CS 677: Big Data

Stream Sampling

Lecture 12

Data Reduction via Sampling

- One way to make the dataset more manageable is to generate a representative **sample** and analyze it
  - If the rules aren't working in your favor, change the rules!
- The key is knowing **when** you can sample and what **algorithms** work best

Today's Schedule

- Sampling Overview
- Stream Sampling
- Gap Sampling
- Reservoir Sampling

Today's Schedule

- **Sampling Overview**
- Stream Sampling
- Gap Sampling
- Reservoir Sampling

Sampling

- Ok, so we can mostly agree that one great way to deal with big data is to make it less "big"
  - Dividing up the problem into smaller pieces is one way to do this
- Another simple way to achieve this: sample from our dataset
  - If the sample is **representative**, then it will serve as a good stand-in for the actual, large dataset
  - Sample vs. census: asking **some** instead of **all**

Hmm…

- …isn't this cheating?!
  - Wait! Don't submit your course drop forms yet!
- We can actually do a pretty good job with just a small sample

Approximation

- What's an easy way to speed up processing 1 ZB of data? Ignore almost all of it!
- Let's take a step back and think for a second here, though… What are we losing?
  - A big one: less-represented data points are likely going to be lost
    - We have to be careful what conclusions we draw

Sampling Algorithms

- You might be sitting there in horror right now, thinking “Is Matthew really going to talk about generating a random sample for 3 hours?”
  - (Yes, in this hypothetical situation class actually goes longer than usual just to torture you)
- Luckily, we get to cover some big data-specific algorithms:
  - Stream sampling
  - Gap sampling
  - Reservoir sampling

Implementing our Sample

- The naïve approach: if we have 100 data points and want a 10% sample, randomly select 10
  - Sampling with replacement: put the selected data points back into the dataset after each selection
    - Thought experiment: what dataset does this make sense for?
- In code: pick 10 unique indices, grab the data. Done!
- What about in a distributed setting?
  - Oh, right, that's where things start to get difficult…

Distributed Sampling

- To get started, we can just divide up the work and sample X% from each data partition
- Combine the samples into one bigger sample
- This is pretty decent. It works… unless we don't know how many records we're going to get at each task
  - Maybe we don't even know the total number of inputs we're going to get
    - *Streaming data!*

More Complications

- Let's say we can find out how many records will be assigned to each mapper (or distributed task)
- We may still want additional filtering, for instance removing invalid readings
  - Now we need to know the number of records to remove, and the number of incoming records
- Adding more constraints makes this even more difficult

Multiple Passes

- We can go through the data as a preprocessing step to determine these parameters, then sample it
- The problem: this takes time
  - We are relying on spinning rust to get this work done
- Avoid making multiple passes over the data!

Your Options

1. Don't touch the data at all
2. Only touch the data **once**
3. Wait a really, really, really long time

An Aside: Streaming Data

- I have already mentioned that almost all big data problems can be viewed as streaming data problems
- The reason for this is simple: most of the time you can't make multiple passes over the data
  - It's too big to do that efficiently!
- So, when you're dealing with a VERY large dataset, reach for streaming algorithms

Reconfiguring our Algorithm

- Alright, so we can't assume we know the number of records handled by each task
- Instead, we can reduce the amount of **state** information required
- Basically, can we forget about everything we've done in the past but still sample accurately when we're looking at a single data point?

Today's Schedule

- Sampling Overview
- **Stream Sampling**
- Gap Sampling
- Reservoir Sampling

Stream Sampling

- Inspect each data point in isolation, and flip a coin
  - Heads = sample it
  - Tails = ignore it
- This gives us a 50% sample
- If we want a 10% sample, select a random number from 0.0 to 1.0
  - Only keep the data point if the random number is 0.0 to 0.1
    - (or whatever range represents 10% of the possible values)

Stream Sampling: Pros and Cons

- **Pros:**
  - Easy to write; conceptually simple
  - No need for any extra information
- **Cons:**
  - Invokes the random number generator a lot
    - Actually can add up over time
  - We have to parse every input
  - We won't get an exact sample
    - May be a bit more or less than say, 10%

Today's Schedule

- Sampling Overview
- Stream Sampling
- **Gap Sampling**
- Reservoir Sampling

Gap Sampling

- To reduce the amount of data we parse, let's skip over records that won't be sampled
  - Works for a stream of unknown size
- Start by skipping a random amount of records. If we want a 10% sample, skip 0 to 10 records
- After sampling the first data point, just keep skipping ahead by 10 records
- Decide you want a 50% sample instead? Skip every other record.

Gap Pros and Cons

- **Pros:**
  - We can actually avoid processing records! (Speed!)
- **Cons:**
  - To be a ***true*** random sample, all data points must have an opportunity to be picked
    - We only kind of satisfy this constraint
    - We can modify this slightly. Instead of moving ahead 10 records, we could add some random noise to make sure we move ahead by an average of 10 records
  - Once again, we might not get the exact sample size we're hoping for

Today's Schedule

- Sampling Overview
- Stream Sampling
- Gap Sampling
- **Reservoir Sampling**

Reservoir Sampling

- Useful when the size of the incoming stream is unknown or there are memory constraints
- Initialize as a fixed size array on creation
  - Limits memory usage
- As data points stream in, place them at random array indexes
- Over time, update the array less and less
  - Ensures long-term representativeness

Intuition: Elimination Game

- Let's say I'm going to give one lucky winner an "A" in the class, right now
- I'll pick two students and have them flip a coin
  - Heads: Student 1 survives, Student 2 is eliminated
  - Tails: Student 2 survives, Student 1 is eliminated
- Then I'll pick the next "challenger" for the coin flip
- The last student standing (i.e., not eliminated) is the winner and receives an A
- Ready to play? Are the rules fair?

Eliminated!

- No, the rules aren't fair!
  - (When are they?!)
- The first two students have the worst chance of survival
- Even if you are the luckiest person on earth, there is a not a great chance you'll win 17 coin flips
  - Unless you are using a trick coin…
- So the last student to play has a HUGE advantage
  - A more intuitive example: arm wrestling competition with the same rules
- How do we fix this issue?

Reservoir Algorithm

- Online sampling technique that creates representative random samples when:
  - The number of incoming data points in unknown
  - The total dataset cannot fit in main memory
    - Fixed size ($\boldsymbol{n}$)
- When data points arrive, they are assigned a random insertion key ($\boldsymbol{k}$) in the range $[\mathbf{0}, \mathbf{1}]$
- If $\boldsymbol{k} < \frac{n}{C}$, where $\boldsymbol{C}$ is the total number of observations, the data point replaces a random entry in the reservoir
  - The probability of replacement decreases over time

Reservoir Sampling Extensions

- Reservoir sampling can be augmented by allowing sample weights to increase the likelihood of certain data points being placed in the array
  - We may place a greater weight on samples from a particular sensor, for instance
- Additionally, storing the insertion key when placing data in the reservoir allows merging later
  - To determine which elements go in the merged arrays, just sort by insertion key

Distributed Reservoirs

- Each map task maintains a reservoir of size n
  - Insert each record into the reservoir
  - If the record gets stored in the reservoir, also store its insertion key (the random number associated with it)
- At the end of the Map phase, emit n entries, plus their insertion keys
- During the reduce (on a single reducer) keep the elements with the smallest insertion keys

Representativeness

- While reservoir sampling provides a replacement for our standard random sampling procedure, it does have weaknesses
- The sample must fit into memory (generally acceptable)
- Outliers or uncommon values will be under-represented

Stratified Sampling

- Sometimes the outliers are actually more interesting than the common cases!
- Here, we can use **stratified sampling** to produce a sample that better represents all populations rather than just the majority
- Observe the distribution of data points, and then create sub-reservoirs across the distribution
  - Uncommon data points now have their own reservoir and won't be overpowered by the majority

Needle in a Haystack

- Ultimately, if I ask you to find a specific record, sampling won't help
- There's a good chance it won't even be in your sample
- Sampling **is** appropriate for quickly gaining aggregate knowledge

Saving Subsets

- If you're strategic, you can build smaller subsets of the overall dataset
  - Use these samples to do initial exploratory analyses
- Done frequently with data warehousing systems such as Hive
  - Build summary tables that answer certain "business questions" as a background batch process

To Conclude

- If you can still get a reasonably correct answer
  - Ideally a 100% correct answer…
- And you will probably reuse the sample more than once
- Then sample! (And use one of these Big Data-oriented algorithms!!)