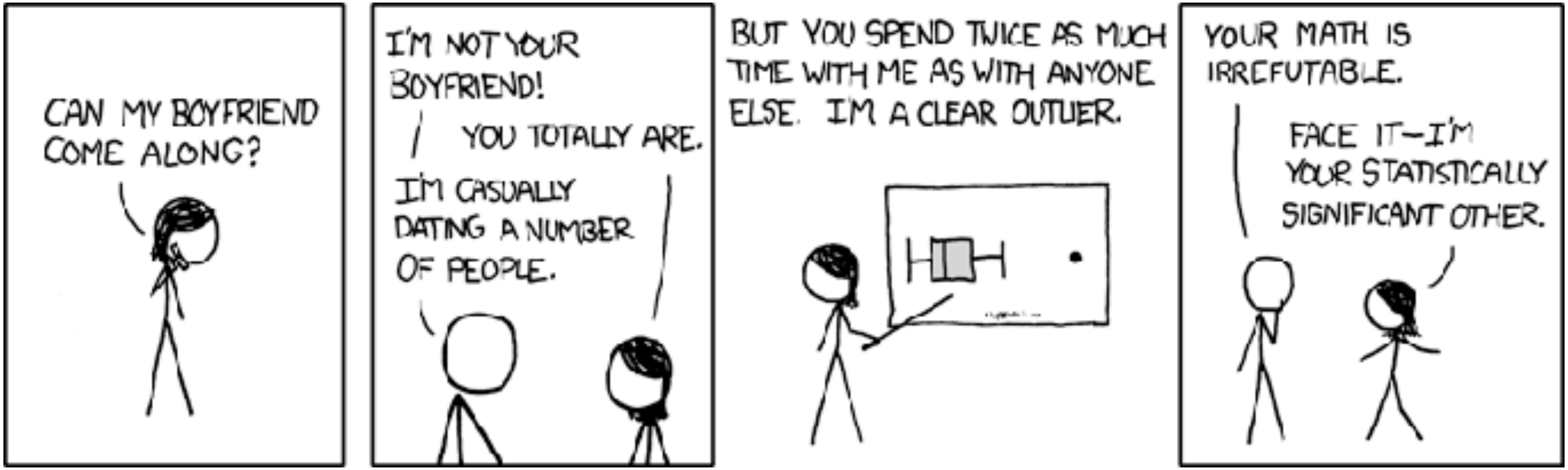
Anything from Jim Crist or Mathew Rocklin:

- https://github.com/jcrist

Alex Perrier:

- https://www.opendatascience.com/blog/riding-on-large-data-with-scikit-learn/

Smarter Processing


CAN MY BOYFRIEND COME ALONG?
I'M NOT YOUR BOYFRIEND!
YOU TOTALLY ARE.
I'M CASUALLY DATING A NUMBER OF PEOPLE.
BUT YOU SPEND TWICE AS MUCH TIME WITH ME AS WITH ANYONE ELSE. I'M A CLEAR OUTLIER.
YOUR MATH IS IRREFUTABLE.
FACE IT—I'M YOUR STATISTICALLY SIGNIFICANT OTHER.

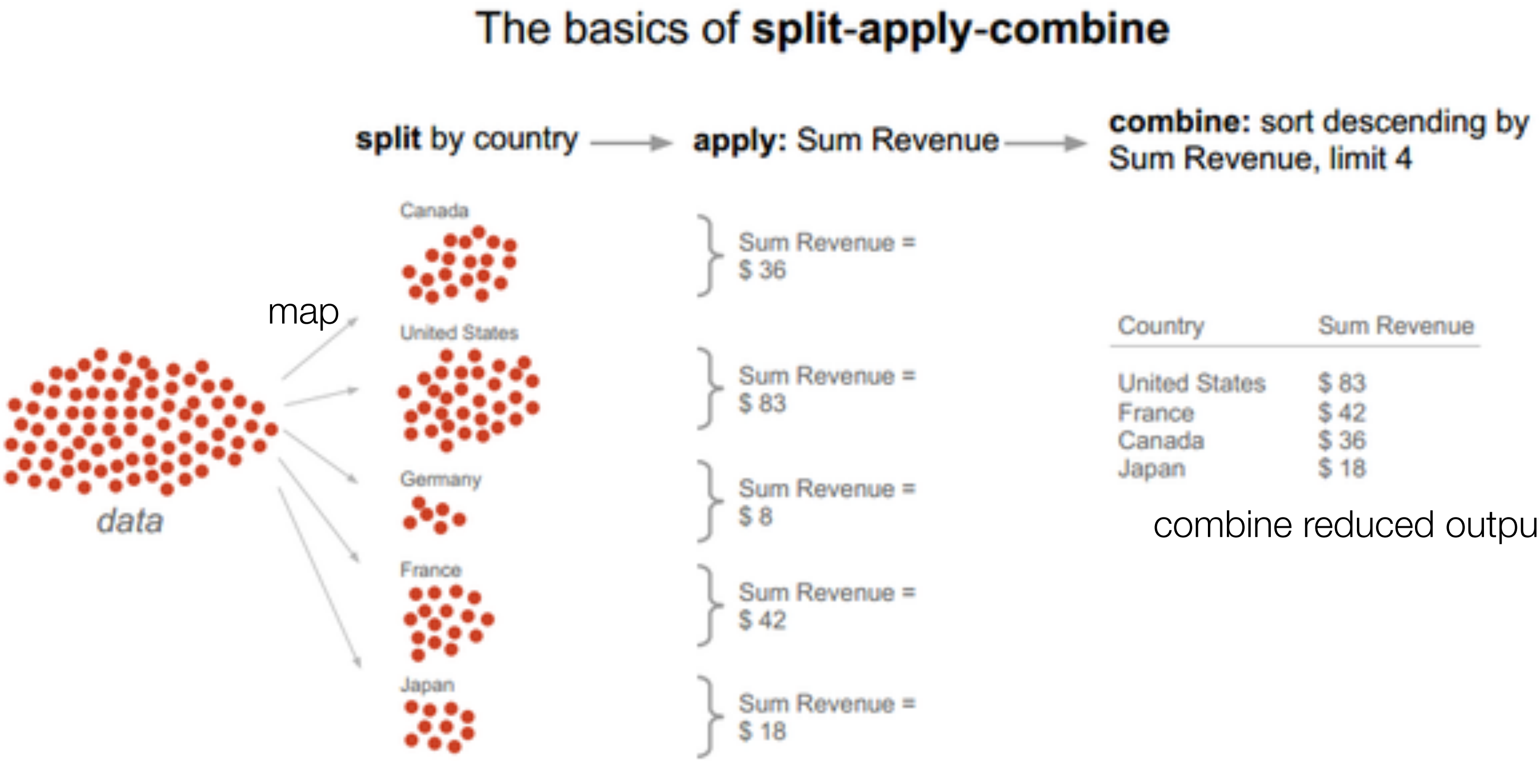
Smarter Processing

- **Data Parallelism**
  - Common form: split-apply-combine for **embarrassingly parallel tasks**
  - Map-Reduce with one reducer
  - **Mini-batch** with averaging
- **Task Scheduling**
  - Tasks **agglomerated** until required to be executed
  - Multiple tasks run in **one** data **access**
  - lazy evaluation
- These are **not mutually exclusive**!

Spilt-Apply-Combine

- Just an embarrassingly parallel form of map-reduce
  - **Split**: partitioned grouping of the data (map)
  - **Apply**: a custom aggregation of the group (reduction, done in parallel)
  - **Combine**: combining the groups together (emission of reduced output)
- **Example**: taking grouped statistics (min, max, mean)

Spilt-Apply-Combine



Country	Sum Revenue
United States	$ 83
France	$ 42
Canada	$ 36
Japan	$ 18

map

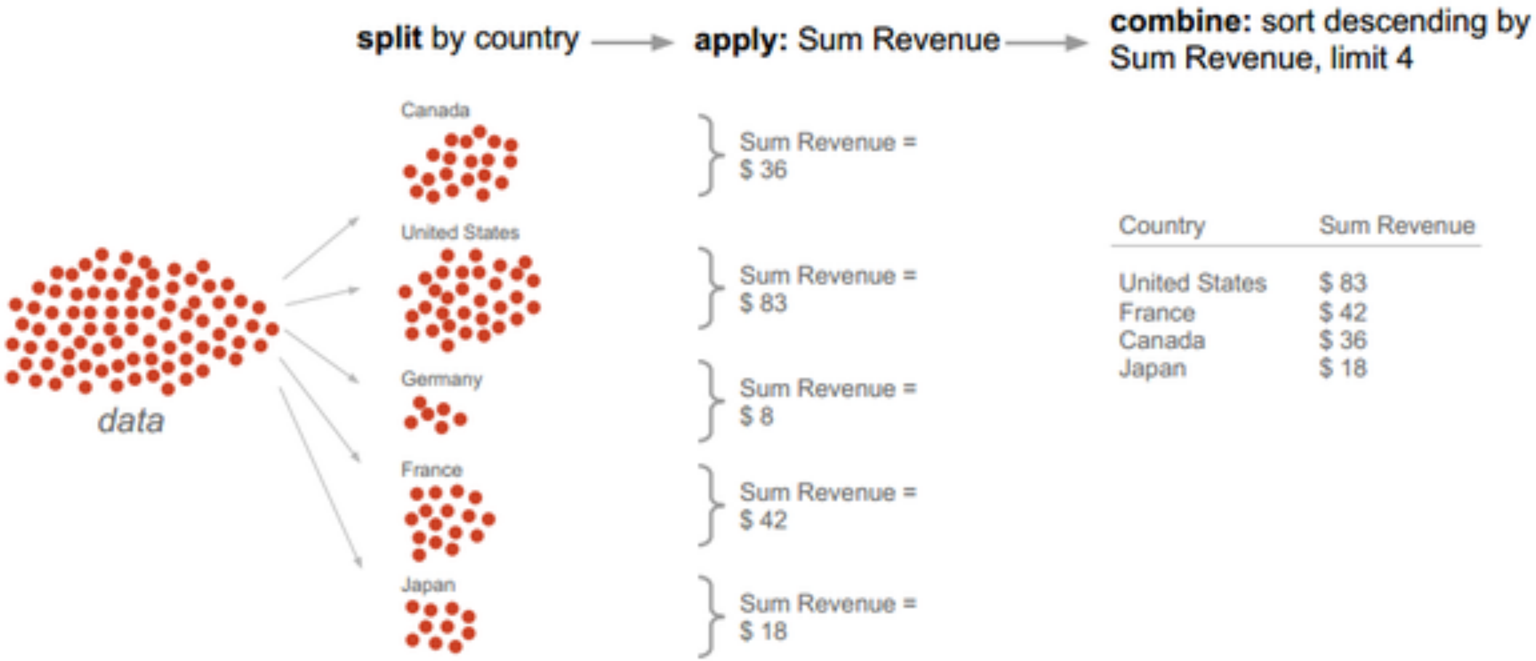
parallel processes

emit reduced key/value

combine reduced outputs

https://twitter.com/medriscoll/status/557989114805747712

The basics of **split-apply-combine**



Country	Sum Revenue
United States	$ 83
France	$ 42
Canada	$ 36
Japan	$ 18

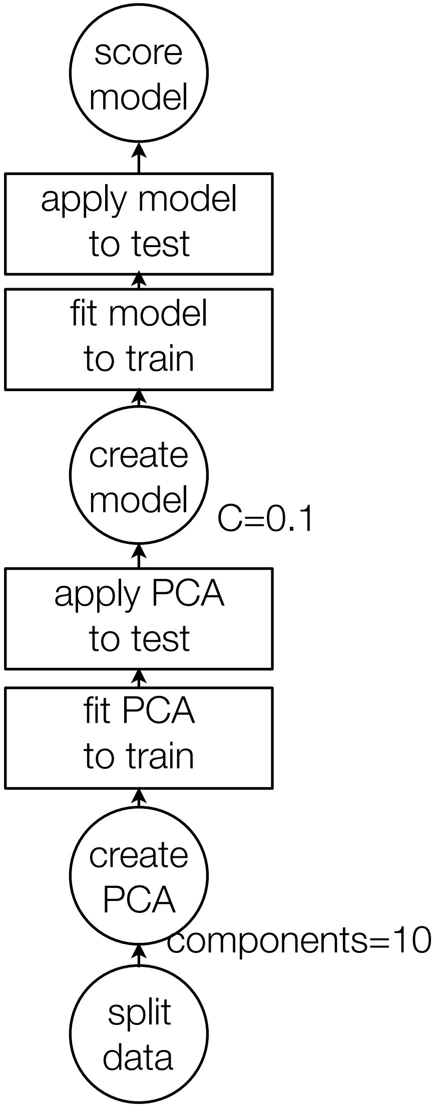
- **Self Test 11.3**: Would split-apply-combine in a distributed environment work well?
    - (A) Yes, data is properly divided for parallelism
    - (B) No, split is not consistent for where data resides in each server
    - (C) Yes, but you need to distribute data by group before carrying out operations
    - (D) No, splits might be unequal sizes, negating advantages

https://twitter.com/medriscoll/status/557989114805747712

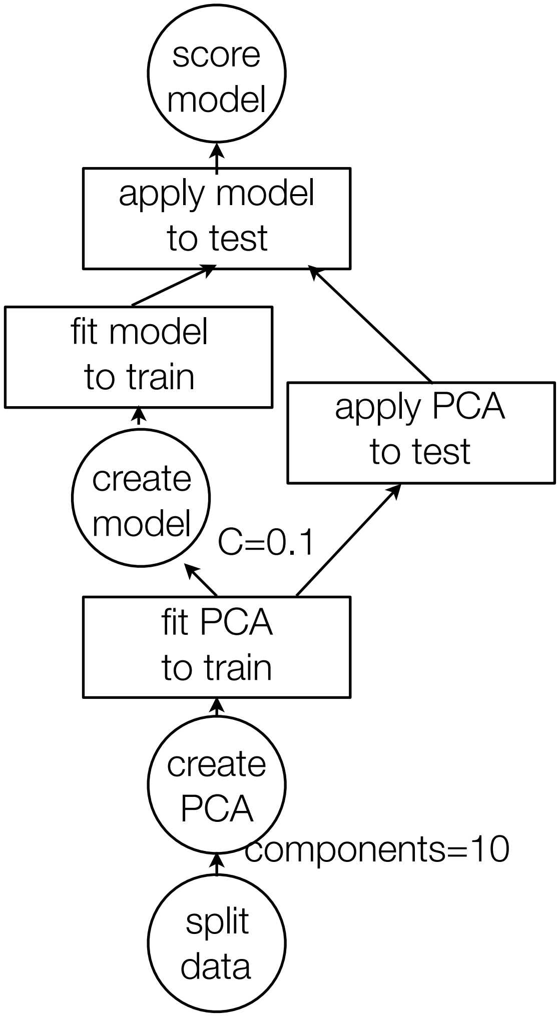
Lazy Evaluation

- **Wait** to evaluate until all tasks are agglomerated
- Typically for single machine architectures where explicit programming of graph is **subsumed**
- Use **Computation Graph**
  - only perform **needed** operations
  - **parallelize** operations where possible
  - **inline** functions for speedy operations

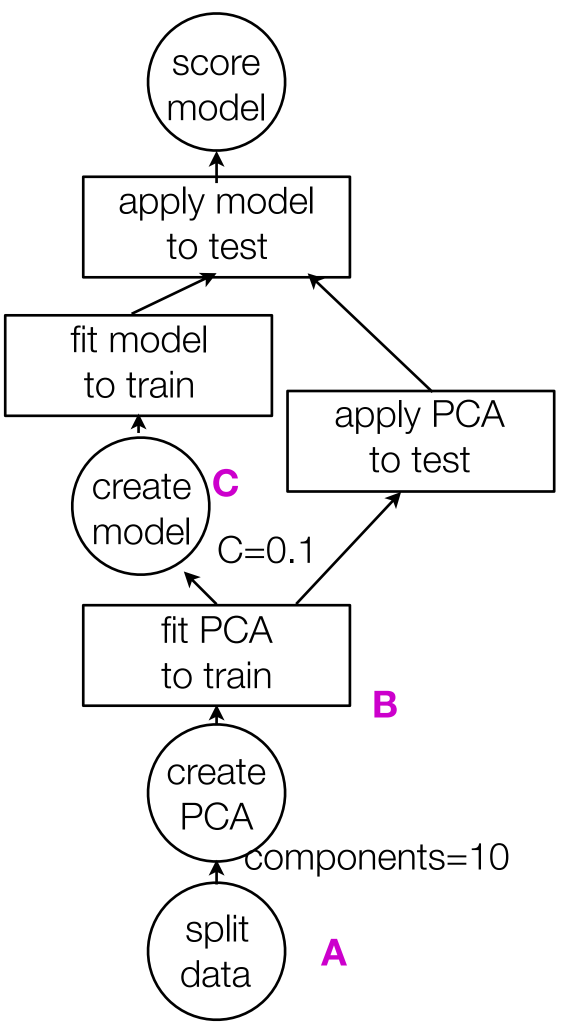
Lazy Evaluation: Example

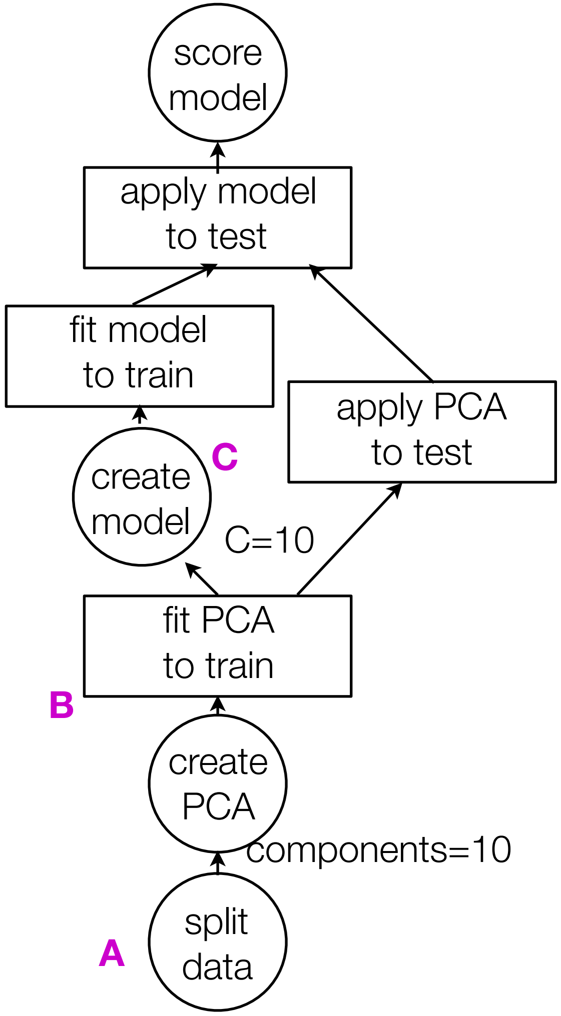


Lazy Evaluation: Example



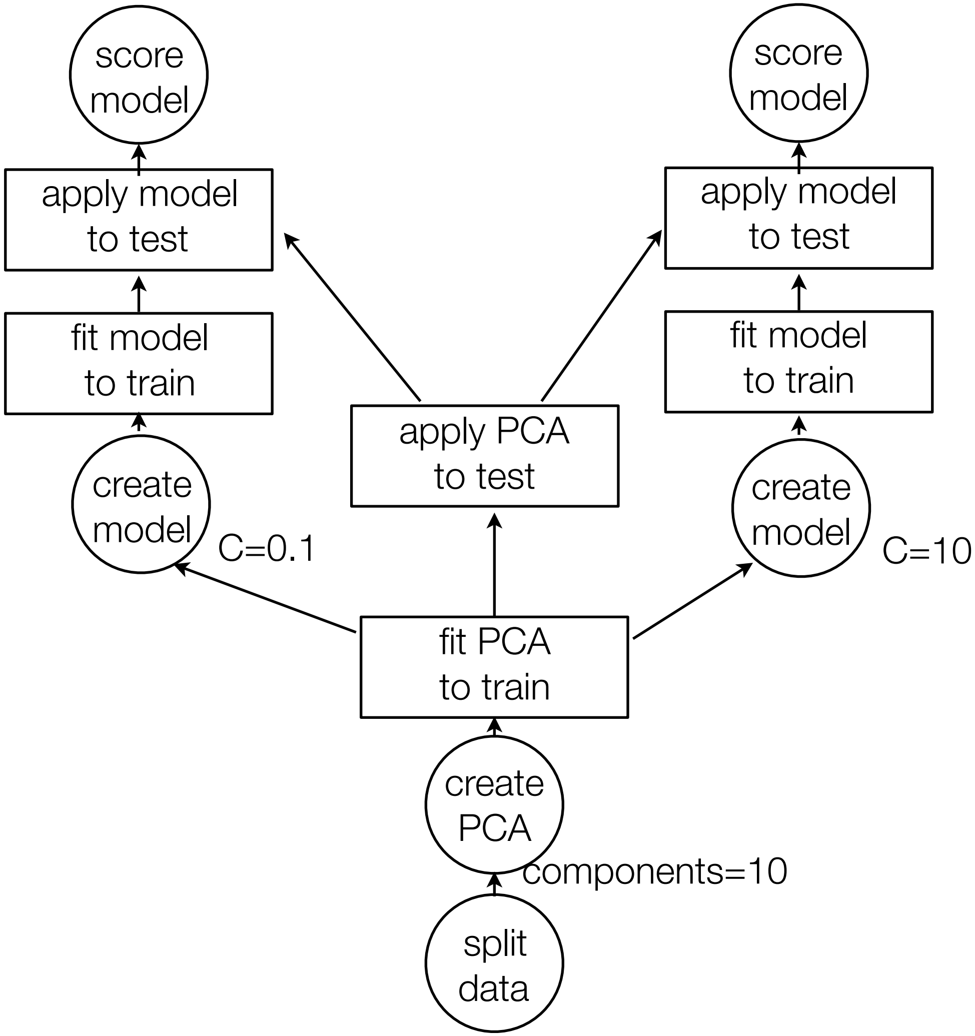
Lazy Evaluation: Example



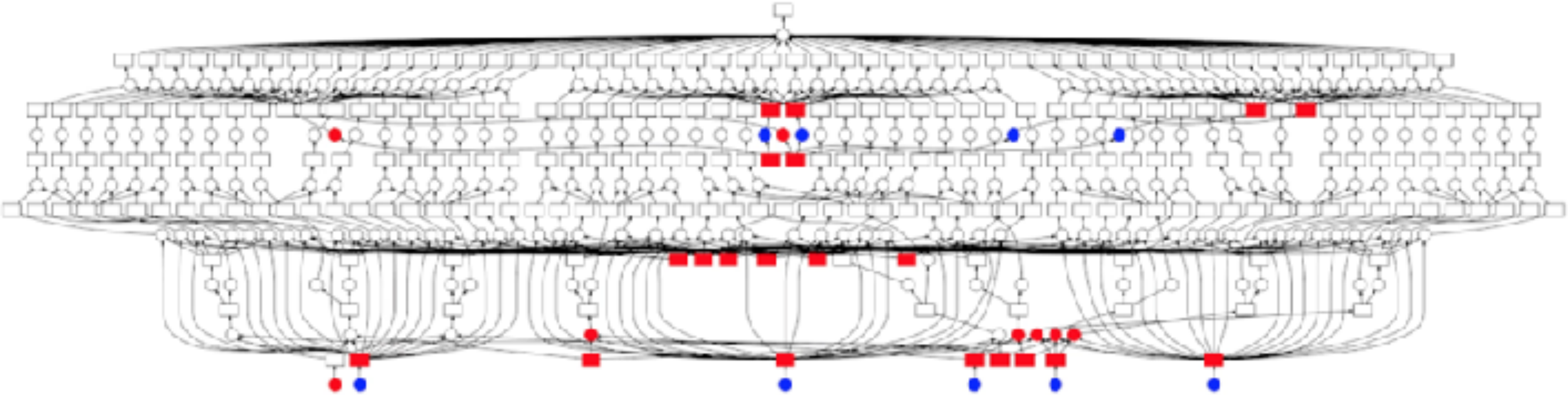


Self test 11.4: Which parts
of the two models can be combined?

Lazy Evaluation: Example



Lazy Evaluation: Example



task graph for **GridSearch**

http://matthewrocklin.com/blog/work/2016/07/12/dask-learn-part-1

Lazy Evaluation: Caveats

- **Optimization** of task graph **is overhead**
  - Small for long running tasks
  - Might **increase memory footprint** for cached values
  - Doesn't solved any issues with copying data…
- But can easily (and should) be **combined** with **out-of-core** processing

Lazy Evaluation: Caveats

- Self Test 11.4: Computation Graphs are used by which of the following Companies/APIs for interacting with data?
  - (A) Dask
  - (B) Turi (graphlab)
  - (C) TensorFlow
  - (D) Theano

Demo

Putting it all together

Out of Core and Lazy Evaluation

in Dask (*notebook 12a*)

in GraphLab (*notebook 12b*)

with Airlines

Other tutorials:

- https://github.com/rasbt/pattern_classification/blob/master/machine_learning/scikit-learn/outofcore_modelpersistence.ipynb

Anything from Jim Crist or Mathew Rocklin:

- https://github.com/jcrist

Alex Perrier:

- https://www.opendatascience.com/blog/riding-on-large-data-with-scikit-learn/

For next time

- Back to Neural Networks
- Deep Learning Architectures
  - will finish out semester
  - convolutional NN
  - recurrent NN
  - TensorFlow examples
    - but you can use whatever API you wish
    - including SKFlow (now a part of TensorFlow)