Hydrus

Chris, Ankit, Vibodh, Vamsi

Estimating Gaussians

In distributed systems

The Normal Distribution

$$\mathcal{N}(\mu, \sigma^2)$$

The Normal Distribution – Reparametrized

$$\mathcal{N}(\mu, \sigma^2) = \mathcal{N}\left(\mu, \frac{M_2}{n}\right)$$

The Normal Distribution – Reparametrized

$$\mathcal{N}(\mu, \sigma^2) = \mathcal{N}(n, \mu, M_2)$$

Welford's Algorithm

- Given an estimated Normal distribution, $\mathcal{N}_t = \mathcal{N}\left(n_t, \mu_t, M_{2,t}\right)$
- And a new data point, x_{t+1}
- Update the distribution, $\mathcal{N}_{t+1} = \mathcal{N}\left(n_{t+1}, \mu_{t+1}, M_{2,t+1}\right)$

Welford's Algorithm

$$n_{t+1} = n_t + 1$$

$$\mu_{t+1} = \mu_t + \frac{x_{t+1} + \mu_t}{n_{t+1}}$$

$$M_{2,t+1} = M_{2,t} + (x_{t+1} - \mu_t)(x_{t+1} - \mu_{t+1})$$

Welford’s Algorithm

```
def welford(dist, x):
    (count, mean, m2) = dist
    new_count = count + 1
    new_mean = mean + (x - mean) / count
    new_m2 = m2 + (x - mean) * (x - new_mean)
    return (new_count, new_mean, new_m2)
```

Welford's Algorithm

- Pros:
  - Easy to implement
  - Optimal time complexity: $O(n)$
- Cons:
  - Not parallel

Chan's Algorithm

- Given a dataset, X_{AB}
- Partitioned into two, X_A and X_B
- With distributions for both, $\mathcal{N}_A$ and $\mathcal{N}_B$
- Compute the combined distribution, $\mathcal{N}_{AB}$

Chan’s Algorithm

$$\delta = \mu_B - \mu_A$$

$$n_{AB} = n_A + n_B$$

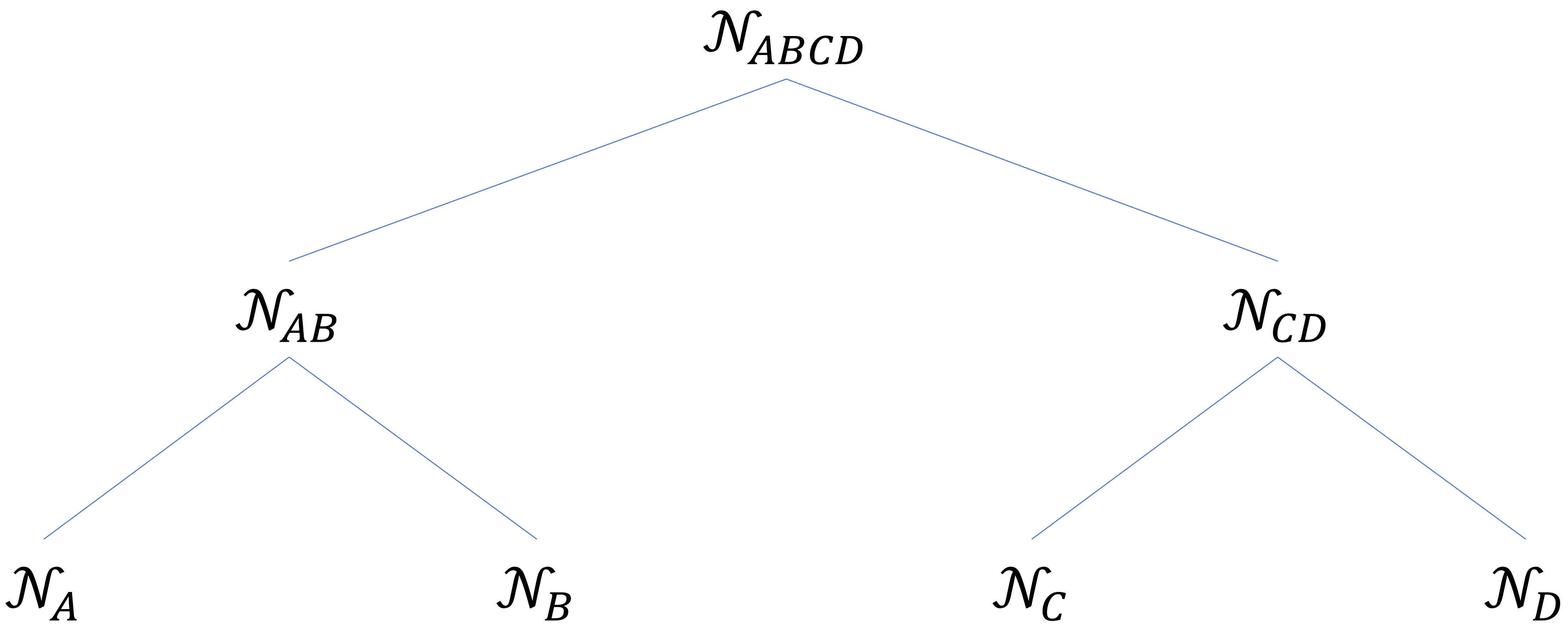
$$\mu_{AB} = \mu_A + \delta \cdot \frac{n_B}{n_{AB}}$$

$$M_{2,AB} = M_{2,A} + M_{2,B} + \delta^2 \cdot \frac{n_A n_B}{n_{AB}}$$

Chan's Algorithm

```python
def chan(dist_a, dist_b):
    (count_a, mean_a, m2_a) = dist_a
    (count_b, mean_b, m2_b) = dist_b
    delta = mean_b - mean_a
    count = count_a + count_b
    mean = mean_a + delta * count_b / count
    m2 = m2_a + m2_b + delta**2
    m2 *= count_a * count_b / count
    return (count, mean, m2)
```

Chan’s Algorithm



Chan’s Algorithm

- Pros
  - Easy to implement
  - Parallel
- Cons
  - Parallelism at the cost of CPU time
  - Possible stability issue with μ_{AB}
    - Alternate formula on Wikipedia

Spark

- Spark has methods like `RDD.mean` and `RDD.variance`
  - But not `RDD.meanByKey` or `RDD.varianceByKey`
  - Two passes over the data, one for each op
- We could use `RDD.map` and `RDD.reduceByKey`
  - Forces a shuffle
  - Easy to shoot your foot with a dumb initial key (n or M_2)
- Instead we have `RDD.aggregateByKey`

Spark

```
RDD.aggregateByKey((0,0,0), welford, chan)
```

Spark

Zero-value
bootstraps the online algorithm

Pairwise algorithm
combines the per-partition values

```
RDD.aggregateByKey((0,0,0), welford, chan)
```

Online algorithm
estimates the distribution per partition

☆ Bonus ☆

Matrix Multiplication

with Spark RDDs

Representation

$$
i\left\{\overbrace{\begin{bmatrix} x_{0,0} & x_{0,1} \\ x_{1,0} & x_{1,1} \\ x_{2,0} & x_{2,1} \end{bmatrix}}^{j}\right.
$$

Representation

$$\mathtt{RDD[((}i\mathtt{,}\ j\mathtt{),}\ x_{ij}\mathtt{)]}$$

Representation

$$((i,\ j),\ x_{ij})$$

Matrix Multiplication

$$((i,\ j),\ x_{ij}) \times ((j,\ k),\ y_{jk})$$

$$= ((i,\ k),\ \textstyle\sum_j x_{ij} y_{jk})$$

Step 0 – Start

$$((i,\ j),\ x_{ij}) \times ((j,\ k),\ y_{jk})$$

Step 1 – Rekey

$$(j,\ (i,\ x_{ij}))\ \times\ (j,\ (k,\ y_{jk}))$$

Step 2 – Join

$$(j,\ ((i,\ x_{ij}),\ (k,\ y_{jk})))$$

Step 3 – Multiply

$$(j,\ ((i,\ k),\ x_{ij}y_{jk}))$$

Step 4 – Drop j

$$((i,\ k),\ x_{ij}y_{jk})_j$$

Step 5 – Sum by key

$$((i,\ k),\ \sum_j x_{ij} y_{jk})$$

Step 6 – Profit!

- Ideal for sparse matrices
  - E.g. the document-term matrix
  - If the value is zero, don’t include it in the RDD
  - The join step implicitly handles the zeros!
- i, j, and k don’t need to be integers
  - Instead, use semantic values like document ID, word, and label

References

- Estimating Gaussians

  - https://en.wikipedia.org/wiki/Algorithms_for_calculating_variance
  - B. P. Welford (1962), "Note on a method for calculating corrected sums of squares and products" (JSTOR), *Technometrics* 4(3):419–420
  - *Chan, Tony F.; Golub, Gene H.; LeVeque, Randall J. (1979), "Updating Formulae and a Pairwise Algorithm for Computing Sample Variances." (PDF), Technical Report STAN-CS-79-773, Department of Computer Science, Stanford University*.