Week 3 Review on Sampling and Mobile Computing

Samplers & Sensors

- Sampler
  - Cascaded Compression Sampling
  - Cover Tree
  - Generative Neural Sampler
- Sensor
  - Concept Drift
  - (coreset)

Cascaded Compression Sampling

- it makes use of low-rank of matrix to do matrix sketching with O($(c_1 + c_2)$(m+n)) time and space complexity
  - randomized algorithm with error bound
  - it uses two phase to capture key rows/columns from matrix
  - tries to solve low-rank decomposition of large data

Cascaded Compression Sampling

Algorithm 2: Cascaded Compression Sampling (CCS)

Input: $\mathbf{A}$; **Output**: $\mathbf{A} \approx \bar{\mathbf{U}}\bar{\mathbf{S}}\bar{\mathbf{V}}^\top$

1: *Pilot Sampling*: randomly select k columns and k rows $\mathbf{C} = \mathbf{A}_{[:,\mathcal{I}^c]}$, $\mathbf{R} = \mathbf{A}_{[\mathcal{I}^r,:]}$, $\mathbf{W} = \mathbf{A}_{[\mathcal{I}^r,\mathcal{I}^c]}$.

2: *Pilot approximation*: run $[\mathbf{U}, \mathbf{S}, \mathbf{V}] = \mathtt{sketching}(\mathbf{C}, \mathbf{R}, \mathbf{W})$, let $\mathbf{P} = \mathbf{U}\mathbf{S}^{\frac{1}{2}}$, and $\mathbf{Q} = \mathbf{V}\mathbf{S}^{\frac{1}{2}}$.

3: *Follow-up sampling*: perform weighted k-means on $\mathbf{P}$ and $\mathbf{Q}$, respectively, to obtain row index $\bar{\mathcal{I}}^r$ and column index $\bar{\mathcal{I}}^c$; let $\bar{\mathbf{C}} = \mathbf{A}_{[:,\bar{\mathcal{I}}^c]}$, $\bar{\mathbf{R}} = \mathbf{A}_{[\bar{\mathcal{I}}^r,:]}$, $\bar{\mathbf{W}} = \mathbf{A}_{[\bar{\mathcal{I}}^r,\bar{\mathcal{I}}^c]}$.

4: *Follow-up approximation*: $[\bar{\mathbf{U}}, \bar{\mathbf{S}}, \bar{\mathbf{V}}] = \mathtt{sketching}(\bar{\mathbf{C}}, \bar{\mathbf{R}}, \bar{\mathbf{W}})$.

P,Q:
compact, not necessarily accurate embedding of matrix

identify representative samples

sketch: calculate & normalize SVD

Canopy Sampling

- A fast lookup structure with adaptive rejection sampler
  - generating a sample is much faster than EM but with same accuracy
- Three parts
  - cover tree: hierarchical data structure that retrieves data in log time
  - adaptive sampler at top
  - rejection sampler at bottom
- Complexity

$$O\left(|S_{\hat{\imath}}| + c^6 \log n + c^6 \log m + c^4 e^{2^{\hat{\imath}+2}\|\tilde{\phi}(x)\|}\right)$$

c is sample expansion rate, m and n are number of cluster and number of data points

Algorithm

- Data Tree is consisted: for each data point x, it records ancestors(of above level) as prototype of this data point; the tree is built on sufficient statistic
- Algorithm:
  - pick a level: then it will have O(# of clusters) elements per node
  - perform alias sampler(sample n outcome at once)
  - then for each observation, sample by Metropolis-hasting sampling scheme
- http://proceedings.mlr.press/v70/zaheer17l

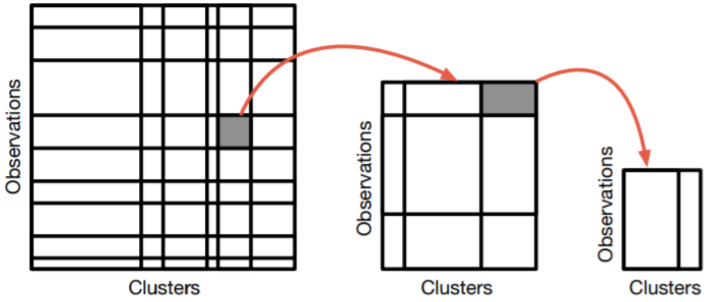


Figure 2. Hierarchical partitioning over both data observations and clusters. Once we sample clusters at a coarser level, we descend the hierarchy and sample at a finer level, until we have few number

Generative Neural Sampler

- they take a random input vector and produce a sample from a probability distribution defined by the network weights.
- Its training objective is all sets of divergence function, e.g. KL divergence
  - thus we can derive it to Bayesian Inference

Name	$D_f(P\|Q)$	Generator $f(u)$	$T^*(x)$
Kullback-Leibler	$\int p(x)\log\frac{p(x)}{q(x)}\,\mathrm{d}x$	$u\log u$	$1+\log\frac{p(x)}{q(x)}$
Reverse KL	$\int q(x)\log\frac{q(x)}{p(x)}\,\mathrm{d}x$	$-\log u$	$-\frac{q(x)}{p(x)}$
Pearson χ^2	$\int\frac{(q(x)-p(x))^2}{p(x)}\,\mathrm{d}x$	$(u-1)^2$	$2(\frac{p(x)}{q(x)}-1)$
Squared Hellinger	$\int\left(\sqrt{p(x)}-\sqrt{q(x)}\right)^2\,\mathrm{d}x$	$(\sqrt{u}-1)^2$	$(\sqrt{\frac{p(x)}{q(x)}}-1)\cdot\sqrt{\frac{q(x)}{p(x)}}$
Jensen-Shannon	$\frac{1}{2}\int p(x)\log\frac{2p(x)}{p(x)+q(x)}+q(x)\log\frac{2q(x)}{p(x)+q(x)}\,\mathrm{d}x$	$-(u+1)\log\frac{1+u}{2}+u\log u$	$\log\frac{2p(x)}{p(x)+q(x)}$
GAN	$\int p(x)\log\frac{2p(x)}{p(x)+q(x)}+q(x)\log\frac{2q(x)}{p(x)+q(x)}\,\mathrm{d}x-\log(4)$	$u\log u-(u+1)\log(u+1)$	$\log\frac{p(x)}{p(x)+q(x)}$

Fast Online Concept Drift Detection

- Concept Drift: the statistical properties of the target variable, which the model is trying to predict, change over time in unforeseen ways
  - most data stream assumes stable performance and all true labels are already classified
  - needs algorithm that detect concept drifts without true labels
- Incremental Kolmogorov-Smirnov(IKS)
  - given sample A and B, want to know if reject null-hypothesis; thus it can detect if two samples are from same distribution
- Scenario
  - online streaming

$$D \overset{?}{>} c(\alpha)\sqrt{\frac{n+m}{nm}}$$

Machine learning x Mobile

- A little search on “LTE” “mobile” “cellular” and machine learning in general
- mobile-friendly matrices and neural networks
- a little thoughts on end-to-end model vs combinations of models

Multitask federated learning

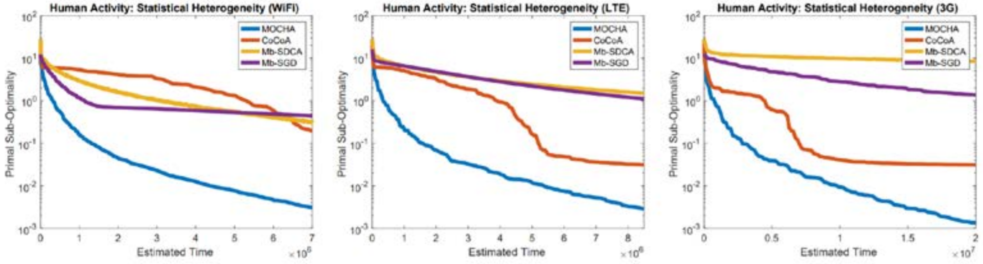
- The only paper among three years' NIPS and ICML mentioned "LTE"
- learn a shared prediction model while keeping all the training data on device
  - distributed and decentralized

for iterations $h = 0, 1, \cdots, H_i$ **do**
  for tasks $t \in \{1, 2, \ldots, m\}$ **in parallel over** m **nodes do**
    call local solver, returning θ_t^h-approximate solution $\Delta\boldsymbol{\alpha}_t$ of the local subproblem
    update local variables $\boldsymbol{\alpha}_t \leftarrow \boldsymbol{\alpha}_t + \Delta\boldsymbol{\alpha}_t$
    return updates $\Delta\mathbf{v}_t := \mathbf{X}_t\Delta\boldsymbol{\alpha}_t$
  reduce: $\mathbf{v}_t \leftarrow \mathbf{v}_t + \Delta\mathbf{v}_t$
Update $\boldsymbol{\Omega}$ centrally based on $\mathbf{w}(\boldsymbol{\alpha})$ for latest $\boldsymbol{\alpha}$

- Thus it needs to synchronize results by heterogeneous computing power, hardware and network condition

Dataset & Evaluation tools it uses

- Dataset (to do multi-task learning)
  - Google Glass
  - Human Activity Recognition(UCI dataset)
  - Vehicle Sensor
- Simulations on communication costs of LTE, 3G, Wi-Fi



- Machine learning community seems to lack a “plug-and-play” dataset.

TODO: resource management

Mobile-Friendly Matrices

- Small Foot-Print
- Structured Matrix Transformation
- A solution to our previous problem:
  what can we do about sparsity

Low-Rank Matrix

- decompose $M = GH^T$, where G and H are with both rank r
  - then we only need O(mr+rn) to store the matrix
- For example, in RNN

$$h_t^l = \sigma[W^l h_t^{l-1} + U^l h_{t-1}^l + b^l] \qquad l = 1, \ldots, L$$
$$h_t^l = \sigma[W_a^l W_b^l h_t^{l-1} + U_a^l U_b^l h_{t-1}^l + b^l]$$

- Can reduce parameter ~30%

Structural Matrix

- Non-trivial matrices has mn parameters and usually it takes O(mn) to perform a matrix-vector multiplication
- Taking advantage of Structural Matrix

(i) *Toeplitz*

$$\begin{bmatrix} t_0 & t_{-1} & \cdots & t_{-(n-1)} \\ t_1 & t_0 & \cdots & \vdots \\ \vdots & \vdots & \vdots & t_{-1} \\ t_{n-1} & \cdots & t_1 & t_0 \end{bmatrix}$$

Matrix Multiplication: O(nlogn)

(ii) *Vandermonde*

$$\begin{bmatrix} 1 & v_0 & \cdots & v_0^{n-1} \\ 1 & v_1 & \cdots & v_1^{n-1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & v_{n-1} & \cdots & v_{n-1}^{n-1} \end{bmatrix}$$

Matrix Multiplication: O(nlogn)

(iii) *Cauchy*

$$\begin{bmatrix} \frac{1}{u_0 - v_0} & \cdots & \cdots & \frac{1}{u_0 - v_{n-1}} \\ \frac{1}{u_1 - v_0} & \cdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \frac{1}{u_{n-1} - v_0} & \cdots & \cdots & \frac{1}{u_{n-1} - v_{n-1}} \end{bmatrix}$$

Matrix Multiplication: $O(nlog^2n)$

- A lot of matrices M (full rank, linear combination, inversed, etc. of structural matrices) can be written in a form of

$$\mathbf{M}(\mathbf{G}, \mathbf{H}) = \sum_{i=1}^{r} \mathbf{Z}_1(\mathbf{g}_i)\mathbf{Z}_{-1}(\mathbf{h}_i)$$

where $\mathbf{G} = [\mathbf{g}_1 \ldots \mathbf{g}_r], \mathbf{H} = [\mathbf{h}_1 \ldots \mathbf{h}_r] \in \mathbb{R}^{n \times r}$

and $Z_f(v) = \begin{bmatrix} v_0 & f v_{n-1} & \cdots & f v_1 \\ v_1 & v_0 & \cdots & f v_2 \\ \vdots & \vdots & \vdots & f v_{n-1} \\ v_{n-1} & \cdots & v_1 & v_0 \end{bmatrix}$

With this form of matrix, we can have a fast calculation on

- multiplication
- inversion
- Jacobian derivative

by Fast Fourier Transform

Application on RNN/LSTM

- Reduced 70% parameters with cost of 0.3 error rate(author's further paper)
  - but didn't mention implementation details
- Hybrid with Hardware Design(IBM paper)
  - with 512kb memory; going through 512*512 LSTM layer takes only 1.7μ second
  - choose first three layers(input, output, forgetting) of LSTM to transform to structural matrix training
    - as they are more robust/insensitive to error
    - keeping memory layer precise

Other applications in CNN

- Feature map in CNN
  - making use of convolution nature of structural matrices, one can compress the convolutional layers
  - http://proceedings.mlr.press/v70/wang17m/wang17m.pdf
- MEC
  - TODO: similar idea applies
  - https://arxiv.org/abs/1706.06873

Modules	Examples	Algorithms
Sensing	Detection of network anomalies or events by multiple-entry data from hybrid sources	Logistic Regression (LR) Support Vector Machine (SVM) Hidden Markov Model (HMM)
Mining	Classifying services according to the required provisioning mechanisms (e.g., band-width, error rate, latency)	Supervised learning: • Gradient Boosting Decision Tree (GBDT) Unsupervised learning: • Spectral Clustering • One-class SVM • Replicator Neural Networks (RNN)
Prediction	Forecasting the trend of UE mobility or the traffic volume of different services	Kalman Filtering (KL) Auto-Regressive Moving Average (ARMA) Auto-Regressive Integrated Moving Average (ARIMA) Deep Learning (DL): • Recurrent Neural Networks (RNN) • Long-Short Term Memory (LSTM) Compress Sensing (CS)
Reasoning	Configuration of a series of parameters to better adapt services.	Dynamic Programming (DP) • Branch-and-Bound Method • Primal-and-Dual Method Reinforcement Learning (RL) • Actor-critic Method • $\mathcal{Q}$-Learning Method Transfer Learning (TL)

maybe an end-to-end model cannot cover everything?

source:https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7886994