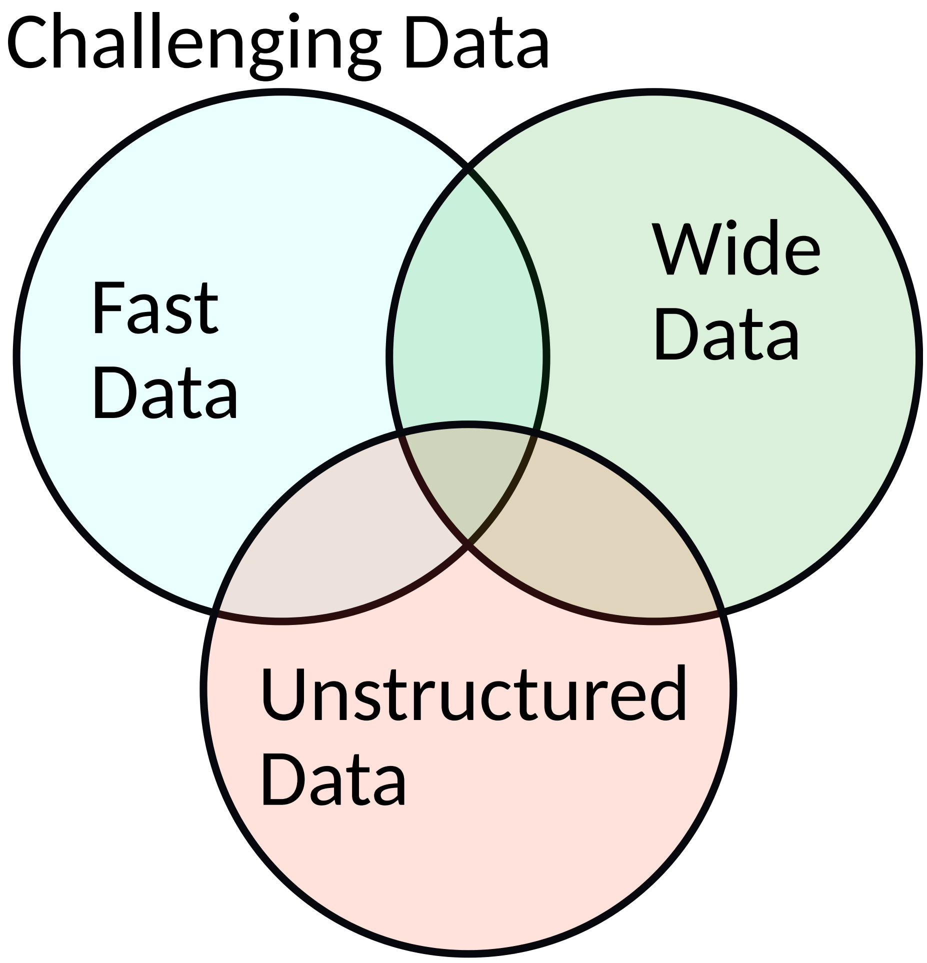
Challenging Data

The 27 V’s of challenging data

- volume = big
- velocity = fast
- vide (OK I meant wide, but surely w counts as two v’s)
- very long = e.g., data collected over decades: e.g., SAX 45 and up
- variety = sources, formats, content, …
- variability = inconsistent data rate, e.g., event triggered
- veracity = big doesn’t mean correct, but it does make it hard to clean the data
- vacuity = there’s a lot of data, but no signal
- priVacy = what it says
- vagueness = data but no question?
- volatility = noisy, all over the place
- vaultification = data is kept locked up in separate boxes
- victimised = the data has already been tortured
- verbosity = wordy and unstructured
- viscousity = hard to wade through
- vagrancy = hard to pin down
- vapidity = are we bored yet?



Wide Data

We’ve already talked about this – data that lives in a high-dimensional space. Invoke the curse of dimensionality and move on.

Fast Data

Fast data that arrives at you fast.

The metaphor I like is “drinking from a fire hose.”

Typically we need to process the data in (near) real-time.

Examples:

- Packets on a data network. A 10 Gbps link is common. Internet packets can be as small as 40 bytes. So you could see more than 30 million packets per second. That means you have a few **nanoseconds** to process each packet. In a big network, you might have thousands of these links.
- Every minute (according to Wired) 48 hours of video are uploaded to YouTube.
- Visa processes 24,000 transactions per second. Each needs to be dealt with in milliseconds.

Truly fast data is arriving so fast that you cannot store it. Even if your disks are big enough, the disk on your computers it too slow to move the data into storage. Really, really fast data is too fast for conventional computer memory (RAM) so you need specialised hardware, *e.g.,* SRAM instead of DRAM, or even CAMs and TCAMs. For instance cache memory in CPUs (on modern computers) uses SRAM. But SRAM is more expensive, and much, much smaller than DRAM.

We are used (by now) to data that appears as a tensor. Another model for data, often used for fast data, is that of streaming data where you see a stream of (key,value) pairs. That is, the data may be a stream $(k_i, \mathbf{v}_i)$ the keys index into some space and the values are of interest. A simple example might be bank transfers where the key corresponds to the account numbers of the source and destination accounts, and the value vector contains the amount being transferred.

You could model this as rows of an array, but here certain elements in the row have particular meanings.

Dealing with Fast Data

Apart from fast memory, there are 4 general techniques for dealing with fast data:

1. Sample: stats is the answer.
2. Aggregate: more on this in a minute.
3. Filter (or pre-filter): not useful for generic problems, but can answer specifics, if filtering is fast enough
4. Parallelise: Hadoop it.

Underlying this, however, should be a realisation that there are several different classes of operations we want to do, and different approaches work for each:

1. Operate per item:
    - *e.g.,* fraud detection on bank transactions: can’t sample or aggregate, filtering is slow, so parallelise and pre-filter.
    - *e.g.,* packet forwarding: can’t sample or aggregate or filter, so parallelise.
2. Operate on groups,
    - $e.g.$, to collect data to optimise a network, sampling and aggregation work, but have to be based on relevant groups.
3. Save the data
    - Give up: the data is too big to save. :)

Less cynically, you save summaries (aggregates) of the data.

Unstructured Data

Most data we deal with is *structured*. For instance in tabular data the columns have meaning (they are variables) and a consistent type (per column). If you add new n samples, you increase the size of the data by $n \times k$ (where k is the size of one sample, which is constant). In image data, each image is the same size (often) or at least has a common format, structure and set of assumptions.

Some data sets are not so lucky.

- Text data is perhaps the simplest case: we don't have a pre-defined length of a novel, for instance.
- Twitter has structure, but ultimately comes in variable length chunks of text (Tweets are also fast: 6,000 per second, 500 million per day.)
- Similarly for emails or social media posts in general.
- Agglomerations of data such as a health record (that could include text, test results, histories, imaging, …).

Sometimes you have data that has structure, but that structure doesn't fit neatly into tables or tensors. For example a graph/network could be described by a (big) adjacency matrix, but that isn't a good way to work on the data. Geographic data can often be in this category as well, where it is structure in one sense, but that structure might not be a natural way to work with the data. We might call that data *semi-structured*.

How we define data (RE structure) comes down to

1. Whether there is a suscint and precise data model that defines how the data is organised and what it looks like.
2. Whether the data model provides a convenient way to work on the data, *e.g.,* to search or summarise it.

Structure may also be present, but we don't know about it.

It is claimed that the vast majority of data in business is unstructured. We could argue.

Dealing with unstructured data

The *Anna Karenina principle:* "All happy families are alike; each unhappy family is unhappy in its own way (Tolstoy)" applies doubly to unstructured data.

But there are some tools, for instance to deal with variable length data:

- Hashes
- Recursive algorithms (starting with the EWMA, then Kalman filter, then RNNs)

And these techniques often help with fast data as well.

More on how to deal

One approach I want to foster here is the use of randomised algorithms. More in a minute.

Links

- https://moviecultists.com/why-sram-is-faster-than-dram
- https://www.geeksforgeeks.org/what-is-content-addressable-memory/
- https://www.techopedia.com/definition/31631/ternary-content-addressable-memory-tcam
- https://www.netapp.com/pdf.html?item=/media/9517-na-305.pdf
- https://monkeylearn.com/unstructured-data/