Supervised Learning II

Big Data Analysis I

Christoph Kern[1] Frauke Kreuter Marcel Neunhoeffer Sebastian Sternberg

March 25, 2019

[1]c.kern@uni-mannheim.de

Table of contents

Introduction

Introduction

Some limitations of (single) trees

- Difficulties in modeling additive structures
- Lack of smoothness of prediction surface
- High variance / **instability** due to hierarchical splitting process

$\rightarrow$ **Ensemble methods**

- Address instability via combining multiple prediction models
- Combine diverse models into a more robust ensemble

Ensembles

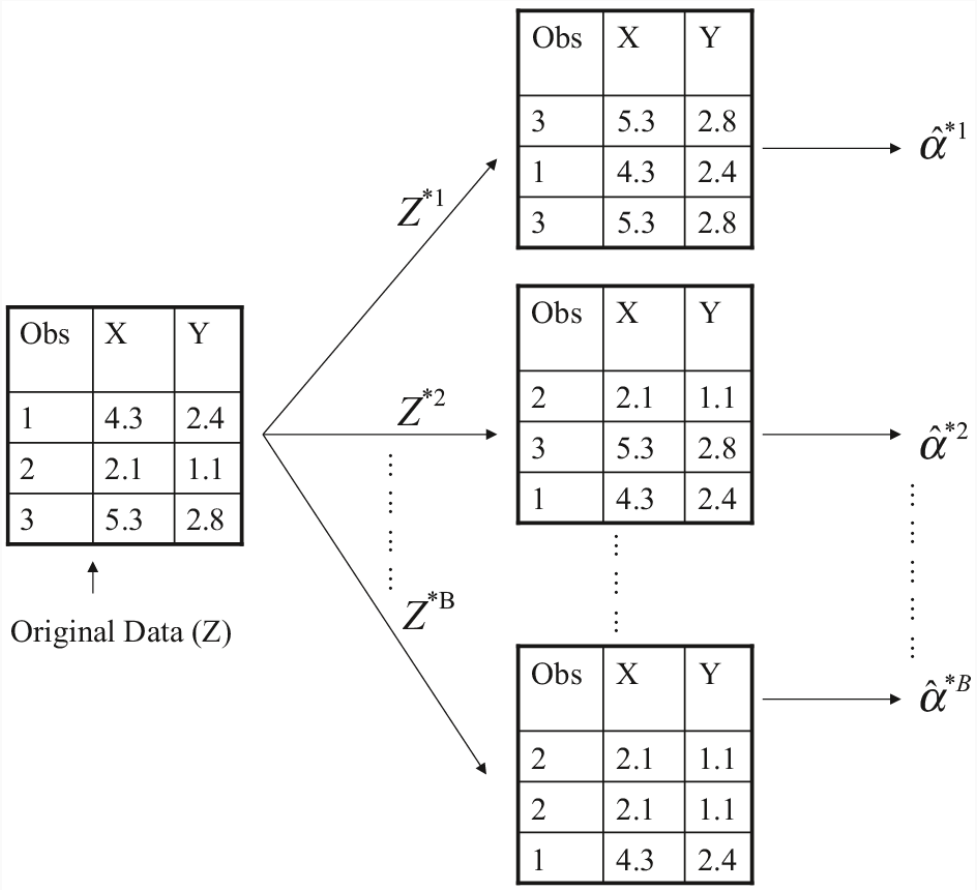
How to construct ensembles?

- Apply one method with different tuning parameter settings
- Combine models with different features
- Use one method with different subsets of the data
  - **Bagging**: Can be applied to different **base learners** (e.g. CART)
- Combine models based on different methods
  - Stacking: Build a meta-model that uses (multiple) predictions as input

Bagging

Bootstrap

Figure 1: Bootstrap process



James et al. (2013)

Bootstrap

Bootstrap: Sampling B samples of size n with replacement from original data set

Applications

- Estimate the variability of model parameters
  - e.g. standard errors of regression coefficients
- Estimate test error with training data
  - Fit model on bootstrap samples and predict original training set
  - ".632" & ".632$^+$" estimator
- Construct an ensemble of learners for prediction
  - **Bagging**: Bootstrap Aggregating
  - Train prediction models on bootstrap samples

Bagging Trees

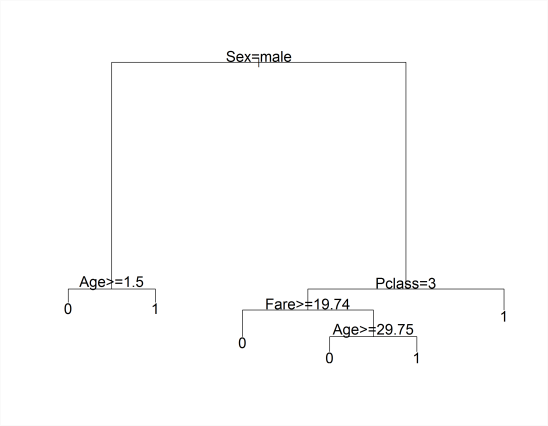
Algorithm 1: Bagging Trees

```
1  Set number of trees B;
2  Define stopping criteria;
3  for b = 1 to B do
4  |   draw a bootstrap sample from the training data;
5  |   assign sampled data to root node;
6  |   if stopping criterion is reached then
7  |   |   end splitting;
8  |   else
9  |   |   find the optimal split point among the predictor space;
10 |   |   split node into two subnodes at this split point;
11 |   |   for each node of the current tree do
12 |   |   |   continue tree growing process;
13 |   |   end
14 |   end
15 end
```

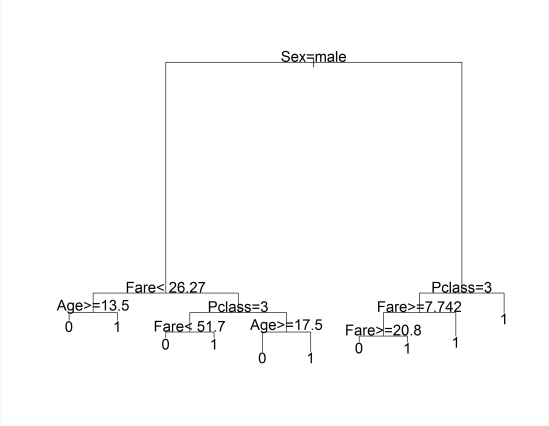
Bagging Trees

Figure 2: Bagging Trees

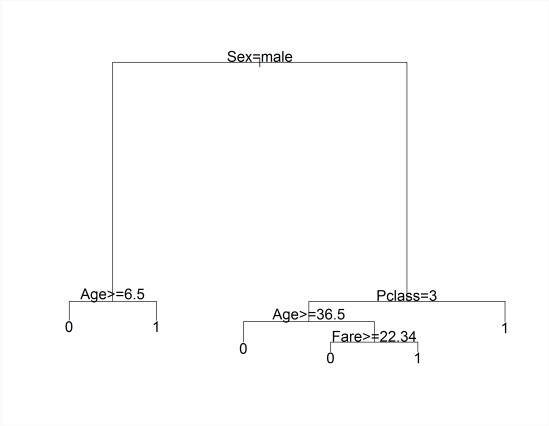
(a) b = 1



(b) b = 2



(c) b = 3



OOB Error

Observations in each bootstrap sample

$$
\begin{aligned}
P(\text{obs } i \in \text{ sample } b) &= 1 - \left(1 - \frac{1}{n}\right)^n \\
&\approx 1 - e^{-1} \\
&= 0.632
\end{aligned}
$$

Out-of-bag (OOB) Error

- Sampling with replacement leads to models based on subsets of the data
- Unused (OOB) observations can be used for test error estimation
  1. Generate predictions for case *i* using models where *i* was OOB
  2. Average predictions for *i* and estimate test error
  3. Compute OOB error over all cases

Bagging Trees

General motivation: Assume training observations (x_i, y_i) from a distribution $\mathcal{P}$ and bootstrap data x_i^*, y_i^* sampled from $\mathcal{P}$

Aggregate estimator: $f_{ag}(x) = \mathrm{E}_{\mathcal{P}}\hat{f}^*(x)$

$$\begin{aligned}\mathrm{E}_{\mathcal{P}}[Y - \hat{f}^*(x)]^2 &= \mathrm{E}_{\mathcal{P}}[Y - f_{ag}(x) + f_{ag}(x) - \hat{f}^*(x)]^2 \\ &= \mathrm{E}_{\mathcal{P}}[Y - f_{ag}(x)]^2 + \mathrm{E}_{\mathcal{P}}[\hat{f}^*(x) - f_{ag}(x)]^2 \\ &\geq \mathrm{E}_{\mathcal{P}}[Y - f_{ag}(x)]^2\end{aligned}$$

$\rightarrow$ Suggests that Bagging decreases mean-squared error

Random Forests

Random Forests

From Bagging to Random Forests

Variance of an average of B i.i.d. random variables

$$\frac{1}{B}\sigma^2$$

$\rightarrow$ Bagging: Averaging over B trees decreases variance

Variance of an average of B i.d. random variables with $\rho > 0$

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

$\rightarrow$ **Random Forests**: Averaging over B trees with m out of p predictors per split decreases variance and decorrelates trees

Random Forests

From Bagging to Random Forests

Variance of an average of B i.i.d. random variables

$$\frac{1}{B}\sigma^2$$

$\rightarrow$ Bagging: Averaging over B trees decreases variance

Variance of an average of B i.d. random variables with $\rho > 0$

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

$\rightarrow$ **Random Forests**: Averaging over B trees with m out of p predictors per split decreases variance and decorrelates trees

Random Forests

The Random Forest trick (Breiman 2001)

- Randomization with respect to rows *and* columns
- Weaker predictors have more of a chance
- Results in diverse and *decorrelated* trees

Can be taken one step further...

1. Draw a random sample m from the p predictors (w/o Bootstrapping)
2. Draw random split(s) per feature
3. Split node using the best of these random splits

$\rightarrow$ Extremely Randomized Trees (Geurts et al. 2006)

Random Forests

The Random Forest trick (Breiman 2001)

- Randomization with respect to rows *and* columns
- Weaker predictors have more of a chance
- Results in diverse and *decorrelated* trees

Can be taken one step further...

1. Draw a random sample m from the p predictors (w/o Bootstrapping)
2. Draw random split(s) per feature
3. Split node using the best of these random splits

$\rightarrow$ Extremely Randomized Trees (Geurts et al. 2006)

Growing a Forest

Growing a Forest

Algorithm 2: Grow a Random Forest

```
1  Set number of trees B;
2  Set predictor subset size m;
3  Define stopping criteria;
4  for b = 1 to B do
5  |   draw a bootstrap sample from the training data;
6  |   assign sampled data to root node;
7  |   if stopping criterion is reached then
8  |   |   end splitting;
9  |   else
10 |   |   draw a random sample m from the p predictors;
11 |   |   find the optimal split point among m;
12 |   |   split node into two subnodes at this split point;
13 |   |   for each node of the current tree do
14 |   |   |   continue tree growing process;
15 |   |   end
16 |   end
17 end
```

Growing a Forest

A Random Forest

$$\{\mathcal{T}_b\}_1^B$$

consists of a set of $b = 1, 2, \ldots, B$ trees which can be used for prediction by...

- Regression
  - Averaging predictions over all trees
  - $\hat{f}_{rf}^B(x) = \frac{1}{B}\sum_{b=1}^B \mathcal{T}_b(x)$
- Classification
  - Using most commonly occurring class among all trees
  - $\hat{C}_{rf}^B(x) = \text{majority vote}\{\hat{C}_b(x)\}_1^B$
- Probability estimation
  - Using the proportion of class votes of all trees
  - Averaging predicted probabilities over all trees

Tuning RFs

Tuning Random Forests

- Predictor subset size m out of p
  - Most important tuning parameter in RF
  - Starting value; $m = \sqrt{p}$ (classification), $m = p/3$ (regression)
  - Can be chosen using OOB errors based on different m
- Number of trees
  - sufficiently high (e.g. 500)
- Node size (number of observations in terminal nodes)
  - sufficiently low (e.g. 5)

Interpretation

Interpretation

Interpreting Random Forests

- Inspect each tree of the forest
  - Inefficient for 500+ trees
- Variable importance
  - Summary of “effect size”
- Partial dependence plots
  - Graphical representation of “effect structure”

Variable Importance

Variable importance with CART

$$\mathcal{I}_\ell^2(T) = \sum_{t=1}^{J-1} \hat{\imath}_t^2 I(v(t) = \ell)$$

- Sum of squared improvements $\hat{\imath}^2$ over all internal nodes with predictor X_ℓ
  - Regression: Overall reduction in RSS caused by X_ℓ
  - Classification: Overall reduction of impurity caused by X_ℓ

Importance with Random Forests

$$\mathcal{I}_\ell^2 = \frac{1}{M} \sum_{m=1}^{M} \mathcal{I}_\ell^2(T_m)$$

- Average improvement caused by predictor X_ℓ over all trees

Variable Importance

Permutation feature importance[1]

1. Estimate the original model error $e_{orig}(\hat{f}) = L(Y, \hat{f}(X))$
2. For each feature $j \in 1, \ldots, p$
   - 2.1 Generate feature matrix $X_{perm\,j}$ by permuting the values of feature X_j in X
   - 2.2 Estimate error $e_{perm} = L(Y, \hat{f}(X_{perm\,j}))$ based on the predictions of the permuted data
   - 2.3 Calculate permutation feature importance $FI_j = \frac{e_{perm}(\hat{f})}{e_{orig}(\hat{f})}$ or via $FI_j = e_{perm}(\hat{f}) - e_{orig}(\hat{f})$
3. Output FI for all variables

[1] https://christophm.github.io/interpretable-ml-book/

Partial Dependence Plots

Plotting results from “black box” learning methods

$$\tilde{f}(x) = \frac{1}{n} \sum_{i=1}^{n} f(x, x_{iC})$$

- Compute $\tilde{f}(x)$ over the range of x while averaging the effects of the remaining predictors x_{iC}
- Generate artificial datasets by fixing x for all cases
  - Regression: Averaging over $f(x, x_{iC})$ for each value of x
  - Classification: Averaging over $\text{logit}(p)$ for each value of x
- Outlook: ICE plots (Goldstein et al. 2014)

Partial Dependence Plots

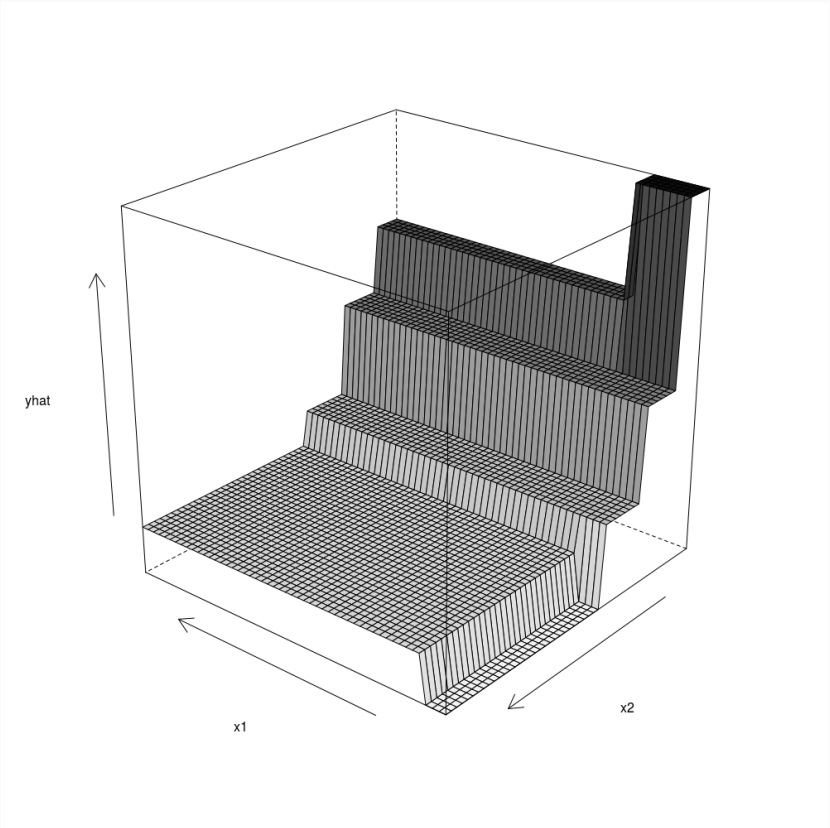
Constructing PDPs

1. Choose a range of values $\{x_{11}, x_{12}, \ldots, x_{1k}\}$ of x_1
2. For each $i \in \{1, 2, \ldots, k\}$
   - 2.1 Generate an artificial dataset by fixing x_1 to x_{1i} for all cases
   - 2.2 Compute predictions for all cases using the prediction model (e.g. RF)
   - 2.3 Average the predictions over all cases
3. Plot the obtained average predictions against x_{1i} for $i = 1, 2, \ldots, k$

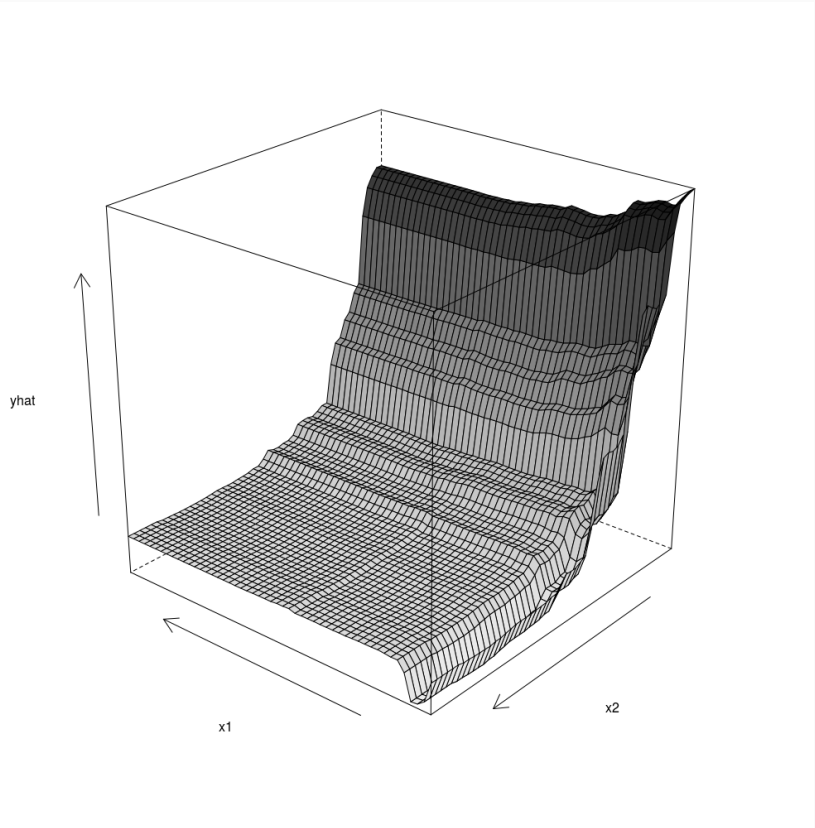
Partial Dependence Plots

Figure 3: Partial dependence plots

(a) CART



(b) Random Forest



Boosting

Boosting

- Class of ensemble methods which combine **sequential** prediction models
- Adaptive approach with focus on “difficult observations”
- Different flavors exist
  - AdaBoost
  - Gradient Boosting Machines (GBM)
  - ...
- Can be applied to different (weak) base learners
  - Boosting trees
  - ...

AdaBoost

AdaBoost

- Algorithm for classification problems ($Y \in \{-1, 1\}$)
- Estimate a sequence of classifiers using reweighted data
- AdaBoost process
  1. Fit classifier $G_m(x)$ to weighted data (intitial weights $w_i = \frac{1}{n}$)
  2. Compute the misclassification rate

$$\text{err}_m = \frac{\sum_{i=1}^{n} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{n} w_i}$$

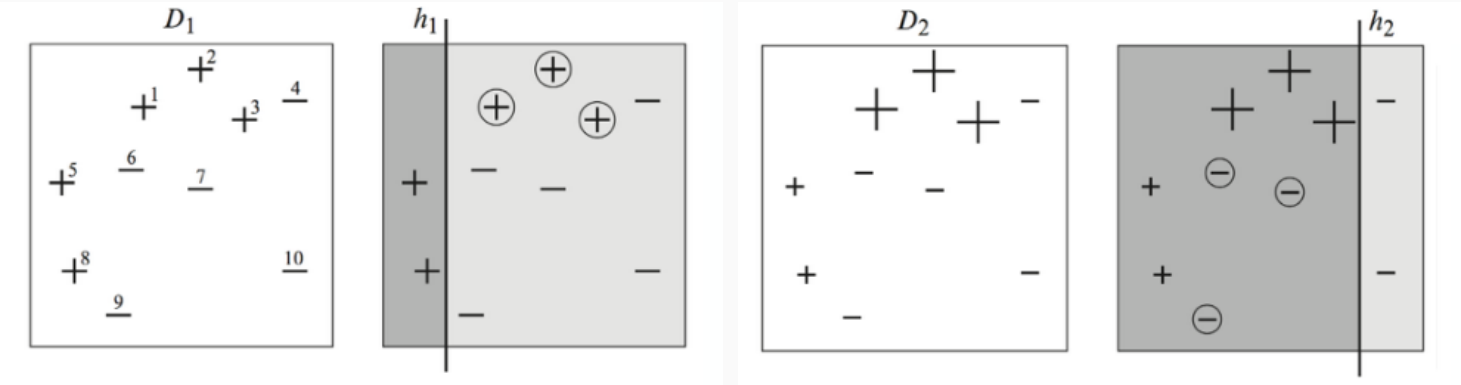
  3. Compute the classifier weight $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$
  4. Recalculate weights $w_i = w_i \exp(\alpha_m I(y_i \neq G_m(x_i)))$
- Majority vote classification: $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$

Boosting Stumps

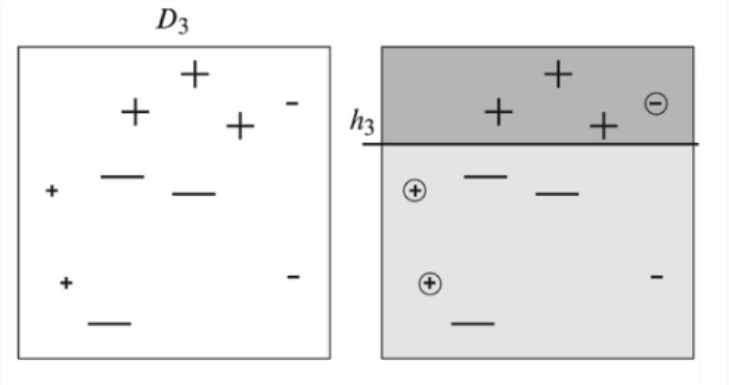
Figure 4: (Ada)Boosting stumps (example)[2]

(a) Step 1: $\alpha_1 = 0.42$

(b) Step 2: $\alpha_2 = 0.65$



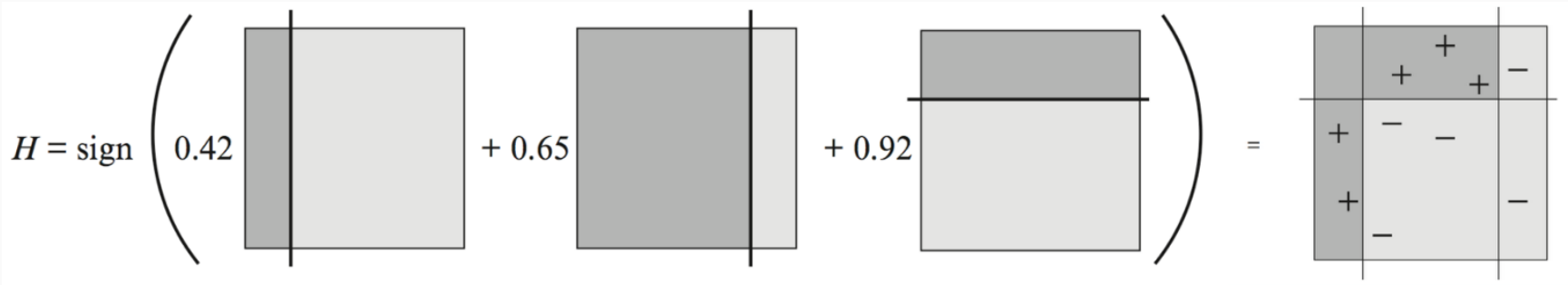
(c) Step 3: $\alpha_3 = 0.92$



[2] Source: Shapire & Freund 2012

Boosting Stumps

Figure 5: Step 4: Combine models



GBM

Gradient Boosting Machines (GBM)

- General approach to sequential learning
- Applicable with various loss functions
- Boosting trees
  1. Initialize model (with a constant $f_0(x)$)
  2. Compute pseudo-residuals based on current model

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$

  3. Fit a regression tree to the pseudo-residuals
  4. Compute $\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$
  5. Update the current model: $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$
- Output $\hat{f}(x) = f_M(x)$

$\rightarrow$ Analogue to steepest descent

GBM

Table 1: GBM components for different loss functions

Setting	Loss function	r_i	$f_0(x)$
Regression	$\frac{1}{2}(y_i - f(x_i))^2$	$y_i - f(x_i)$	$\text{mean}(y_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}(y_i - f(x_i))$	$\text{median}(y_i)$
Classification	Deviance	$I(y_i = G_k) - p_k(x_i)$	prior p's

Shrinkage, Subsampling, Tuning

Shrinkage

- Additional tweak in Gradient boosting
- Slow down learning rate to avoid overfitting
- Learning rate is controlled by λ
  - $f_m(x) = f_{m-1}(x) + \lambda \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

Subsampling

- Optional add-on in Gradient boosting
- Use a random sample (w/o replacement) of pseudo-residuals in each step
- Can be introduced to improve performance and speed
  - “Stochastic gradient boosting”

Shrinkage, Subsampling, Tuning

Tuning Gradient Boosting Machines

- Number of trees M
  - Number of "iterations"
  - Overfitting can occur for large M
- Interaction depth D
  - Number of splits for each tree
  - Boosting stumps: $D = 1$
- Shrinkage parameter λ
  - e.g. $\lambda = 0.01$, $\lambda = 0.001$
  - Smaller λ needs larger M
- ...

Boosting for Regression

Algorithm 3: Gradient Boosting for regression

Set number of trees M;
Set interaction depth D;
Set shrinkage parameter λ;
Use $\bar{y}$ as initial prediction;
for $m = 1$ *to* M **do**
    compute residuals based on current predictions;
    assign data to root node, using the residuals as the outcome;
    while *current tree depth* $< D$ **do**
        tree growing process;
    end
    compute the predicted values of the current tree;
    add the shrinked new predictions to the previous predicted values;
end

Boosting for Regression

Table 2: Boosting Example with 5 obs and 2 x's

ID	x_1	x_2	y	$f_0(x)$
1	0	0	1	1.2
2	0	2	3	1.2
3	1	2	2	1.2
4	2	3	0	1.2
5	0	1	0	1.2

Boosting for Regression

Table 3: Step 1: Split $x_2 > 2.5$

ID	x_1	x_2	y	$f_0(x)$	r_{i1}	γ_{j1}	$f_1(x)$
1	0	0	1	1.2	−0.2	0.3	1.5
2	0	2	3	1.2	1.8	0.3	1.5
3	1	2	2	1.2	0.8	0.3	1.5
4	2	3	0	1.2	−1.2	−1.2	0
5	0	1	0	1.2	−1.2	0.3	1.5

Table 4: Step 2: Split $x_2 < 1.5$

ID	x_1	x_2	y	$f_0(x)$	$f_1(x)$	r_{i2}	γ_{j2}	$f_2(x)$
1	0	0	1	1.2	1.5	−0.5	−1	0.5
2	0	2	3	1.2	1.5	1.5	0.66	2.166
3	1	2	2	1.2	1.5	0.5	0.66	2.166
4	2	3	0	1.2	0	0	0.66	0.66
5	0	1	0	1.2	1.5	−1.5	−1	0.5

XGBoost

XGBoost

Extreme Gradient Boosting

- Widely used (and competitive) in ML challenges
- Introduces regularization and a modified splitting criterion
- Scalable due to various algorithmic optimizations
  - Sparsity-aware split finding
  - Multicore processing
- Trees as base learners (`xgbtree`), or linear models (`xgblinear`)

$\rightarrow$ Chen and Guestrin 2016

XGBoost

The XGBoost ensemble

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

with K functions f of the set of all possible trees $\mathcal{F}$

Regularized objective function

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

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

with number of leaves T, vector of leaf scores w, regularization parameters γ, λ

XGBoost

Objective of a sequence of XGBoost trees

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

Optimization via second-order approximation

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} (g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)) + \Omega(f_t)$$

with first and second order gradient statistics g_i, h_i

$\rightarrow$ Tree quality score

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

XGBoost

Objective of a sequence of XGBoost trees

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

Optimization via second-order approximation

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} (g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)) + \Omega(f_t)$$

with first and second order gradient statistics g_i, h_i

$\rightarrow$ Tree quality score

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

XGBoost

Objective of a sequence of XGBoost trees

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

Optimization via second-order approximation

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} (g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)) + \Omega(f_t)$$

with first and second order gradient statistics g_i, h_i

$\rightarrow$ Tree quality score

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

XGBoost Tuning

- `nrounds`
  - Number of trees
- `eta`
  - Shrinkage, learning rate
- `max_depth`
  - Maximum tree depth
- `subsample`
  - Subsample ratio of observations
- `gamma`
  - Minimum loss reduction required to split
- `colsample_bytree`
  - Subsample ratio of columns

→ `https://xgboost.readthedocs.io/en/latest/parameter.html`

Summary

Summary

- Bagging and random forests stabilize high-variance trees
- Boosting sequentially combines multiple models into a powerful ensemble
- Random forests, boosting are “general purpose” approaches
- A lot of different flavors exist
    - Specific implementations can be compared in a large train and tune loop
- Drawbacks: Low interpretability and (often) high computational costs

Software Resources

Software Resources

Resources for R I

- Standard package to grow RFs: `randomForest`
- Fast implementation of RFs: `ranger`
- Extremely Randomized Trees: `extraTrees`
- Visualization
  - Partial Dependence Plots: `pdp`
  - Plot model surfaces (also PDPs): `plotmo`

Software Resources

Resources for R II

- AdaBoost: `fastAdaboost`
  - AdaBoost implementation with C++ backend
- Standard package for Gradient Boosting: `gbm`
  - Implementation and extensions of AdaBoost and GBM
- Extreme Gradient Boosting: `xgboost`
  - Competitive and scalable boosting approach

References

References I

Breiman, L. (1996). Bagging Predictors. *Machine Learning, 24*(2), 123–140.

Breiman, L. (2001). Random forests. *Machine Learning 45*(1), 5–32.

Chen, T., Guestrin, C. (2016). *XGBoost: A scalable tree boosting system.* `https://arxiv.org/abs/1603.02754`.

Friedman, J. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics, 29*(5), 1189–1232.

Geurts, P., Ernst, D., Wehenkel, L. (2006). Extremely Randomized Trees. *Machine Learning 63*(1), 3–42.

Goldstein, A., Kapelner, A., Bleich, J., Pitkin, E. (2014). *Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation*. arXiv: 1309.6392v2.

Hastie, T., Tibshirani, R., Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York, NY: Springer.

References II

James, G., Witten, D., Hastie, T., Tibshirani, R. (2013). *An Introduction to Statistical Learning*. New York, NY: Springer.

Schapire, R. E. and Freund, Y. (2012). *Boosting: Foundations and Algorithms*. Cambridge, MA: MIT Press.