MSA220 - Statistical Learning for Big Data

Lecture 15

Rebecka Jörnsten

Mathematical Sciences
University of Gothenburg and Chalmers University of Technology

Big n!

Our final theme!

- When the sample size is large, there's a couple of things we need to be concerned about
- Computations can become impossible or slow, even for simple statistical tasks
- Storage issues
- Visualization and model diagnostics

Big n statistics

3 main approaches

- Subsampling
- Divide and Conquer/Split and Merge/Divide and Recombine
- Online updating

Big n statistics

Different attitudes/goals with these approaches

- Subsampling
  - Each subsample to provide a good estimate of the parameter of interest.
  - Simple aggregation with means
- Divide and Conquer
  - Work out analytically how parameter estimates from each split should be combined to produce the estimate you would have obtained had you used the full sample
- Online updating
  - When data arrives sequentially

BLB - bag of little bootstraps

Understanding the BLB vs bootstrap

- True distribution P
- Empirical distribution P_n with mass $1/n$ at each observed sample point
- Parameter of interest θ
- Sampling distribution $\hat{\theta}_n \sim Q_n(P)$
- Goal: Get an estimate of $\Psi(Q_n(P))$, e.g. CI, SE - that we can use to draw inferences about θ

BLB - bag of little bootstraps

Understanding the BLB vs bootstrap

- Goal: Get an estimate of $\Psi(Q_n(P))$, e.g. CI, SE
- We don't know P and therefore not $Q_n(P)$ either
- Sometimes we can work out the latter given assumptions on P (e.g. assume normally distributed errors, then get a normally distributed regression coefficient estimates (if known σ^2, o/w t).
- If we can't work it out, have to estimate it.

BLB - Bag of little bootstraps

Understanding the BLB vs bootstrap

- Bootstrap: Plug-in estimate $\Psi(Q_n(P)) \simeq \Psi(Q_n(P_n))$
- Can't compute $\Psi(Q_n(P_n))$ directly usually BUT we can use simulations to estimate it!
- Draw data from P_n repeatedly (b, bootstrap), each time compute your estimate $\hat{\theta}_n^b$
- The empirical (observed) distribution of your estimate $\hat{\theta}_n^b \sim W_n$
- Final answer: $\Psi(Q_n(P)) \simeq \Psi(W_n)$

BLB - bag of little bootstraps

Understanding the BLB vs bootstrap

- Final answer: $\Psi(Q_n(P)) \simeq \Psi(W_n)$
- For W_n to be a good substitute for $Q_n(P_n)$ it needs to "behave" the same. The sample size from simulation has to comparable to the original sample size
- Computationally burdensome if n is large!
- m-out-of-n bootstrap: good properties, smaller sample size to work with
- Problems?

Understanding the BLB vs bootstrap

- m-out-of-n bootstrap: good properties, smaller sample size to work with
- We estimate $W_{n,m}$: bootstrap sampling distribution of $\hat{\theta}$ in m-out-of-n bootstrap
- Final estimate: $\Psi(Q_m(P)) \simeq \Psi(W_{n,m})$
- Problems?
- IF we know the convergence rate of the parameter of interest (e.g. $SE(\hat{\theta}) \sim 1/\sqrt{n}$)
- THEN we can correct the m-out-of-n estimate by factor (here $\sqrt{m/n}$)

Bag of Little Bootstraps

- We draw s subsets of data of size $m < n$
- For each of the s subsets, draw r samples of size n
- Consider subset $j \in \{1, \cdots, s\}$: we draw data from the empirical distribution $P^j_{n,m}$
- Each bootstrap sample in j is of size n!
- Estimate: $s^{-1} \sum_{j=1}^{s} \Psi(Q_n(P^j_{n,m}))$
- We don't know $Q_n(P^j_{n,m})$ so this is where the r bootstraps for each subset comes in

Bag of Little Bootstraps

- Estimate: $s^{-1}\sum_{j=1}^{s}\Psi(Q_n(P_{n,m}^{j}))$
- We don't know $Q_n(P_{n,m}^{j})$ so this is where the r bootstraps for each subset comes in
- Draw r samples of size n from $P_{n,m}^{j}$ and estimate parameter of interest
- The observed bootstrap sampling distribution is denoted $W_{n,j}$
- Final estimate: $s^{-1}\sum_{j=1}^{s}\Psi(W_{n,j})$

Bag of Little Bootstraps

- Wait a minute! Didn't this just make the computations explode?
- Actually, no - drawing a sample of size n from the subset s of size m is equivalent to assigning *weights* to the m observations in s
- So the computation is actually performer only on the smaller sample size m

Bag of Little Bootstraps

The algorithm

- For $j = 1, \cdots, s$, draw a sample of size m (or disjoint partition of the original data)
    - For $k = 1, \cdots, r$,
    - Draw weights from Multinomial(n,m)
    - Estimate your statistics of interest
- Combine by averaging quantities of interest across s (e.g. estimates, lower and upper CI limits, etc)

Bag of Little Bootstraps

- Recommended size of $m = n^\gamma$, $\gamma \in [.5, 1]$
- In the original BLB paper (Kleiner et al, 2014) they use $\gamma = 0.6$ (reducing a data set of 10^6 to about 4000 for computation).
- Kleiner et al found that BLB is fairly robust to choices of m, consistency of estimates and good convergence rates
- Completely parallelizable for each set of size m so allows for fast and scalable computing
- Implemented in the `datadr` R package

Leveraging

- Another variant for subsampling was proposed by Ma and Sun (2013)
- Like the BLB, they suggest that we estimate model parameters from a much smaller data set and then combine the results
- However, they differ in how the subsampling is done
- Idea is to create ONE reduced sample that represents the full data and can be used as a proxy for analyzing the full data.

Leveraging

- Recap from regression
- $y = X\beta + \epsilon$
- LS: $\min_\beta ||y - X\beta||^2$
- $\hat{\beta} = (X'X)^{-1}X'y$
- $\hat{y} = X\hat{\beta} = X(X'X)^{-1}X'y = Hy$

Leveraging

- Specifically, $\hat{y}_i = \sum_j h_{ij} y_j$
- Element h_{ii} is called the *leverage* of observation i, i.e. how much it influences its own fitted values
- Leverage basically captures if observation i is close or far from the center of the data. Observations near the center (in X-space) have limited contribution to the fit.

Leveraging

- Sample r observations from the original n where $r << n$
- The sampling probability π_i for observations i is $\pi = \frac{h_{ii}}{\sum_j h_{jj}}$
- Estimate to regression parameters
  - Alt 1: use standard OLS
  - Alt 2: use weighted LS, where the weights are the inverse sampling probabilities

Leveraging

- Two alternatives: different goals!
  - Alt 1: use standard OLS
  - Alt 2: use weighted LS, where the weights are the inverse sampling probabilities
- Ma and Sun show that WLS with weights $1/\pi_i$ results in an unbiased estimate of the regression coefficients you would have gotten had you analyzed the full data!
- BUT, the OLS estimate, while biased for full-data estimate, is an unbiased estimate of the TRUE coefficient and has a smaller variance too.

Leveraging

- Ma and Sun noticed that the WLS can be sensitive to the smaller values of π_i in the sample and lead to increased variance
- They propose to regularize the sampling probabilities

$$\pi_i = \alpha \frac{h_{ii}}{\sum_j h_{jj}} + (1-\alpha)\frac{1}{n}$$

- α around 0.8-0.9 recommended values.

Leveraging

- Fast SVD computation for the leverage makes the method fast and easy to use
- $X = UDV'$
- $H = X(X'X)^{-1}X' = UU'$
- and so $h_{ii} = ||u_i||^2$ for u_i i-th row in U

Leveraging

- Fast and simple
- A bit careful about outliers....
- A big pro: can use the subsample to visualize the data
- Model diagnostics in a big-n world - and we could remove outliers at this point...

Maximum MEAN Likelihood

Liang et al 2013

- So far we mostly talked about regression models
- What about more general models that we estimate with MLE?
- Computationally prohibitive for large *n* - what to do?
- Likelihood approximation (use a more simple model essentially)
- New proposal: resample-based approximation

Maximum MEAN Likelihood

- IDEA: find estimate θ by minimizing the Kullback-Leibler divergence between our model distribution f_θ and the unknown true distribution g

$$KL(f_\theta, g) = E_g[\log(\frac{f_\theta}{g})]$$

- We can approximate the KL distance

$$KL = C - \binom{n}{m}^{-1} \sum_{i=1}^{\binom{n}{m}} \log f_\theta(y_i)$$

- Notice, this is an approximation based on all subsamples of size m from the data (of course we can use fewer for a more coarse approximation).

Maximum MEAN Likelihood

- We have our approximate loss function

$$KL = C - (\binom{n}{m})^{-1} \sum_{i=1}^{\binom{n}{m}} \log f_\theta(y_i)$$

- Now we minimize this with respect to θ
- A system of equations

$$\frac{\partial KL}{\partial \theta} = -\binom{n}{m}^{-1} \sum_{i=1}^{\binom{n}{m}} \frac{\partial log f_\theta(y_i)}{\partial \theta} = -\binom{n}{m}^{-1} \sum_{i=1}^{\binom{n}{m}} \nabla_\theta \log f_\theta(y_i)$$

Maximum MEAN Likelihood

- A system of equations

$$\frac{\partial KL}{\partial \theta} = -\binom{n}{m}^{-1} \sum_{i=1}^{\binom{n}{m}} \frac{\partial logf_\theta(y_i)}{\partial \theta} = -\binom{n}{m}^{-1} \sum_{i=1}^{\binom{n}{m}} \nabla_\theta \log f_\theta(y_i)$$

- where

$$\nabla_\theta(\log f_\theta) = [\frac{\partial \log f_\theta}{\partial \theta_1}, \frac{\partial \log f_\theta}{\partial \theta_2}, \cdots, \frac{\partial \log f_\theta}{\partial \theta_p}]'$$

Maximum MEAN Likelihood

The ALGORITHM

- Initialize the parameter estimate vector θ_0
- For $t = 1, \cdots, T$, draw a sample of size m from the full data, without replacement
- Update each parameter estimate as

$$\theta_j^{t+1} = \theta_j^t + a_{t+1} \nabla_{\theta_j} \log f(\theta^t, y_t)$$

- *That is, use the gradient vector based on new data and the previous estimate of the parameters!!!*
- There is a check-point before the estimate is accepted, the new estimate can't be too far off the previous one (technical details in the paper beyond the scope of this class).

Maximum MEAN Likelihood

The ALGORITHM

- Initialize the parameter estimate vector θ_0
- For $t = 1, \cdots, T$, draw a sample of size m from the full data, without replacement
- Update each parameter estimate as

$$\theta_j^{t+1} = \theta_j^t + a_{t+1} \nabla_{\theta_j} \log f(\theta^t, y_t)$$

- a_{t+1} is a learning rate parameter
- You also need a stopping criteria (T), e.g. when CIs have a certain volume or some convergence criteria.

Divide and Conquer

- Idea is to split the data into K chunks
- Estimate your model parameters on each chunk separately
- Combine the estimates into a final estimate

Divide and Conquer

- Simple enough!
- Easy when we have linear models because the recombination of the estimates is straight forward
- Example from regression
- N samples: $\hat{\beta} = (X'X)^{-1}X'Y$ where X is the $N \times p$ design matrix and Y is the $n \times q$ response data. ($q = 1$ for standard regression)
- Now, let's say we had divided the data into K chunks X_k, Y_k
- Estimates $\hat{\beta}_k = (X_k'X_k)^{-1}X_k'Y_k$
- How do we combine the $\hat{\beta}_k$ to get the original $\hat{\beta}$?

Divide and Conquer

- Estimates $\hat{\beta}_k = (X_k' X_k)^{-1} X_k' Y_k$
- How do we combine the $\hat{\beta}_k$ to get the original $\hat{\beta}$?
- NOT a simple average

$$\hat{\beta} = (X'X)^{-1}X'Y = (\sum_k (X_k' X_k))^{-1} (\sum_k X_k' Y_k) =$$

$$= (\sum_k (X_k' X_k))^{-1} (\sum_k (X_k' X_k)(X_k' X_k)^{-1} X_k' Y_k) =$$

$$= (\sum_k (X_k' X_k))^{-1} (\sum_k (X_k' X_k) \hat{\beta}_k)$$

- Consider that last line...

Divide and Conquer

- Combined estimate

$$\hat{\beta} = (\sum_k (X_k' X_k))^{-1} (\sum_k (X_k' X_k) \hat{\beta}_k)$$

- Now, what do you know about regression coefficient estimates?
- $Var(\hat{\beta}_k) = \sigma^2 (X_k' X_k)^{-1}$
- So the combined estimate is similar to a weighted average with weights inversely proportional to the variance of each chunk estimate!
- What do we have to do in practice?
- Save $\hat{\beta}_k$ and $X_k' X_k$ for each chunk!!!

Divide and Conquer

- This worked out fine because our estimates were linear
- What about more complicated operations, like nonlinear models - how do we combine then?
- Lin and Xi (2011) proposed the following
- Consider a general estimation problem
- You need to solve the *score equation*

$$\sum_i \Psi(y_i, \theta) = 0$$

- Example: Normal equations in regression $\sum_i (y_i - x_i'\beta)x_i = 0$
- Example: MLE $\sum_i \frac{\partial \log f_\theta(y_i)}{\partial \theta} = 0$

Divide and Conquer

- You need to solve the *score equation*

$$\sum_i \Psi(y_i, \theta) = 0$$

- Example: Normal equations in regression $\sum_i (y_i - x_i'\beta)x_i = 0$
- Example: MLE $\sum_i \frac{\partial \log f_\theta(y_i)}{\partial \theta} = 0$
- In the regression case the scoring equation is linear and that's what makes this work so easily
- In the general case, the equation system can be quite nonlinear.

Divide and Conquer

- For chunk k of data we solve

$$M_k(\theta) = \sum_{i \in k} \Psi(y_i, \theta) = 0$$

- Denote the estimate $\hat{\theta}_{n,k}$
- We compute

$$A_k = -\sum_{i \in k} \frac{\partial \Psi(y_i, \theta)}{\partial \theta}|_{\hat{\theta}_{n,k}}$$

- and *linearlize the scoring equation* (1st order Taylor expansion)

$$M_k(\theta) \simeq A_k(\theta - \hat{\theta}_{n,k})$$

- The approximate solution to the global scoring equation is

$$\sum_k M_k(\theta) = \sum_k A_k(\theta - \hat{\theta}_{n,k}) = 0$$

- which can be solved as

$$\hat{\theta} = (\sum_k A_k)^{-1} (\sum_k A_k \hat{\theta}_{n,k})$$

Divide and Conquer

- The solution for the nonlinear problem now looks very similar to the regression example

$$\hat{\theta} = (\sum_k A_k)^{-1}(\sum_k A_k \hat{\theta}_{n,k})$$

- AND, if you recall what you know about MLE....

$$A_k = -\sum_{i \in k} \frac{\partial \Psi(y_i, \theta)}{\partial \theta}|_{\hat{\theta}_{n,k}}$$

- The expected value of A_k if called the *Information matrix* and its inverse is the asymptotic variance of the MLE!!!
- So, the solution above is also a kind of weighted average of estimates with weights inversely proportional to the estimation variance!!!

Divide and Conquer

THE ALGORITHM

- Partition the data into K chunks that can fit in computer memory
- Compute $\hat{\theta}_{n,k}$ and A_k for each chunk, then disregard raw data
- Combine the estimates

$$\hat{\theta} = (\sum_k A_k)^{-1}(\sum_k A_k \hat{\theta}_{n,k})$$

Divide and Conquer

- Properties of combined estimates?
- As long as K doesn't grow too fast one can show that the combined estimate are consistent and asymptotically equivalent to the estimates you would have gotten had you used the full sample
- This linearization approach can be used to extend divide and conquer to very complex problems

Split and Conquer

Chen and Xie (2014) proposed a divide and conquer method for model selection (LASSO)

- Remember our discussion about p-values for large p problems?
- One approach (Meinhausen and Buhlmann) was to split the data and do model selection on one split and compute p-values using only the selected variables on the other split.
- Here, Chen and Xie take a similar approach for the purpose of big-n modeling

Split and Conquer

- Split the data into K chunks
- On each chunk, run penalized regression
- For each chunk, a different number and set of variables may have been selected
- The final set of selected variables are defined as those that are selected in at least w chunks (where w is a tuning parameter)
- Like the above Divide and Conquer scheme, a final estimate is obtained as a weighted average
- Careful: only those estimates that are non-zero contribute to the final estimate!

Split and Conquer

- Chen and Xie show that their final estimate (weighted average of selected coefficients) is asymptotically equivalent to the penalized regression estimates you would have gotten using the full data
- Can be generalized to other kinds of penalized models for big-n and big-p modeling!

Online methods

- Sometimes data arrives sequentially.
- Or it only possible to hold a subset of data in memory and we want to avoid having to create "chunks" and cycle
- Online methods pass data objects through only once and instead keeps summary statistics or model updates
- May require changes to algorithms/methods since many are based on repeated cycles through objects
- AND makes possible to develop methods that adapt to a changing structure in the data stream (either class proportions, distributions of features etc).

Online clustering methods

- Usually a two-stage process (though one can also parameterize dynamics of cluster parameters)
- Online component: without deciding on explicit clusters, gather (adaptive) density information through "micro-clusters"
- Offline component: at any time during the stream, be ready to apply a clustering method where (weighted) micro-clusters are now the objects processed by e.g. kmeans or dbscan

Online clustering methods

Handling micro-clusters:

- Cluster features: reduce the clusters to statistics instead of keeping the objects in the clusters in memory
- CF (Cluster Features) = (LS, SS, N) where LS is the sum of the objects (a vector), SS is the sum of squares and N is the number of objects
- From the CFs we can compute cluster centroid and radius
- Also, a simple addtive update for the CF as a new object in the stream arrives and is allocated to the cluster

Online clustering methods

- Micro-clusters can be grid-based or density based
- Allocation can be to existing clusters or seed a new one, depending on current state of the micro-clusters
- To adapt to changing data streams or outliers: micro-clusters can "expire" (keep tabs on when generated and when last added observations too - timestamp). Can also address this with micro-cluster weights that are a function of the timestamp.

Online clustering methods

- Macro-clusters: off-line component
- Use the micro-clusters as objects in a clustering procedure, possibly with weights.
- Any-time output
- Since micro-clusters "fade" this allows for clustering structure to change over time

Online classification methods

- Methods should view data objects, process, and then discard them
- Requires a change in how methods are built
- Ensemble methods are popular for online learning since they allow for easy adaptation - by dropping, updating, generating members of an ensemble on the fly

Online classification methods

- Nonstationary vs stationary (should learners update or adapt - and, if so, how?)
- Chunks vs stream

Online classification methods

- How are ensembles updated?
  - adapt weights
  - retrain existing members
  - replace/remove members
  - create new members

Online bagging

- Classical bagging: require full data set to resample from
- Oza and Russel: sample weights for samples from Poisson(1) like sampling from binomial (sample with replacement) with probability (1/N) N times.
- Can generate the weights for observations without having access to the full sample.
- Update ensemble members

Online bagging - adaptive versions

- In leverage bagging, a higher weight for new members - more randomness in each learner
- Can add a change detector - are all ensemble members working well or should one be replaced?

Online methods - adaptive and fast

- A break-through paper in 2000 (Domingos, Hulten): VFDT (very fast decision trees) or Hoeffding trees
- How to build trees and ensembles online
- Idea: you don't need much data to decide on a "stump" - i.e. a one-split tree
- First part of stream picks root split, next part picks subsequent split. Splits are only made when enough data has arrived.
- Enough data? Hoeffding bound: $P(|X - E(X)| > \epsilon) < \delta$, where $\delta = 2e^{-2n\epsilon^2}$ -¿ need $n > log(2/\delta)/(2\epsilon^2)$

Online methods - adaptive and fast

- Lots of work expanding on this
- Adapt: ensemble members can be removed and new ones generated based on track of performance
- Adapt: can retrain inner nodes if there is sufficient evidence they are no longer separating classes very well. Kill root and build a new subtree. Track alternative subtrees.
- ASHT: different size trees. Some small some large - the small ones adapt faster in response to detected change. When a tree reaches its maximum size, either reset or start with newest split as root.

Online methods - adaptive and fast

- How do we adapt/detect change?
- Track performance: warnings and triggers. We start saving data if we are warned of a change and once that change is established, kill off a member of the ensemble and use the saved data during the warning period to retrain a new member
- Track distribution in each node of the tree (ADWIN) which alerts if there is an indication that the stream (at each node) represents data drawn from two different means.

Online methods

- This is an active area of research.
- Lots of tuning: number of members in ensemble, base learner, how adaptive, how ”forgetfull”....
- Software documentation is not great...