ML Project TODO

- Prepare proposal and think how to tackle the problems at hand
- Discuss the proposal with prof either in office hours or directly on piazza
- Start coding and working! Make 3 day progress reports!

This document will act as sort of a notes for my entire project from which I can very easily write the report.

Idea-1: The Chow-Liu algorithm scales quadratically with the number of attributes/variables. Develop an approximate version of the Chow-Liu algorithm so that it scales linearly with the number of attributes and examples. Use this algorithm to learn cutset networks and their mixtures and compare the performance with the conventional cutset networks learning algorithm both in terms of time and test-set log likelihood score.

Idea-2: Develop learning algorithms so that cutset networks can be used for solving multi-label classification problems. Use the IJCAI 2019 paper described above as a reference. Compare with the Mauro'16 paper (dcsn) for multi label approach.

Can also compare the performance of the methods in project to the simple methods discussed in class (these can act as the baselines).

Nov 5 office hours

Questions

- Discuss the approximation scheme for a fast approximation to CLtree learning. Main bottleneck is the quadratic number of mutual info pairs that need to be evaluated. Meil '99 proposes a method (under sparsity assumptions) to reduce this cost by leveraging the individual counts of each binary variable.
  - We can go about it from two ways: Either first compute a "close enough graph to the original clique, and then apply a mst algorithm on it..Or can use an approx algorithm for mst.
  - Compute the approx to mst on the fly using some greedy strategy so that it results in a spanning tree (which may not be maximal). This can also be seen as a regularization? As we are not closely depending on the exact and all mutual information pairs.
- Need the code for Cnet learning (c++/python) or can use the Python code used in Mauro'16 multilabel paper – available on github.
- For the multi-label problem, there is already a paper (Mauro'16) so is the suggestion to extend that? Any pointers towards it? Can combine it with the fast CLtree heuristic that will (hopefully) be developed? Can later even work on this combination? Ans: The paper does some heuristics but the IJCAI'19 paper shows a way to directly model the $P(Y|X)$ using the conditional cutset network from which we can just obtain a argmax for obtaining Y given an input X.

Discussion

For faster learning of CLtree it should ideally be always be worse than (sanity check) learning an exact CLtree since we will be applying heuristic methods to make it faster. We can use similar heuristics for

internal splitting of the nodes in the rooted OR tree to reduce its time complexity too. If the sanity check fails, itand checking for

Sampling based approach to reduce the d term in the overall time complexity since that can sometimes overpower the n^2 term. Only see from a subset of data?

Heuristic-1 to construct the spanning tree, select any node at random r and look at any k other nodes and estimate the counts needed for computing the mutual info. Then using some strategy extend the tree one node at a time, hence this algorithm will run for only n iterations (mst has n nodes). For each of the k nodes, get the count distribution i.e how many times was it 1 or 0, similarly for the current node. Compare amongst and select a node from the k-available nodes. For two variables $X1$ (dist of r) and $X2$ we are using $P(X1)$ and $P(X2)$ (comparing them) to see which edge to add to r.

- Thought 1: Instead of selection a node at random, why not select the one with highest count? According to meila paper, this may have some correlation with higher mutual information. ("Educated" guess)
- Do it randomly many times? and select the best?
- Focus on the base problem of approximation mutual information using just the count data of the variables.

Email/ask Prof. Daescu about this heuristic. Explain to him the problem setting and your proposed approach. What seems reasonable/not-so-reasonable to him and also any other suggestions? – **He was no help at all LOL!!!**

H2: Focus on D factor in N^2D term: Reduce it by data sampling? Create a perf graph for various ratios of sampling from the data: 0.1, 0.2, 0.3,....1.0 (full train data). Will need many runs to average out the LL results. (along with the run times too). Maybe use the full data to get the indiv entropies, and then when estimating the joints (bw 2 vars), use a reduced subset of the data.

H3: LB/UB for mutual information: Use some weak and naive lower bound for mutual information and learn the spanning tree using that? Learning a bottleneck spanning tree? Faster way described in CLRS – edge weight no more than a certain threshold. But HAVE to avoid the N^2 computation! Remember, approx methods are of no use if you still need to compute the full CLIQUE!! So perhaps the best bet is to go for "good" randomized methods?

Read the Meila paper once again. Run a profiler to see what parts of the code take most of the time? good idea and will lead to the direction where I should be focusing on?

Also, any way to do even more greedy procedure for computing the OR tree? Since that takes majority of the time. – Data sampling could again be used? Since the CLTree procedure is only being used on very "reduced" subset of the original training data. Also rather than computing the gain for all N variables, just compute it for a selected k variables and select the maximum gain from there?

Code repositories

- mauro dcsn (python)
- shashaJ cnxd (python)
- Email prof for obtaining the C++ code??

Datasets

Use the homework-4 datasets – small enough for a project but representative enough. Will also have other baselines to compare against the proposed model. Can also try to model using some other probabilistic technique (for comparing the log likelihoods).

All of them are a subset of the datasets used in the paper, so it seems like a fair comparison. So this part is done! – Can always try out other schemes later on.

Evaluation

Write down the machine specs, and the code type and packages used. Use a numpy random seed for reproducible results.

1. Test set average log likelihoods (i.e per data point)
2. Running time of the algorithms – for training + pruning (if any) – use recommended python modules

Baselines and Experiments

Simple ones: Indep Bayesian networks, Chow Liu Trees, Forest CLtree mixtures (simple and fast to compute) Compare the approximate scheme by just running it for a CLtree with a normal CLtree i.e not using the cutset network – not a priority since the focus should be on improving Cnets.

Normal Cutset networks and CNets with Post pruning as described in the paper. Cnets + CnetsP using the approximation scheme – Plus if possible, approximate versions of CLtrees. Existing results from the paper (or Re-run the experiments again). Mixture based results take a long time to run, so need to keep that in mind – introduce convergence criteria or ignore mixtures for now.

Little bit slower: Cutset networks, EM CLtree mixtures, EM Cutset mixtures (but our main goal is to compare the performance of the heuristic against a single cutset network - - so NOT trying out many mixture models is okay. If trying out mixtures, check for convergence criteria. Not using them would be a better idea since they have built in randomization – will then need to average out the results by reporting the mean and std values.

- Indep Bnets, CLtrees, approx-CLtrees.
- Cnets, CnetsP i.e no approx.
- app-Cnet + true-CLtree
- app-Cnet + app-CLtree
- true-Cnet + app-CLtree
- Mixtures (if time permits)? – or hard EM threshold and convergence check

Others:

- Exploratory data analysis for all the data sets – maybe important for conclusions etc.. Comparing the distributions of train, val data sets (test data only for final reporting).
- Graphs for training time vs accuracy? Logging the run time of the experiments.
- Track the Eval LL across various time intervals for all exps – plot it out!
- Time take to evaluate the results too (computing log-likelihoods, probability of a single input data point). Store the trained model files as pickle objects to do this analysis later on too.

- Some inference run time comparison??? Since the approximate models should in effect be a bit more “compressed”.

Code

Folders: docs, data, src, results, notebooks. Notebooks for initial data exploration and stats code Results to save the eval metrics in a txt file. Can also store the trained models here in pickled format here. Document the functions used properly! Reference code in data folder.

Running time: Run code in github measure total time for main script whereas we will measure only train time inside the code.

Figures and Tables:

- Figures Eval LL vs Run time during traing for various exps and datasets. – only true for EM-stye algos where params are updated in an iteration stage.
- Run time for single data point eval / test LL various exps on same data
- Table: Run time for all exps across all data
- Table: Test LL for all exps across all data
- Fig: Run time vs dataset size ?? can include diff exps too in the figure.
- Fig: Test LL vs dataset size ??
- Runtime vs performance trade off analysis
- Doing a comparison/analysis of D (num of data points) and N (num of featues) since each affect the runtimes of the training algorithms.

Methods

- 1 for CLtree,
- 1 for ORtree
- One will simply be their hybrid
- Good random approx method (very aggressive?) – random edge selection!! – then compute MST on it.

Try to think about as many ideas possible – good and motivated approximations schemes. Need to show a good motivation for an proposed method – should not be any random scheme? Or is it??? Compare with some random approx method too. (good baseline?)

Heuristic-1 to construct the spanning tree, select any node at random r and look at any k other nodes and estimate the counts needed for computing the mutual info. Then using some strategy extend the tree one node at a time, hence this algorithm will run for only n iterations (mst has n nodes). For each of the k nodes, get the count distribution i.e how many times was it 1 or 0, similarly for the current node. Compare amongst and select a node from the k-available nodes. For two variables $X1$ (dist of r) and $X2$ we are using $P(X1)$ and $P(X2)$ (comparing them) to see which edge to add to r.

Thought 1: Instead of selection a node at random, why not select the one with highest count? According to meila paper, this may have some correlation with higher mutual information. (“Educated” guess)

H2: Focus on D factor in N^2D term: Reduce it by data sampling? Create a perf graph for various ratios of sampling from the data: 0.1, 0.2, 0.3,....1.0 (full train data). Will need many runs to average out the LL results. (along with the run times too). Maybe use the full data to get the indiv entropies, and then when estimating the joints (bw 2 vars), use a reduced subset of the data.

Also, any way to do even more greedy procedure for computing the OR tree? Since that takes majority of the time. – Data sampling could again be used? Since the CLTree procedure is only being used on very "reduced" subset of the original training data. Also rather than computing the gain for all N variables, just compute it for a selected k variables and select the maximum gain from there?

Run a profiler to see what parts of the code take most of the time? good idea and will lead to the direction where I should be focusing on? Also good to show anaylysis – separate the training code into methods for profiling like: `create_graph`, `compute_mst` etc.

Finalized

Randomized edge sampling: Select K pairs for MI (K > N) and construct MST from them. MI pairs: O(KD), MST: O(K log N) – guaranteed connectivity not possible?. Introduce this as a motivation but say tried out other methods.

1. Approx SpanTree (AST): Select node r (randomly or using some heuristic), then select K different nodes for it and add edge between r and best scoring node. (edge weight from full data!) For the next node, select any node randomly from the current tree, (motivate this, if dont do this we will get a linear chain instead of a tree. Also the new K nodes at each step should avoid "back" edges to uphold the tree structure.) Select the K nodes out of the remaining set of vertices .. until we have V-1 edges. doing this procedure N times to directly get a spanning tree. However while setting the structure, compute the pairs and sing probs for the edges addded to the graph. **Hyper parameter**: K Selecting best:
    1. Either compute K MI pairs for best scoring node
    2. Compare the K marginal counts and select the best scoring one!
2. Approx Graph (AGH): Construct graph with NK edges and then compute MST from it. $K \sim logN$, to get a "dense" tree like structure using the above procdeure instead of throwing away the edges. Set edge weights to the MI info and compute the MST from it. While computing the graph, set the probs sing and pairs for later use to predict. Be careful here! **Hyper parameter**: K
3. Approx OR tree (ODS): Only seeing a subset of dataset to select the best attribute. **Hyper parameter**: fraction f of data to sample. f can be like 0.1, 0.2, ... , 1.0. (1.0 should be included for sure!)

So in effect there are two hyper params: f, K. Setting values for f is easy but K depends. Both M2 and M3 will ensure that there are no isolated nodes. Obviously, higher K should result in a better performance: $1 \leq K \leq N$.

- K can be chosen at run time as $K = \log_2 N$
- K can be chosen via hyper-parameter tuning over the Validation set. Select a decent list for K values then.

All todos are below: Add sections above as needed.

Todo Nov 30

- **done** Getting the entire pipeline ready: data loading, exp metrics, storing them, timing parts of code, saving trained models, writing results to text file etc.... **done**
  - **done** Currently storing: Train time, Trn,Val,Tst LL scores, clf object (large size) for later analysis in the dict –

- Need to average out the Running time by doing multiple runs. Report the mean and std of the runtimes.
  - Integrate the following in the main code with the use of a helper function.
- **done** Getting indep, cltree baselines working on at least some data sets. **done**
- **done** Getting cnet, cnetP baselines ready – compare sasha code to yours and do a re-write. **done**
  - Pruning Cnet (cnetP) class remains to be implemented (later, while code is running!)
- Getting experiment design ready == **approximation methods** + theory/motivation behind them!
- Documenting code, Writing the report, Doing analysis with experiments, exploration notebooks.