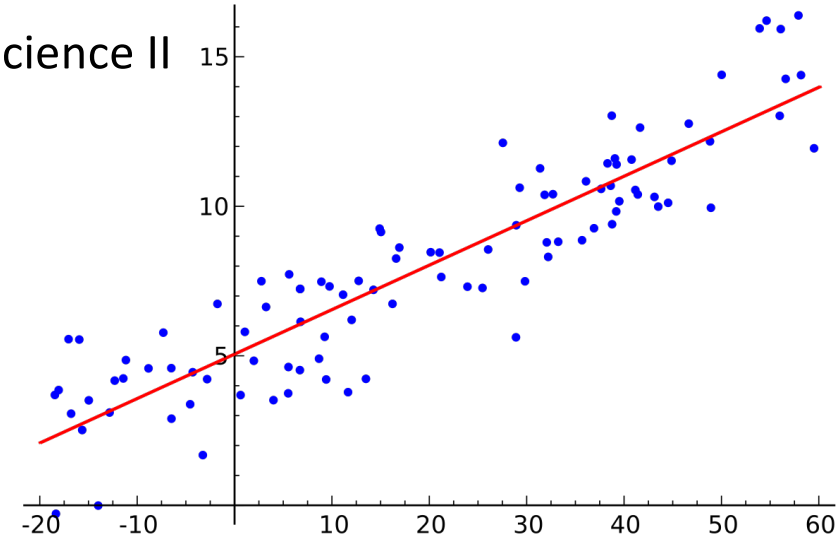
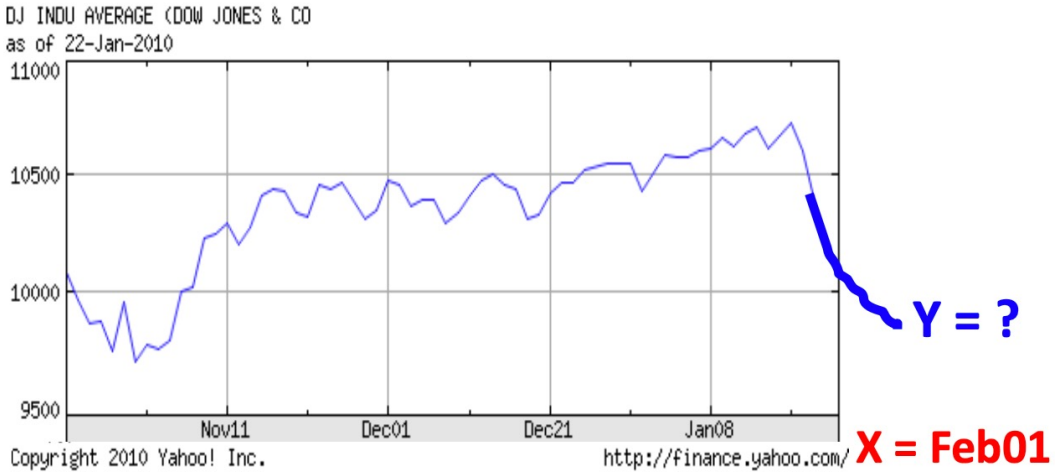
Linear and Ensemble Methods

CSCI 4360/6360 Data Science II



In general, the goal:

- Construct a **predictor** f: X -> Y that minimizes risk R(f)



Classification:

$$R(f) = P(f(X) \neq Y)$$

Probability of Error

Regression:

$$R(f) = \mathbb{E}[(f(X) - Y)^2]$$

Mean Squared Error

Optimal [regression] predictor

$$f^* = \arg\min_f \mathbb{E}[(f(X) - Y)^2]$$

$$= \mathbb{E}[Y|X]$$ (Conditional Mean)

We can prove this!

$$f^* = \arg\min_f \mathbb{E}[(f(X) - Y)^2] \; = \mathbb{E}[Y|X]$$

- Therefore, for any prediction rule f $R(f) \geq R(f^*)$
- We define R(f) $R(f) = \mathbb{E}_{XY}[(f(X) - Y)^2]$
- Expand according to axiom of conditional probability

$$= \; \mathbb{E}_X[\mathbb{E}_{Y|X}[(f(X) - Y)^2|X]]$$

It escalates quickly..

$$= \mathbb{E}_X[\mathbb{E}_{Y|X}[(f(X) - Y)^2|X]]$$

- (dropping subscripts for convenience)

$$= E\left[E\left[(f(X) - E[Y|X] + E[Y|X] - Y)^2|X\right]\right]$$

- Multiplying out the quadratic term

$$\begin{aligned} = E[& E[(f(X) - E[Y|X])^2|X] \\ & +2E\left[(f(X) - E[Y|X])(E[Y|X] - Y)|X\right] \\ & +E[(E[Y|X] - Y)^2|X]] \end{aligned}$$

$$= E\left[(f(X) - E[Y|X])^2\right] + R(f^*)$$

Finally…

$$= \underbrace{E\left[(f(X) - E[Y|X])^2\right]}_{\geq 0} + R(f^*)$$

Intuition: Signal plus (zero-mean) Noise model

$$Y = f^*(X) + \epsilon$$

Depends on **unknown** distribution P_{XY}

Empirical Risk Minimization (EMR)

- Since we don't have direct access to the unknown distribution P(X, Y), we rely on statistics

Optimal predictor:

$$f^* = \arg\min_f \mathbb{E}[(f(X) - Y)^2]$$

Empirical Risk Minimizer:

$$\hat{f}_n = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} (f(X_i) - Y_i)^2$$

Class of predictors **Empirical mean**

$$\frac{1}{n} \sum_{i=1}^{n} [\text{loss}(Y_i, f(X_i))] \xrightarrow[\text{Numbers}]{\text{Law of Large}} \mathbb{E}_{XY} [\text{loss}(Y, f(X))]$$

Part I: Linear Models

Linear Models

- Means each f in F_L is restricted to a class of *linear* functions (in X)

$$\widehat{f}_n^L = \arg\min_{f \in \mathcal{F}_L} \frac{1}{n} \sum_{i=1}^{n} (f(X_i) - Y_i)^2$$

Linear Models

0 response submitted

Which of these is a linear model?

Gaussian Process

Decision Tree

Support Vector Machine with RBF kernel

Logistic Regression

POE TAY TOES! Boil 'em, mash 'em, stick 'em in a stew

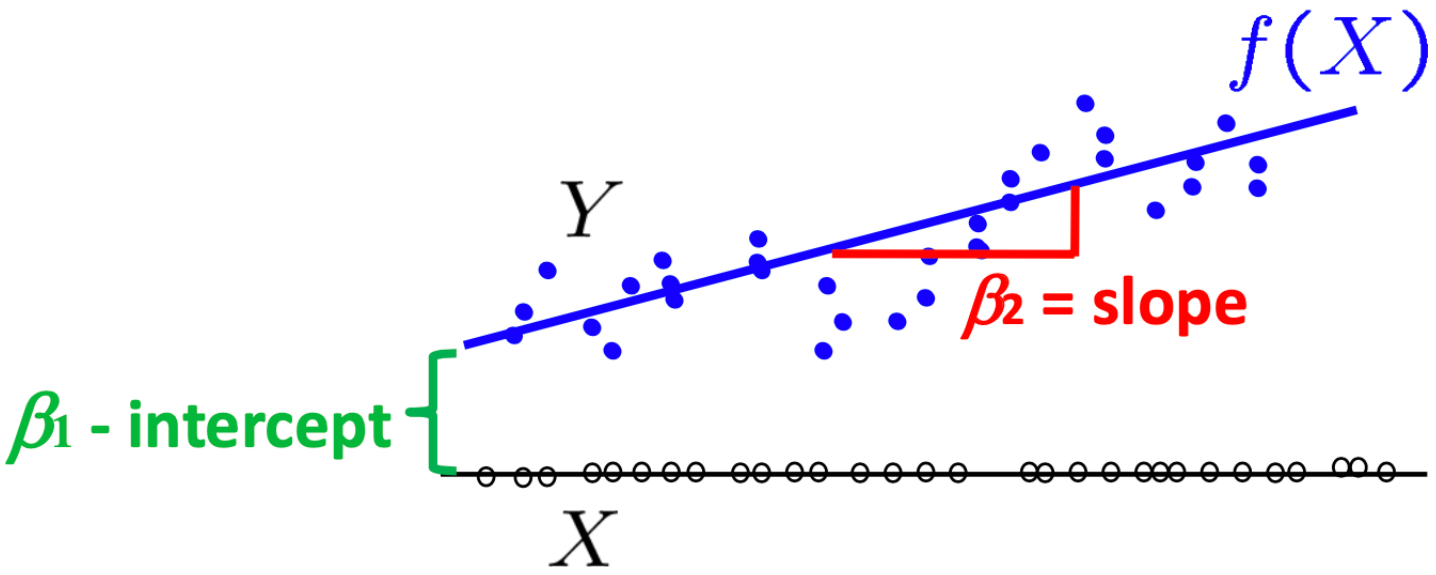
Treemap Bar

1 of 1

Univariate case

- You've seen this a million times…

$$f(X) = \beta_1 + \beta_2 X$$



Multivariate case

$$f(X) = f(X^{(1)}, ..., X^{(p)}) = \beta_1 X^{(1)} + \beta_2 X^{(2)} + ... + \beta_p X^{(p)}$$

$$= X\beta$$

- where

$$X = [X^{(1)}, ..., X^{(p)}], \beta = [\beta_1, ..., \beta_p]^T$$

Finding the estimator

- Back to our original equation $\widehat{f}_n^L = \arg\min_{f \in \mathcal{F}_L} \frac{1}{n} \sum_{i=1}^{n} (f(X_i) - Y_i)^2$
- Substitute in our line parameter $\widehat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} (X_i\beta - Y_i)^2$
- Expand the square

$$= \arg\min_{\beta} \frac{1}{n} (\mathbf{A}\beta - \mathbf{Y})^T (\mathbf{A}\beta - \mathbf{Y})$$

where $\mathbf{A} = \begin{bmatrix} X_1 \\ \vdots \\ X_n \end{bmatrix} = \begin{bmatrix} X_1^{(1)} & \dots & X_1^{(p)} \\ \vdots & \ddots & \vdots \\ X_n^{(1)} & \dots & X_n^{(p)} \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} \mathbf{Y}_1 \\ \vdots \\ \mathbf{Y}_n \end{bmatrix}$

Finding the estimator

$$\widehat{\beta} = \arg\min_{\beta} \frac{1}{n}(\mathbf{A}\beta - \mathbf{Y})^T(\mathbf{A}\beta - \mathbf{Y})$$

$$J(\beta) = (\mathbf{A}\beta - \mathbf{Y})^T(\mathbf{A}\beta - \mathbf{Y})$$

$$= \arg\min_{\beta} J(\beta)$$

Normal Equations

$$(\mathbf{A}^T\mathbf{A})\widehat{\beta} = \mathbf{A}^T\mathbf{Y}$$

p xp p x1 p x1

- If (A^TA) is invertible, this is easily solved!

$$\widehat{\beta} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{Y}$$

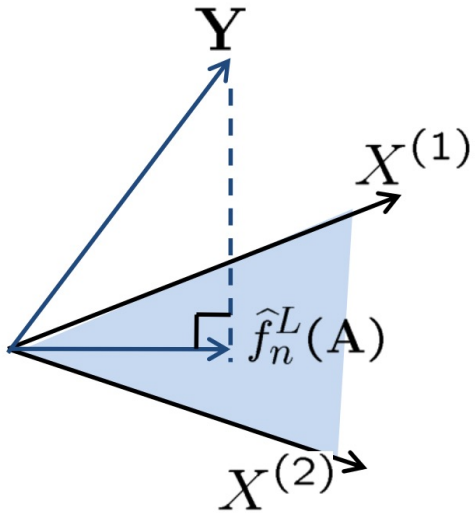
- **When is (A^TA) invertible?**
- When A is full rank (recall Linear Algebra Review lectures)

Geometric Interpretation

$$\widehat{f}_n^L(X) = X\widehat{\beta} = X(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{Y}$$

- $f_n(A)$ is the orthogonal projection of Y onto the linear subspace spanned by the columns of A (aka X)

- (go back and review the Linear Algebra notes!)



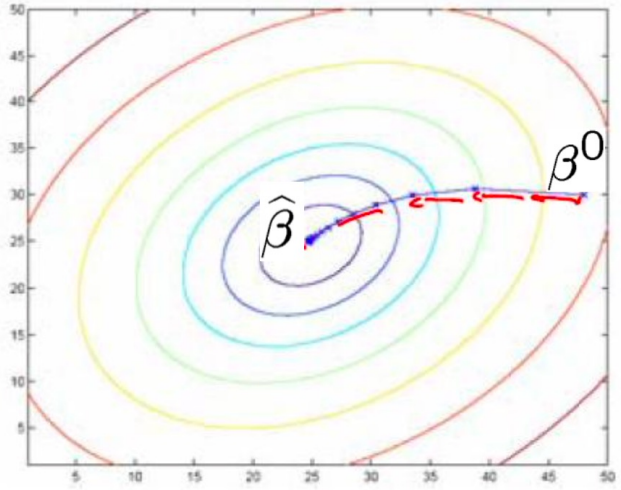
Normal Equations

- We saw how to solve when (A^TA) is invertible
  - Unspoken was the assumption that it is also *small*
- What about when (A^TA) is invertible, but **very large**?
- **Gradient descent!** J(β) is convex!

Initialize: β^0

Update: $$\beta^{t+1} = \beta^t - \frac{\alpha}{2}\frac{\partial J(\beta)}{\partial \beta}\Big|_t$$

$$= \beta^t - \alpha\, \mathbf{A}^T(\underbrace{\mathbf{A}\beta^t - Y}_{0 \text{ if } \beta^t = \widehat{\beta}})$$



Normal Equations

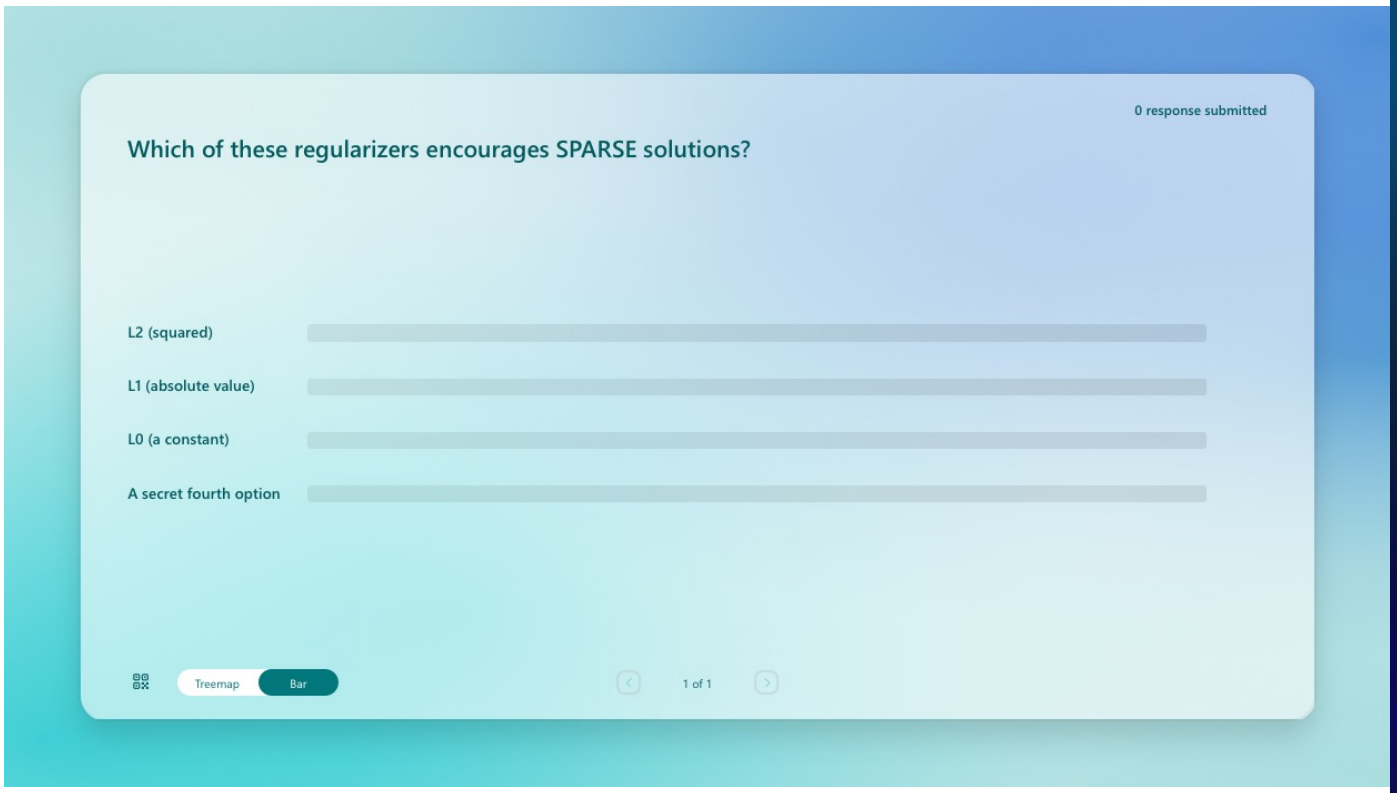
- What about when (A^TA) is **not** invertible?
- **Prior + regularization**

$$\widehat{\beta}_{\text{MAP}} = \arg\max_{\beta} \underbrace{\log p(\{(X_i, Y_i)\}_{i=1}^{n} | \beta, \sigma^2)}_{\text{log likelihood}} + \underbrace{\log p(\beta)}_{\text{log prior}}$$

Non-invertible case

- What kinds of priors?
- Gaussian prior $$\widehat{\beta}_{\text{MAP}} = \arg\min_{\beta} \sum_{i=1}^{n} (Y_i - X_i\beta)^2 + \lambda \|\beta\|_2^2$$
- Laplace prior $$\widehat{\beta}_{\text{MAP}} = \arg\min_{\beta} \sum_{i=1}^{n} (Y_i - X_i\beta)^2 + \lambda \|\beta\|_1$$
- **What's the difference?**

L7 - Regularizers

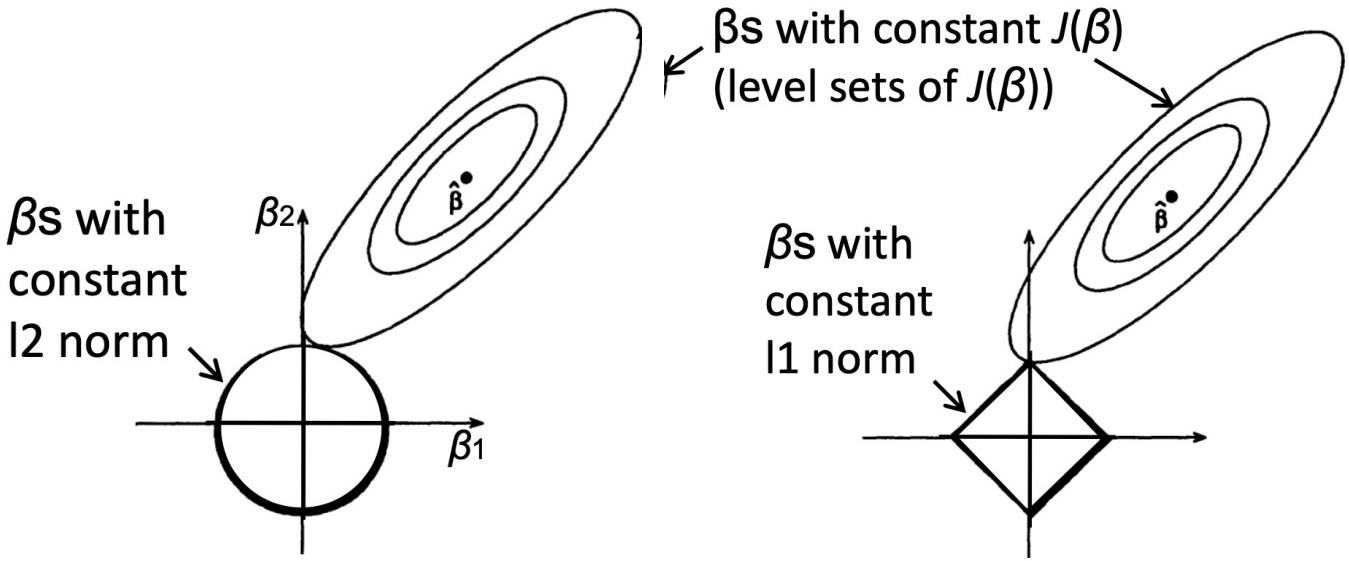


Types of regularization

$$\min_{\beta}(\mathbf{A}\beta - \mathbf{Y})^T(\mathbf{A}\beta - \mathbf{Y}) + \lambda \text{pen}(\beta)$$

Ridge regression (L2)

Lasso regression (L1)



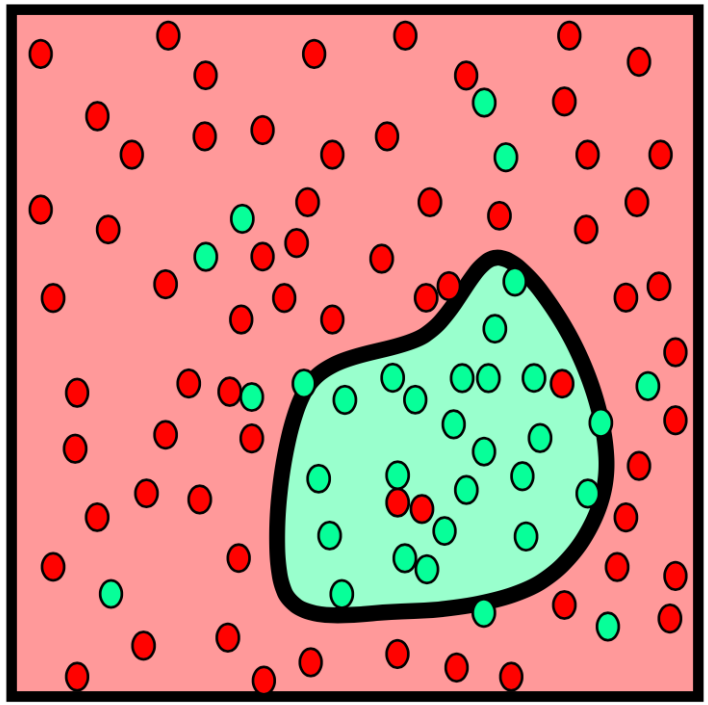
L1 (Lasso) results in *sparse* solutions

Very good for high-dimensional problems!

Part II: Ensemble Models

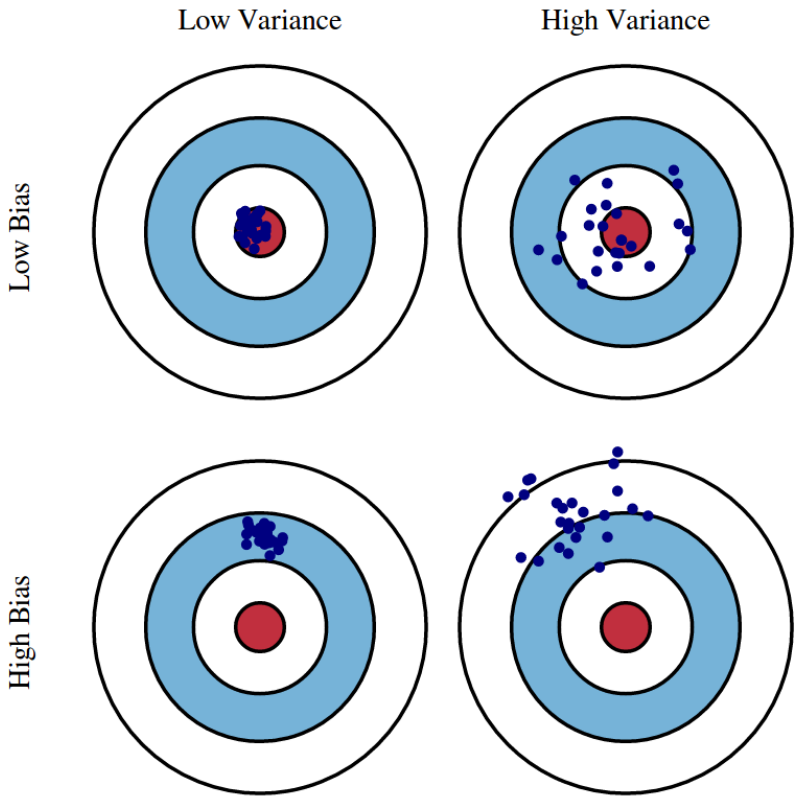
Nonlinear decision boundaries

- Sometimes we encounter problems that linear models can’t solve



Bias-variance trade-off

- Bias
  - $E[\theta - X]$
  - Deviation of the estimator from the true value on average
- Variance
  - $E(X^2) - E(X)^2$
  - Spread of the data from its average
- As one decreases, the other tends to increase
- Low-bias, low-variance is not generally possible



Ensemble Models

- Collections of “weak learners”
  - Decision Trees -> Random Forests
  - Logistic Regression -> Pool of Logistic Regressors
  - Naïve Bayes -> Ensemble of Naïve Bayes
  - Support Vector Machine -> Ensemble of SVMs
- Each is trained independently on a subset of the training data
- Decisions are made by pooling all the individual models’ predictions

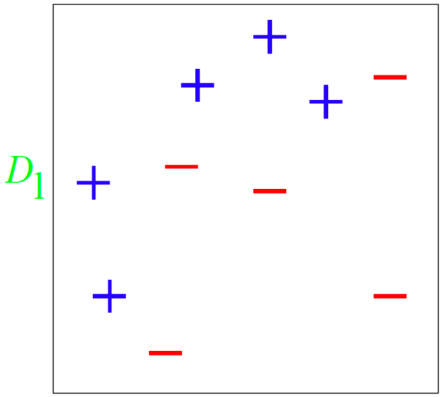
- **Instead of learning a single (weak) classifier, learn many (weak) classifiers that are good at different parts of the input space**

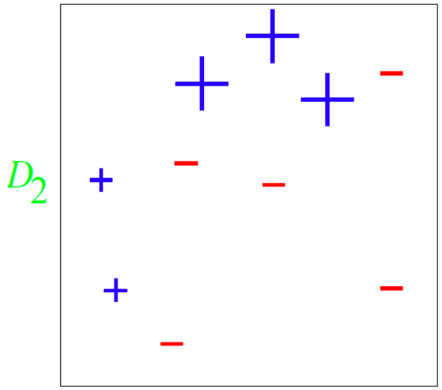
Boosting

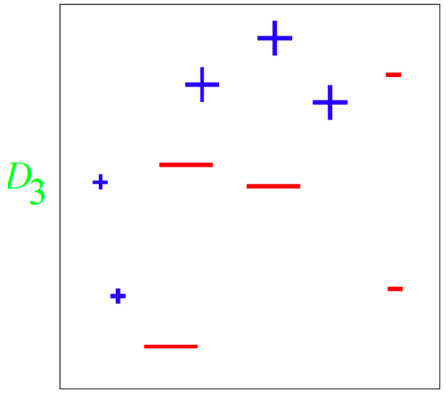
- Given a weak learner, run it multiple times on (reweighted) training data, then let learned classifiers vote
- On each iteration *t*:
  - weight each training example by how incorrectly it was classified
  - learn a weak hypothesis (classifier) h_t
  - a strength of the hypothesis a_t
- Final classifier

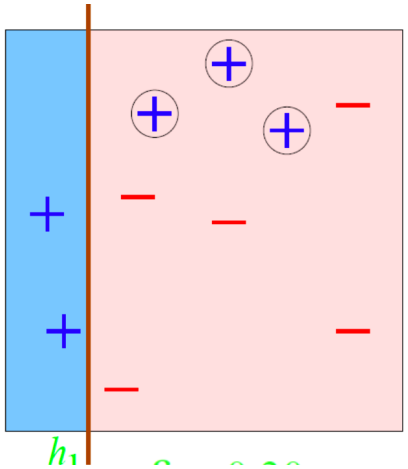
$$H(X) = \text{sign}(\sum \alpha_t \, h_t(X))$$

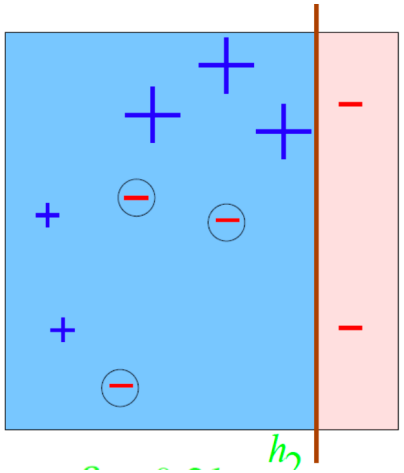
Boosting example

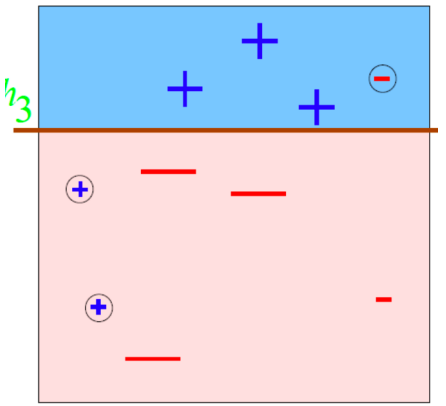




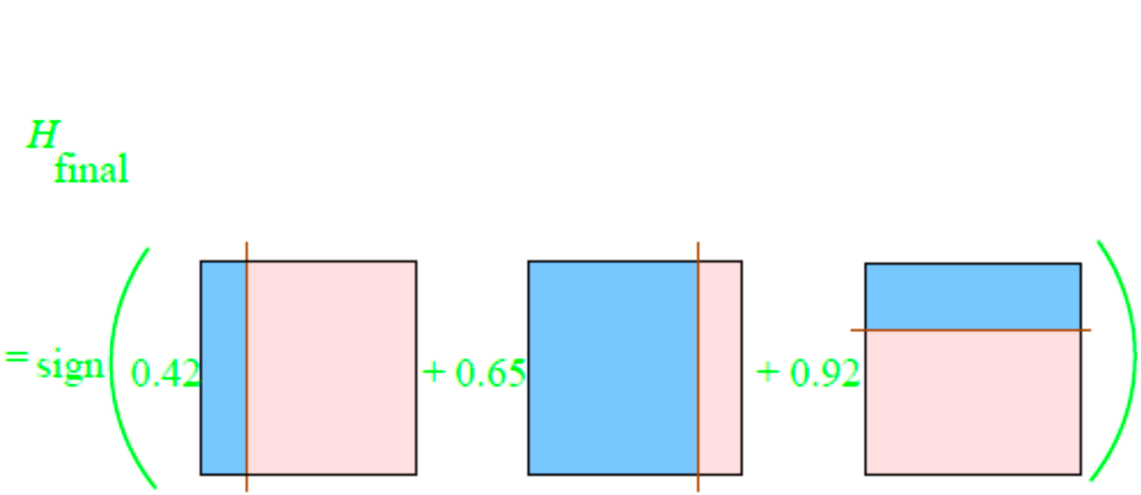


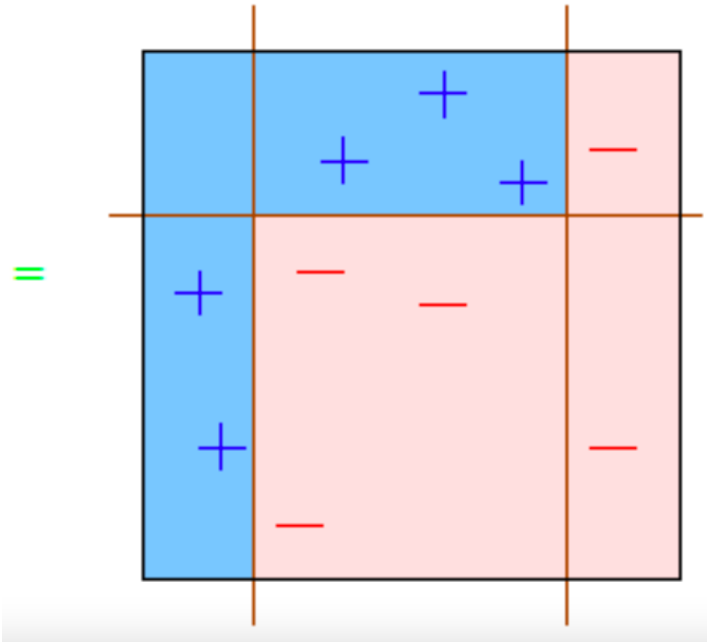






Boosting example





Part III: xgboost

XGBoost

- eXtreme Gradient Boosting
- 29 Kaggle challenges with winners in 2015
  - 17 used XGBoost
  - 8 of these solely used XGBoost; the others combined XGBoost with DNNs
- KDDCup 2015
  - Every single top 10 finisher used XGBoost

XGBoost Applications

- Store sales prediction
- High energy physics event classification
- Web text classification
- Customer behavior prediction
- Motion detection
- Ad click through rate prediction
- Malware classification
- Product categorization
- Hazard risk prediction
- Massive on-line course dropout rate prediction

Properties of XGBoost

- Single most important factor in its success: ***scalability***
- Due to several important systems and algorithmic optimizations

1. Highly scalable end-to-end tree boosting system
2. Theoretically justified weighted quantile sketch for efficient proposal calculation
3. Novel sparsity-aware algorithm for parallel tree learning
4. Effective cache-aware block structure for out-of-core tree learning

What is “tree boosting”?

- Given a dataset (n examples, m features)

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\} \ (|\mathcal{D}| = n, \mathbf{x}_i \in \mathbb{R}^m, y_i \in \mathbb{R})$$

- Tree ensemble uses K additive functions to predict output

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F},$$

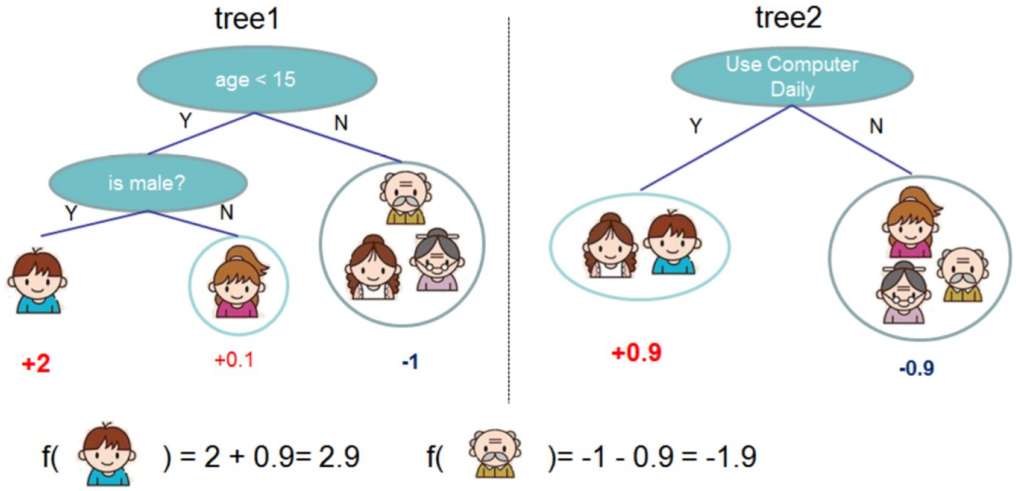


Figure 1: Tree Ensemble Model. The final prediction for a given example is the sum of predictions from each tree.

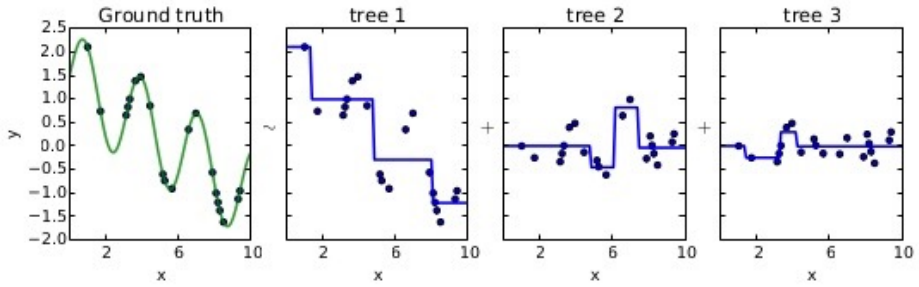
What is “gradient boosting”?

Gradient Boosting [J. Friedman, 1999]

Statistical view on boosting

- ⇒ Generalization of boosting to arbitrary loss functions

Residual fitting



sklearn.ensemble.GradientBoostingClassifier|Regressor

Split-finding algorithms

- Exact
  - Computationally demanding
  - Enumerate all possible splits for continuous features
- Approximate
  - Algorithm proposes candidate splits according to percentiles of feature distributions
  - Maps continuous features to buckets split by candidate points
  - Aggregates statistics and finds best solution among proposals

Comparison of split-finding

- Two variants
  - Global
  - Local

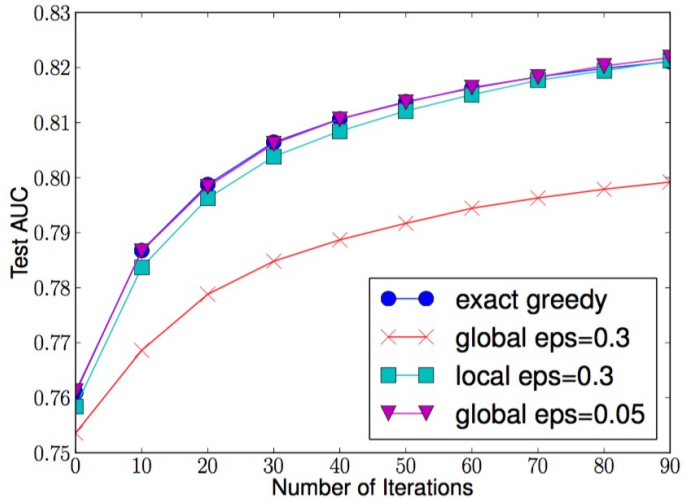


Figure 3: Comparison of test AUC convergence on Higgs 10M dataset. The eps parameter corresponds to the accuracy of the approximate sketch. This roughly translates to 1 / eps buckets in the proposal. We find that local proposals require fewer buckets, because it refine split candidates.

Sparsity-aware split finding

- Equates sparsity with missing values
- Defines a “default” direction: follow the observed paths
- Compare to “dense” method

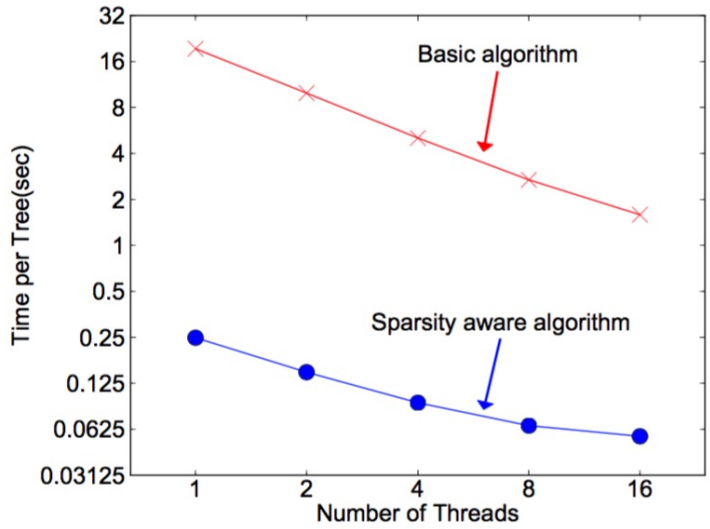
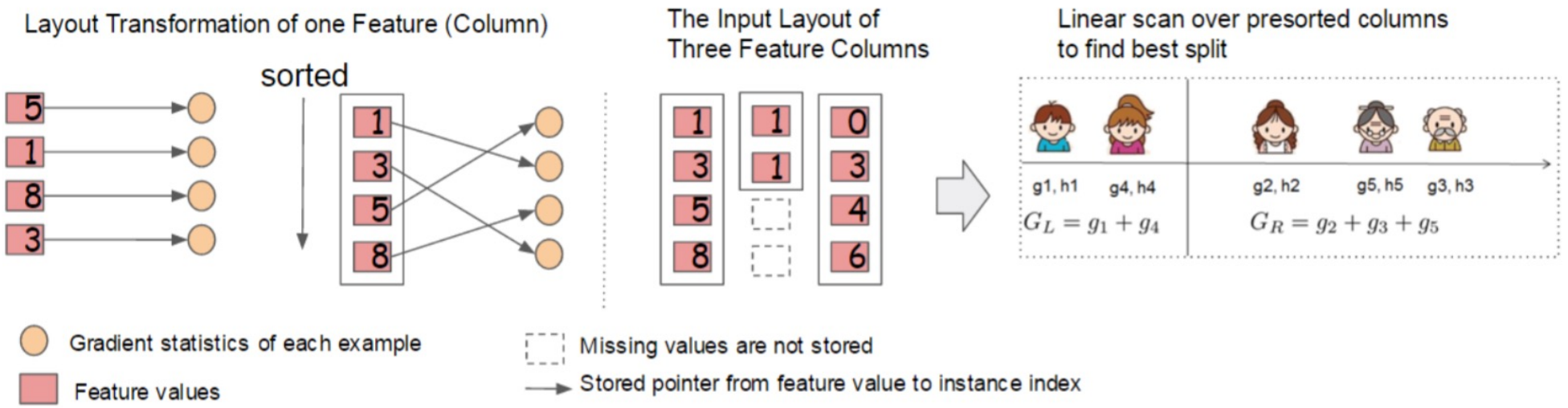


Figure 5: Impact of the sparsity aware algorithm on Allstate-10K. The dataset is sparse mainly due to one-hot encoding. The sparsity aware algorithm is more than 50 times faster than the naive version that does not take sparsity into consideration.

How does this work?

- Features need to be in sorted order to determine splits
- Concept of *blocks*
  - Compressed column (CSC) format
  - Each column sorted by corresponding feature value
- Exact greedy algorithm: all the data in a single block
- Data are sorted once before training and used subsequently in this format

Feature transformations in blocks



More on blocks

- Data is stored on multiple blocks, and these blocks are stored on disk
- Independent threads pre-fetch specific blocks into memory to prevent cache misses
- **Block Compression**
  - Each column is compressed before being written to disk, and decompressed on-the-fly when read from disk into a prefetched buffer
  - Cuts down on disk I/O
- **Block Sharding**
  - Data is split across multiple disks (i.e. cluster)
  - Pre-fetcher is assigned to each disk to read data into memory

Cache-aware access

Exact Greedy Algorithm

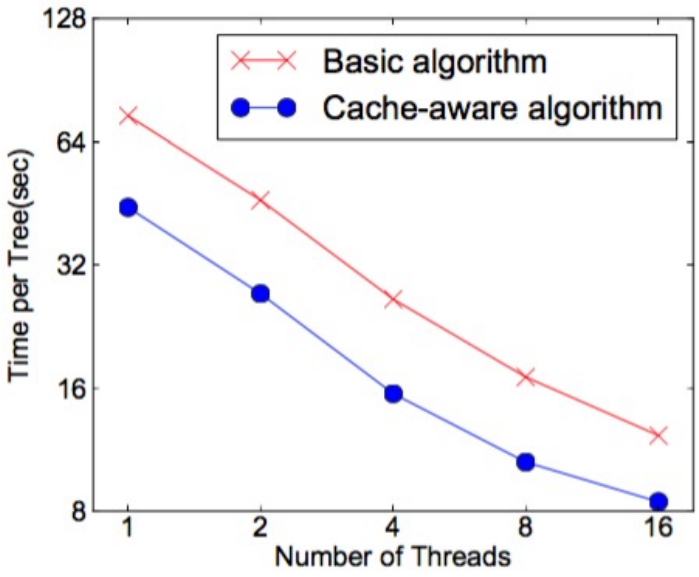
- Allocate an internal buffer in each thread
- Fetch gradient statistics
- Perform accumulation in mini-batch
- Reduces runtime overhead when number of rows is large

Approximate Algorithms

- Choice of block size is critical
- Small block size results in small workloads for each thread
- Large block size results in cache misses as gradient statistics do not fit in cache

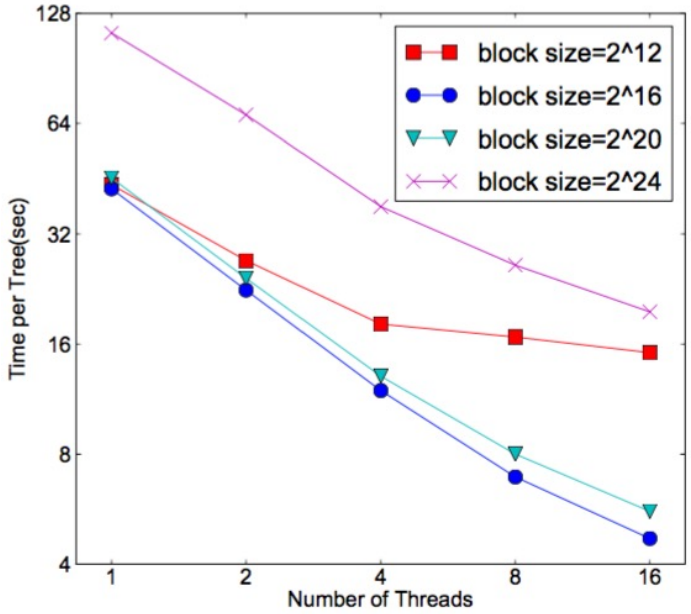
Cache-aware access

Exact



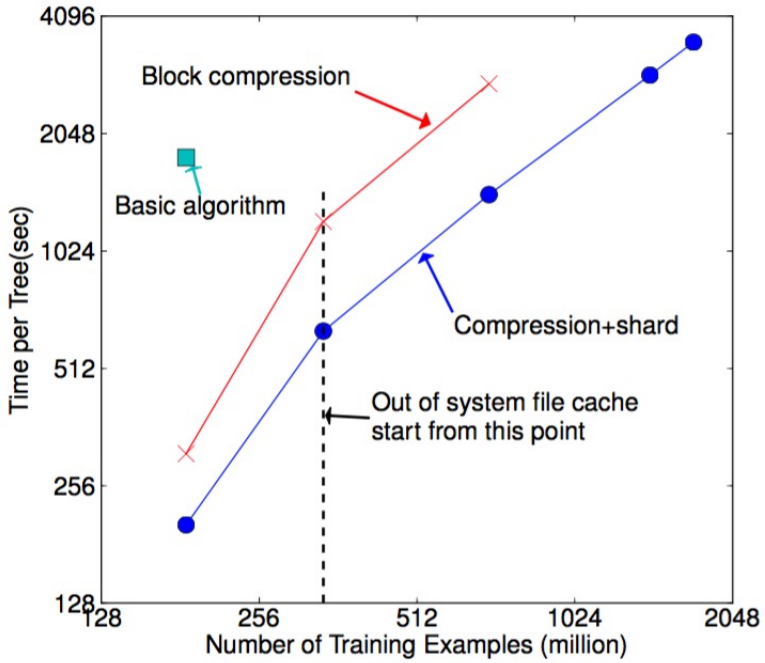
(a) Allstate 10M

Approximate

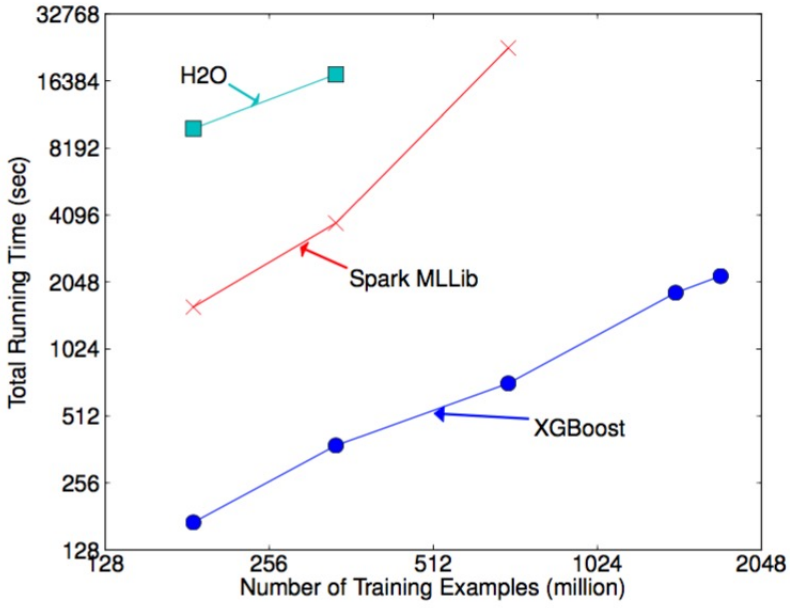


(a) Allstate 10M

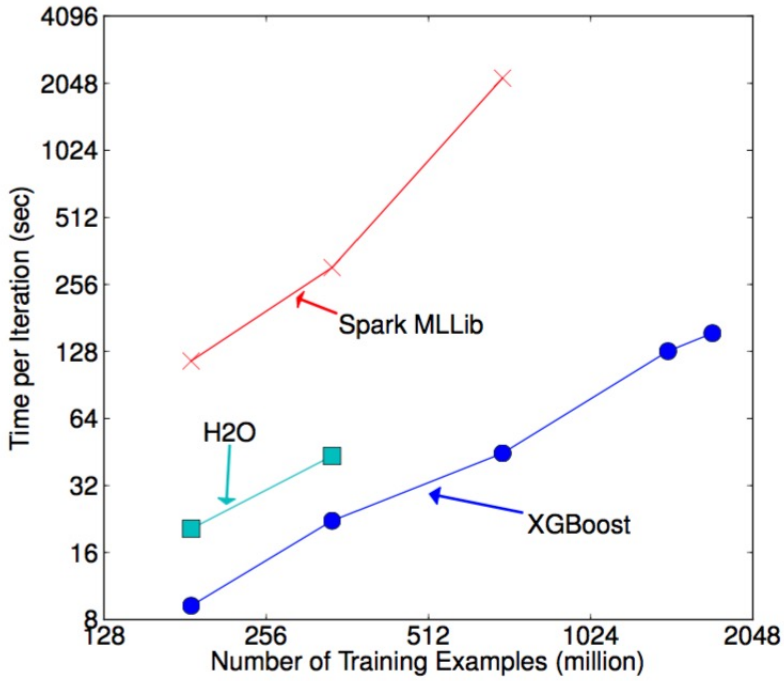
Results: out of core



Results: distributed



(a) End-to-end time cost include data loading



(b) Per iteration cost exclude data loading

Results: scalability

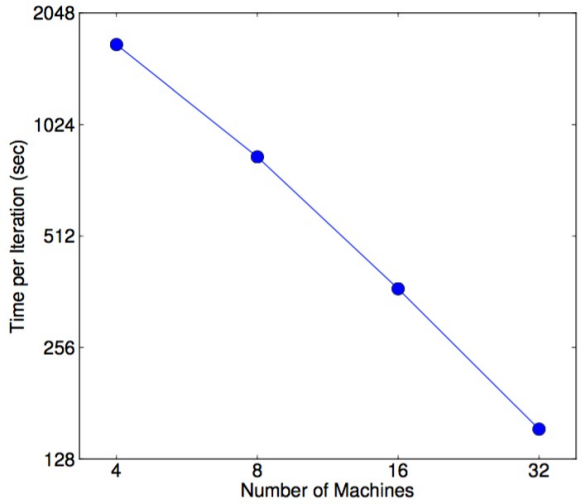


Figure 13: Scaling of XGBoost with different number of machines on criteo full 1.7 billion dataset. Using more machines results in more file cache and makes the system run faster, causing the trend to be slightly super linear. XGBoost can process the entire dataset using as little as four machines, and scales smoothly by utilizing more available resources.

Conclusions

- Most (all?) supervised models are predicated on *minimizing risk*
- Linear models do this by fitting coefficients
  - Can be solved directly (with an inverse)
  - Can be solved iteratively (with gradient descent)
- Regularization can help with fitting but has trade-offs
- Linear models are inherently limited but their utility need not be sacrificed thanks to ensemble models
- Bias-variance trade-off
- Boosting; not to be confused with gradient-boosted trees (xgboost)
  - Combination of algorithmic design and clever hardware implementations make for an easy-to-use and extremely performant ensemble model

References

- *Elements of Statistical Learning,* Chapters 3, 9, and 10
- Xgboost: A Scalable Tree Boosting System
  - http://arxiv.org/abs/1603.02754
- **Xbgoost demo: https://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html**

Regularized objective function

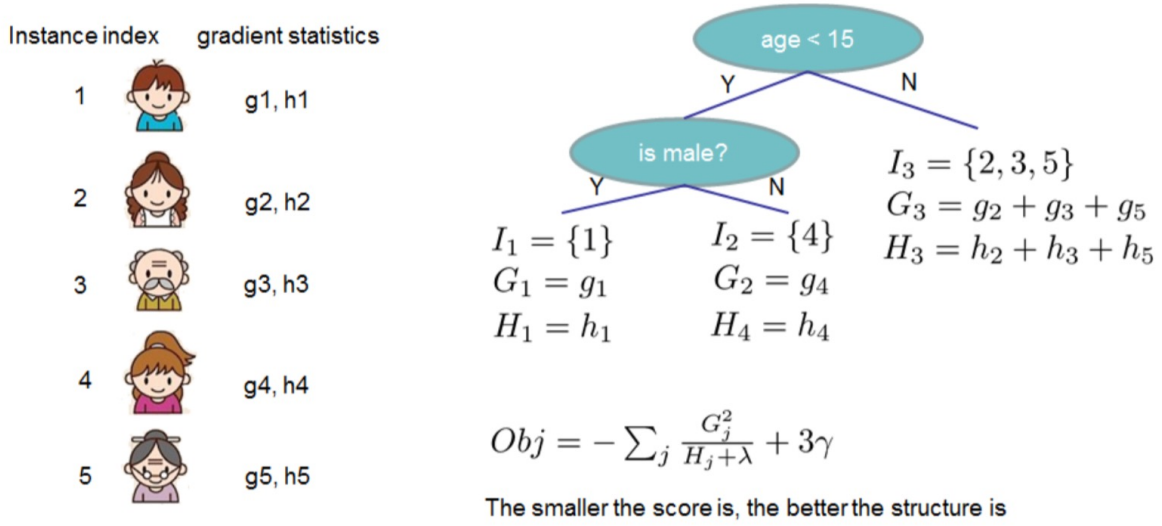


Figure 2: Structure Score Calculation. We only need to sum up the gradient and second order gradient statistics on each leaf, then apply the scoring formula to get the quality score.

Regularized objective function

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

$$\text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

Objective

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t)$$

2nd order approx.

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

Remove constants

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

Scoring function to evaluate quality of tree structure

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T.$$

Shrinkage and column subsampling

- Shrinkage
  - Scales newly added weights by a factor η
  - Reduces influence of each individual tree
  - Leaves space for future trees to improve model
  - Similar to learning rate in stochastic optimization
- Column subsampling
  - Subsample features
  - Used in Random Forests
  - Prevents overfitting more effectively than row-sampling