GGBoost - Graph Gradient Boosting

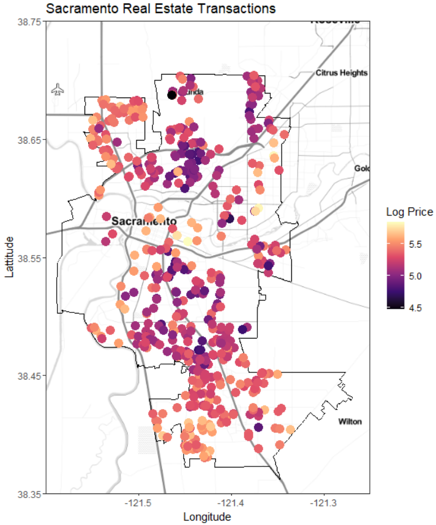
Isaac Ray, Huiyan Sang

Section 1

Motivation

Data

- Response: House prices in Sacramento
- Covariates: **Longitude, Latitude**, sq.ft., # beds, # baths, etc.
- We know that longitude and latitude are highly related!
- Treating them independently is a bad modeling assumption
- How can we predict unobserved housing prices using these covariates?



More generally:

- Observed data $\mathcal{D} = \{d_i\}_{i=1}^{n}; \quad d_i = (\mathbf{x}_i, y_i)$ from random variables $(\mathbf{X}, Y)$
- $\mathbf{X} = \{X_1, \ldots, X_p\} \in \mathcal{X}$; p dimensional 'feature' vector lying in feature space
- $Y \in \mathbb{R}$; real-valued response (for regression)
- There is an unknown function $\phi : \mathcal{X} \rightarrow \mathbb{R}$ relating $\mathbf{X}$ and Y
- **We may know something about $\mathcal{X}$ or the relationships between our X_j's**

Decision Tree Ensembles

- A popular approach to estimate ϕ is using an *ensemble* model
- We choose to model $\hat{\phi}(\mathbf{x}) = \sum_{k=1}^{K} f_k(\mathbf{x})$; f_k is called a *weak learner function*
- The form of our weak learner function is a *decision tree* $\mathcal{T}$
  - Decision trees have a recursive structure; they consist of nodes ζ which define the decision tree structure
  - A decision node η consists of a predicate $P(\mathbf{x}) : \mathcal{X} \to \{0, 1\}$ and 2 child nodes $\{\zeta_t, \zeta_f\}$
  - The value of a decision node is $\eta(\mathbf{x}) = P(\mathbf{x})\zeta_t(\mathbf{x}) + (1 - P(\mathbf{x}))\zeta_f(\mathbf{x})$
  - A leaf node ξ takes a fixed scalar value called its *leaf weight*
- We use our $\hat{\phi}(\mathbf{x}_i) = \hat{y}_i$ to do prediction

Gradient Boosting

- *Gradient Boosting* is a technique for constructing our f_k's in an iterative fashion
- Given some loss function $\ell(y_i, \hat{y}_i)$, we use its gradient to inform our update of f_k then shrink its contribution (learning rate)
- Instead of updating all K weak learners at once, updates are done conditioned on all other weak learners (residual fitting)
- Updating a decision tree weak learner consists of changing a leaf node into a decision node
- **Essentially all existing GBDT models use a predicate of the form $X_j > c$ for a single feature X_j and constant c**

XGBoost

- eXtreme Gradient Boosting uses both first and second order gradient information; at iteration t we have

$$g_i^{(t)} = \partial_{\hat{y}^{(t-1)}} \ell(y_i, \hat{y}^{(t-1)}); \quad h_i^{(t)} = \partial^2_{\hat{y}^{(t-1)}} \ell(y_i, \hat{y}^{(t-1)})$$

- Looks scary; but easy and fast to compute (for example, mean squared error)
- Express our Ω penalized objective function as second-order Taylor expansion:

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} \left(\ell(y_i, \hat{y}_i^{(t-1)}) + g_i^{(t)} f_k(\mathbf{x}_i) + \frac{1}{2} h_i^{(t)} f_k^2(\mathbf{x}_i) \right) + \Omega(f_k)$$

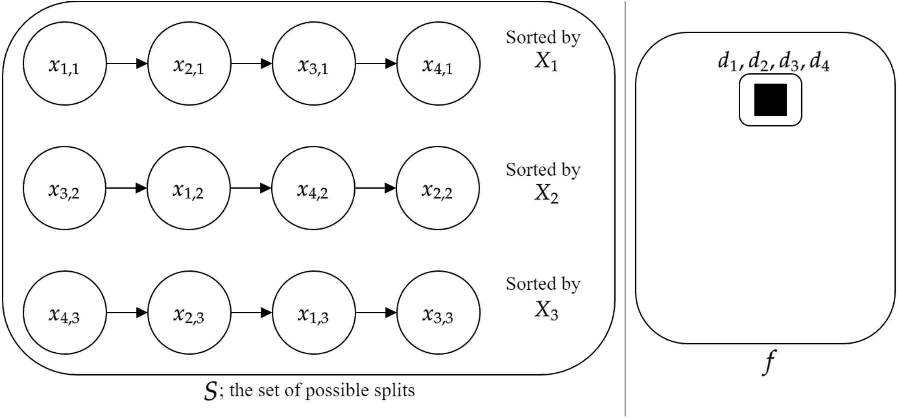
- By sorting each feature, can do a linear scan over observations to greedily choose the best predicate for the new decision node; equivalent to maximizing:

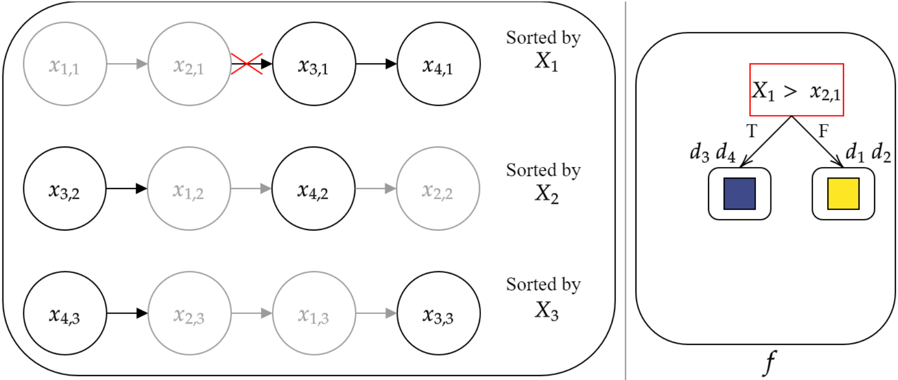
$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G_I^2}{H_I+\lambda}\right] - \gamma$$

where $G_A = \sum_{i \in A} g_i$ and $H_A = \sum_{i \in A} h_i$ for

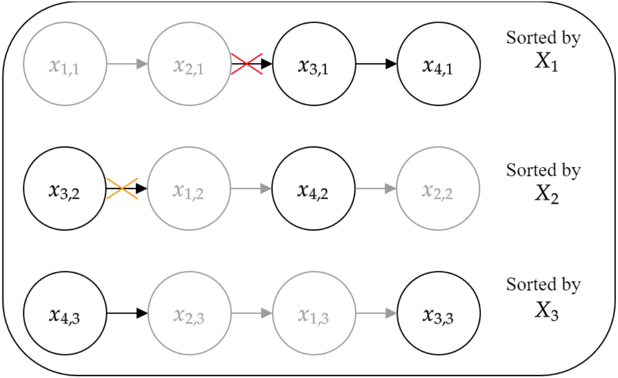
- $A = I$ (current data in leaf node),
- $A = L$ (data that would satisfy new predicate), and
- $A = R$ (data that would not satisfy new predicate)

Consider as an example, $\mathcal{D} = \{d_i\}_{i=1}^{4}$ where $d_i = (\{x_{i,1}, x_{i,2}, x_{i,3}\}, y_i)$

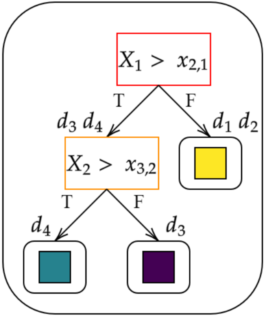




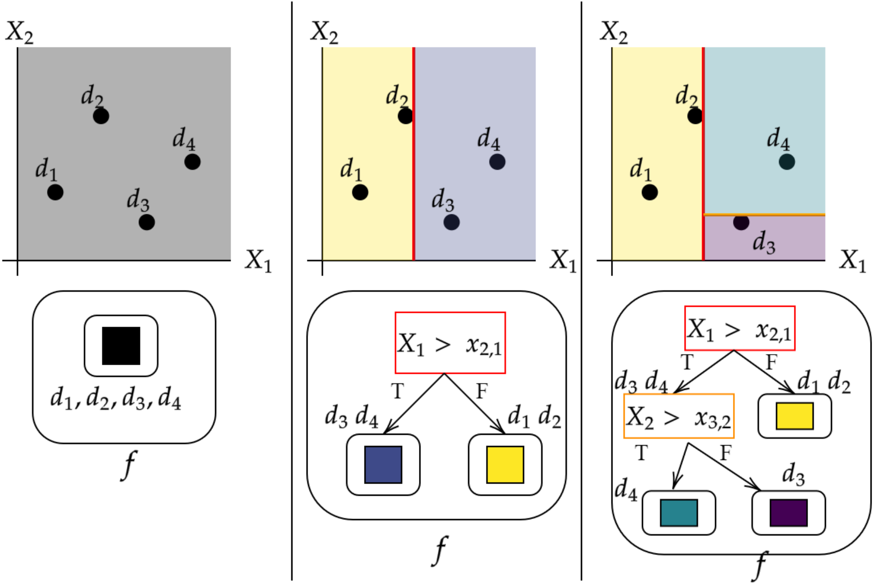
S; the set of possible splits (conditioned on)



S; the set of possible splits (conditioned on ■)



f


X2
X1
d1
d2
d3
d4
d1, d2, d3, d4
f
X1 > x2,1
T
F
d3 d4
d1 d2
X2 > x3,2

Section 2

Graph-Split Decisions

Candidate Graph

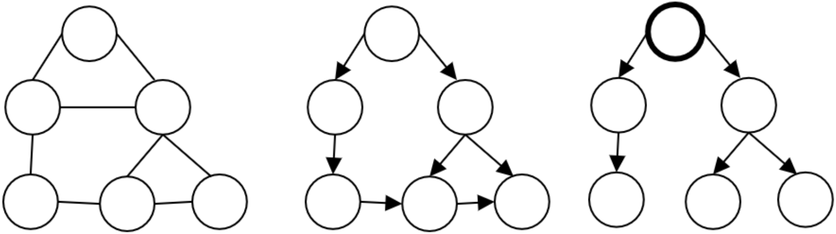
- Before we can propose our graph-split-based predicate, we have to describe our graphs
- We assume that for each weak learner f_k, there is a set of candidate graphs $\mathbb{G}_k$; can vary across weak learners to enhance diversity
- A *candidate graph* $\overrightarrow{\Gamma} \in \mathbb{G}$ is a graph with *arborescence structure*

Definition

An arborescence denoted $\overrightarrow{\Gamma} = \{\mathcal{V}, \overrightarrow{\mathcal{E}}\}$ is a directed tree that connects all the vertices in a graph, and has a designated root vertex from which all other vertices are reachable through directed edges

Why arborescences?

- The choice of restricting candidate graphs to arborescences may seem to come out of nowhere
- We do it because of two essential properties:
    - *Order*: By starting at the root vertex, we can define an ordering on the graph to do a greedy search (parallels the linear scan for XGBoost!)
    - *Split-Separable*: Removing any edge from an arborescence results in two sub-graphs which are also arborescences (parallels the L, R predicate sets for XGBoost!)

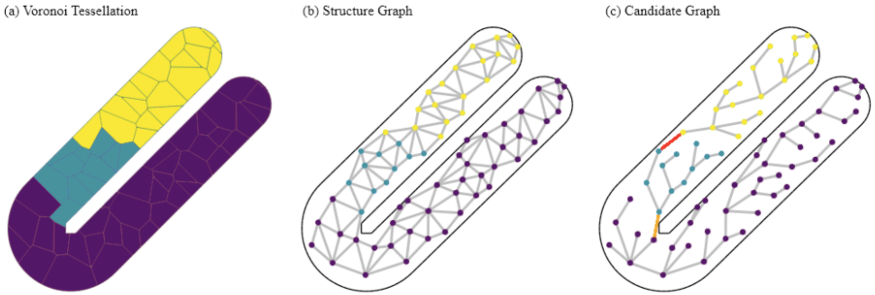


Binning

- What do graphs have to do with our data? Suppose for now that we already have a candidate graph $\overrightarrow{\Gamma}_q$
- We assume there is a known function $Z_q : \mathcal{X} \to \mathcal{V}(\overrightarrow{\Gamma}_q)$ called the *binning function* that maps the feature data to each vertex in $\overrightarrow{\Gamma}_q$
- **We allow Z_q to be a function of multiple features in $\mathbf{X}$ so that a candidate split of $\overrightarrow{\Gamma}_q$ can depend on multiple features**
- $Z_q(\mathbf{x})$ may be defined in a way such that multiple observations are assigned to a single vertex; vertices can be considered bins of data

Building Candidate Graphs

- How do we actually choose a Z_q and $\overrightarrow{\Gamma}_q$?? Usually, we define them simultaneously
  - For univariate data, we can construct $\overrightarrow{\Gamma}_q$ to be a *chain graph* and Z_q to be an ordering function (or ordering + histogram binning, such as LightGBM)
  - For data on a known manifold, we can sample random tessellations to get neighbor graphs + binning, then sample candidate graphs
  - For data with an existing graphical structure, we can sample candidate graphs directly or collapse the graph with a binning function
- This is a flexible framework with lots of unexplored potential! In the most general setting a candidate graph edge is simply a hypothesis about a relationship in the data


(a) Voronoi Tessellation
(b) Structure Graph
(c) Candidate Graph

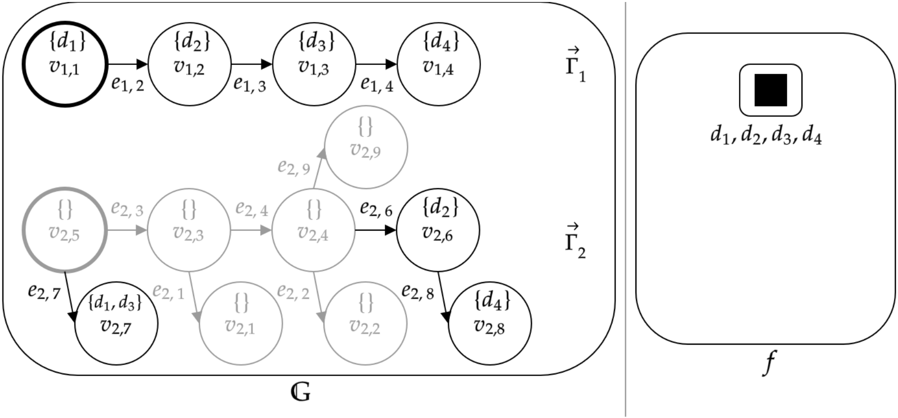
Graph Split Decision Rule

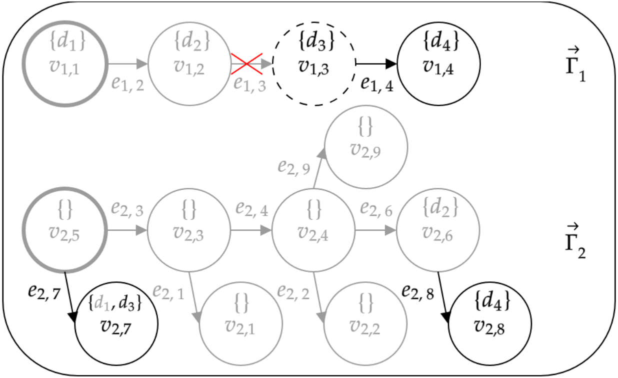
- Now that we've described our candidate graph and binning function, we can define our graph-split predicate for our decision tree

Definition

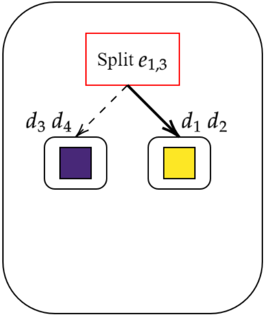
A *graph-split* decision rule is a predicate $P_{q,j}$ uniquely identified by an edge $e_{q,j} \in \overrightarrow{\mathcal{E}}(\overrightarrow{\Gamma}_q)$. For any vertex $v \in \overrightarrow{\Gamma}_q$, $P_{q,j}(v)$ is true iff removing edge $e_{q,j}$ from $\overrightarrow{\Gamma}_q$ causes v to belong to the sub-arborescence rooted at v_j. We denote this as *Split* $e_{q,j}$

- Using this predicate and our known Z, we get our L, R data sets and can evaluate a split using the same *Gain* formula as XGBoost!

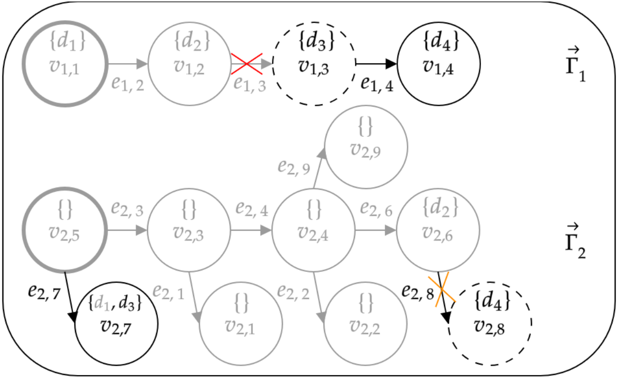

{d_1} v_{1,1}
e_{1,2}
{d_2} v_{1,2}
e_{1,3}
{d_3} v_{1,3}
e_{1,4}
{d_4} v_{1,4}
Γ_1
{} v_{2,9}
e_{2,9}
{} v_{2,5}
e_{2,3}
{} v_{2,3}
e_{2,4}
{} v_{2,4}
e_{2,6}
{d_2} v_{2,6}
Γ_2
e_{2,7}
{d_1, d_3} v_{2,7}
e_{2,1}
{} v_{2,1}
e_{2,2}
{} v_{2,2}
e_{2,8}
{d_4} v_{2,8}
G
d_1, d_2, d_3, d_4
f



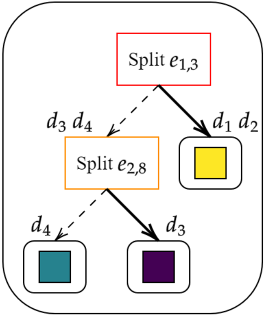
$\mathbb{G}$; conditioned on ■



f



$\mathbb{G}$; conditioned on ■



f

Section 3

Experiments

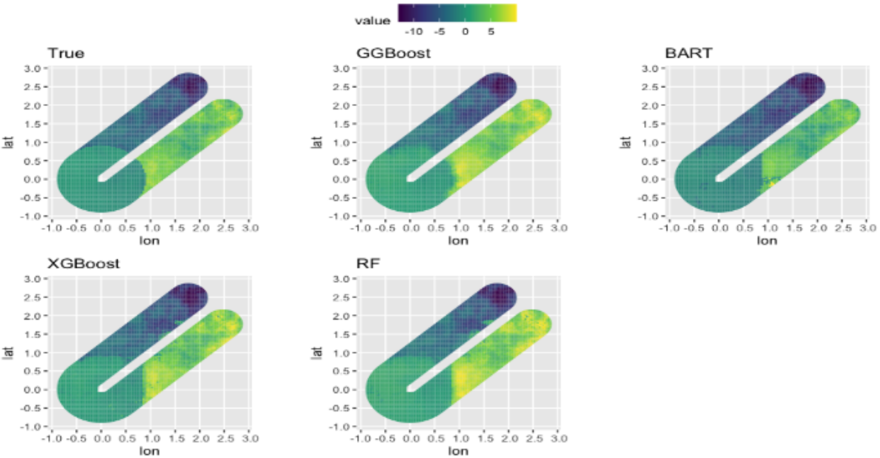
Competing Methods

- GGBoost is implemented in C++ with OpenMP parallelism and an R interface
- We compared against XGBoost and Random Forest (two most popular ensemble decision tree models)
- Also compared against BART because we are fans of Ed George
- We investigated some other approaches like oblique tree ensembles but encountered lots of implementation issues
- Used the `caret` package to do model tuning via cross-validation

Synthetic Data

- Regression tasks with an R (loss is testing mean square error), classification with a C (loss is testing accuracy)

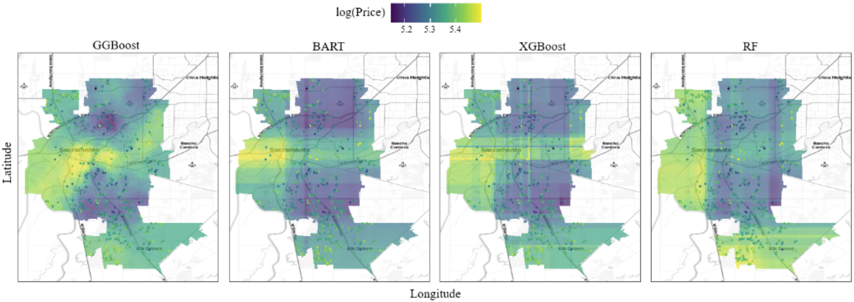
	GGBoost	**XGBoost**	**RF**	**BART**
U-shape$_R$	**0.57**	1.20	1.08	1.29
Torus$_R$	**2.16**	3.28	3.89	4.09
U-shape$_C$	**90.7**	90.3	90.5	89.8
Torus$_C$	**83.2**	80.2	82.3	78.3


value
-10
-5
0
5
True
GGBoost
BART
XGBoost
RF
lat
lon
3.0
2.5
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.0 -0.5 0.0 0.5 1.0 1.5 2.0 2.5 3.0

Real Data

- Includes both spatial data with known boundaries (GeoDa Data & Lab) and graphical data (citation networks)

	GGBoost	**XGBoost**	**RF**	**BART**
Sacramento Homes$_R$	**1.79**	2.10	1.87	1.91
NYC Education$_R$	**0.52**	0.59	0.57	0.68
King County Homes$_R$	**2.58**	2.96	3.26	2.83
Las Rosas Crops$_R$	2.34	2.57	**2.22**	2.39
US Election$_R$	**0.61**	0.68	0.84	0.70
Cora Citations$_C$	**83.8**	76.4	75.1	70.1
PubMed Citations$_C$	90.3	**90.7**	89.0	88.8


log(Price)
5.2
5.3
5.4
GGBoost
BART
XGBoost
RF
Latitude
Longitude

Section 4

Conclusion

GGBoost is neat!

- GGBoost is a highly flexible extension of standard GBDT models, with clear applications to problems with well understood relationships such as spatial covariates
- It maintains the benefits of normal GBDTs (fast, scalable, easily deployed, importance scores, etc) while allowing for more expressive decision trees
- Lots of future directions to investigate
  - Using manifold estimation techniques such as UMAP (estimates feature relationships as a weighted graph already!)
  - Different loss functions / data augmentation schemes for new tasks (density/intensity estimation with logistic approximation)
  - How to incorporate time series data
  - Bagging approach instead of boosting (straightforward, we just haven't tried yet)

Thank you! Questions?