

Algorithms for Data Science (Part 2 - Big Data Algorithms)

Daniel Lawson — University of Bristol

Lecture 08.2.2 (v1.0.1)

Signposting

- ▶ This lecture 8.2 of Algorithms for Data Science
- ▶ The lecture is in two parts:
 - ▶ Part 1 Data Structures
 - ▶ Part 2 Algorithms
- ▶ This is Part 2 on **Big Data Algorithms:**
 - ▶ Sampling for big data (Reservoir/non-uniform)
 - ▶ Bloom filters
 - ▶ Sketching
 - ▶ MinHash

Sampling (for big data)

- ▶ If there are N (large) items, how do we correctly sample n of them?
- ▶ Naive approach: read in the data, choose n at random, done.
- ▶ What if the data don't fit in memory? We might choose a subset e.g. by:
 - ▶ **Random sampling:** Choose each point with probability $p = n/N$
 - ▶ **Uniform sampling:** Choose every n/N th point
 - ▶ Efficiently?

Sampling (when we don't know N)

- ▶ **Reservoir sampling:**
 - ▶ Keep the first n items. For the remaining items i :
 - ▶ Accept the new item with probability n/i
 - ▶ discard uniformly from the n .
 - ▶ Otherwise, keep the old items.
- ▶ Weighted versions etc exist.
- ▶ Generates samples uniformly from the whole set of n with fixed storage.

Non-Uniform sampling

- ▶ Sometimes, most data is “boring”. We want to sample the “most useful” data.
- ▶ One solution is to divide the data into histogram bins and sample inversely with frequency using e.g. reservoir sampling within each
- ▶ How to choose the bins?
 - ▶ Choice in advance requires knowledge of the data, or looking at it already
 - ▶ Dynamic approaches are possible where the bins are learned in a **streaming** manner¹
 - ▶ The algorithm can be tuned for estimating particular quantities, e.g. the mean²

¹Streaming histogram implementation

²Risto Tuomainen Data Sampling for Big Data

Filtering

- ▶ Filters have the goal of retaining information regarding which data have previously been seen, **without storing it** all.
- ▶ Example: we have a datastream of (many) observed MAC addresses from users.
 - ▶ Question: have we seen value x before?
 - ▶ Can we do this with **constant cost** $\Theta(1)$ per item?

Bloom Filter

- ▶ A **bloom filter** can tell in constant time whether:
 1. a data point is not in the database
 2. a data point might be in the database
- ▶ It does this by storing all of the observed data solely as a hash $h(x) \rightarrow (0, r]$.
 - ▶ The data are stored as a bitvector b_r .
 - ▶ The larger the range, the more precise the answer will be but the greater the cost.
 - ▶ For each datapoint x_i we:
 1. Compute k hashes in $[0, r)$, $h_k(x_i)$
 2. Set all bits hashed into to one, i.e. $b_r(h_k(x_i)) = 1$
 - ▶ At lookup time: if any $b_r(h_k(x_i)) = 0$ then we have not seen this item before.
- ▶ See Bill Mill's excellent Bloom filter practical

Choosing parameters for a bloom filter

- ▶ There are three variables: the **number of data expected** to be stored, n , the **number of hashes** k and the **length of the bitvector** r .
- ▶ The **error rate** is expected to be $(1 - \exp(-kn/r))^k$
- ▶ It turns out that this is minimised when $k = r/n \ln(2)$
- ▶ You then trade off error rate for storage size (for the bit vector) and compute cost (for the hashes)
- ▶ Bloom Filters are very useful, for example in Network analysis³ and Network Security⁴

³Broder & Mitzenmacher “Network Applications of Bloom Filters: A Survey” (2003) Internet Mathematics 1:485-509

⁴Geravand & Ahmadi “Bloom filter applications in network security: A state-of-the-art survey” (2013) Computer Networks 57:4047-4064

Sketching

- ▶ Sketching is obtaining the frequency properties of your data from a data stream.
- ▶ One important class is probabilistic counting, which addresses how many of each class there are.

Count-min-sketch

- ▶ Count-min-sketch works just like a bloom filter, except that we store an integer for each has rather than a single bit.
- ▶ We initialise $b_r = 0$, and then:
 1. Compute k hashes in $(0, r]$, $h_k(x_i)$
 2. Add one to all bits hashed into, i.e. $b_r(h_k(x_i)) + = 1$
- ▶ At lookup time, the number of items is estimated to be

$$\operatorname{argmin}_{h_k(x_i)} b_r(h_k(x_i))$$

i.e. the minimum count.

- ▶ See e.g. Python implementation of Count Min Sketch by Rafael Carrascosa (part of PyPI)

Other important algorithms:

- ▶ The **MinHash** algorithm quickly computes similarities between sparse feature vectors such as **documents**.
- ▶ **Locality Sensitive Hashing** reduces the dimensionality of data by representing an object as a set of hashes, chosen so that “similar” items have “similar” hash values
- ▶ The **Hashing Trick** is a Machine-Learning tool for turning arbitrary objects into features - just take one or more locality sensitive hashes of the object as new features.
- ▶ There are a range of sketches with different biases, such as the Count-Mean-Sketch and others⁵.

⁵Goyal, Daume & Cormode “Sketch Algorithms for Estimating Point Queries in NLP” (2012) Proc. EMNLP.

MinHash motivation

- ▶ Consider a very large, potentially sparse, **binary** feature space for which we have observations $A = \{x_i\}$ and $B = \{x_k\}$. How similar are they?
- ▶ One natural measure is the **Jaccard Similarity**:

$$J(x_i, x_j) = \frac{x_i \cap x_j}{x_i \cup x_j}$$

- ▶ This is slow to compute with a large sparse features space, such as **words**.
- ▶ The solution is to approximate the similarity via MinHash.

MinHash algorithm

- ▶ To compute a single MinHash Signature:
 - ▶ Use a **random hash function** and apply it to all values in A and B .
 - ▶ Compute the minimum of each of these.
 - ▶ The probability of these being equal turns out to $J(A, B)$.
- ▶ To estimate J , we simply do this several times.
- ▶ This was used for website Duplicate detection by AltaVista and was confirmed to be still in use by Google in 2007. There are a lot of websites...
- ▶ See e.g. Chris McCormick's Minhash tutorial or the Mining of Massive Datasets book and course.

Discussion

- ▶ Exploiting convenient algorithms forms a key part of many high-throughput models.
- ▶ Many data streams, especially cyber, have a **power-law** distribution of activity: much of the data are seen only once, whilst some **heavy hitters** might make up the majority of the dataset.
- ▶ Identification of heavy hitters and singletons allows them to be treated specially which can massively reduce computational burden.
- ▶ For example, to process a massive cyber dataset:
 - ▶ Use a **Bloom filter** to store only information on IP Addresses you've seen more than once,
 - ▶ A **Count-Min-Sketch** to identify heavy hitters,
 - ▶ Store the remaining data in a suitable **hash table**,
 - ▶ On which you construct a model.

Reflection

- ▶ How could you use these data structures and algorithms in your assessments?
- ▶ To what extent do you need to understand them in order to gain value in data science?
- ▶ By the end of this course, you should:
 - ▶ Be able to work with and recognise the **dynamic data structures** (Queues, Stacks, Hash tables, Binary Trees, Linked Lists)
 - ▶ Be able to recognise and exploit **simple algorithms** (Samplers, Filters, Sketching, MinHash)
 - ▶ Relate their use to Big Data problems

Signposting

- ▶ This is the end of the lecture content.
- ▶ The workshop is very short due to the extra theoretical content.
- ▶ Next block in 09: Neural Networks.

References

- ▶ Advanced algorithms:
 - ▶ The Mining of Massive Datasets book and course.
 - ▶ Risto Tuomainen Data Sampling for Big Data, covering sampling, filtering, sketching, etc.
 - ▶ Streaming histogram implementation
 - ▶ Bill Mill's excellent Bloomfilter practical
 - ▶ Chris McCormick's Minhash tutorial
 - ▶ Python implementation of Count Min Sketch by Rafael Carrascosa (part of PyPI)
 - ▶ Leo Martel notes on Streaming Data Algorithms which is notes on the paper
 - ▶ Cormode's notes on [Count-Min Sketch]<http://dimacs.rutgers.edu/~graham/pubs/papers/cmencyc.pdf>
 - ▶ Chakrabarti's Lecture Notes on Data Stream Algorithms
 - ▶ Broder & Mitzenmacher "Network Applications of Bloom Filters: A Survey" (2003) Internet Mathematics 1:485-509
 - ▶ Geravand & Ahmadi "Bloom filter applications in network security: A state-of-the-art survey" (2013) Computer Networks 57:4047-4064
 - ▶ Goyal, Daume & Cormode "Sketch Algorithms for Estimating Point