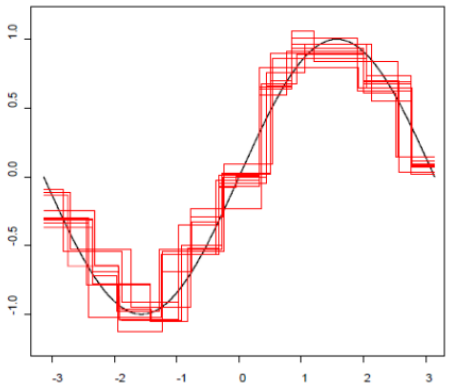
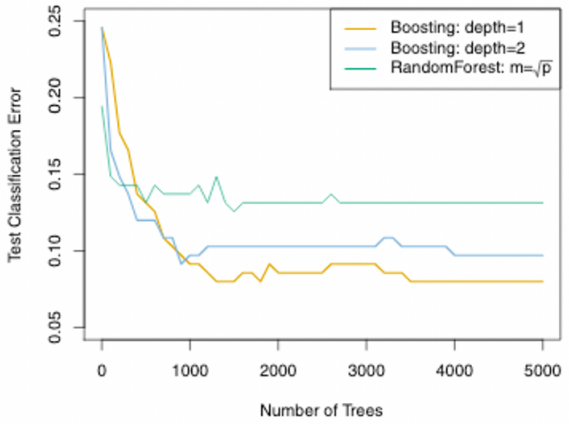
Tufts COMP 135: Introduction to Machine Learning

https://www.cs.tufts.edu/comp/135/2020f/

Boosting

Ensembles of Sequentially-Dependent Predictors





Prof. Mike Hughes

Many slides attributable to:
Liping Liu and Roni Khardon (Tufts)
T. Q. Chen (UW),
James, Witten, Hastie, Tibshirani (ISL/ESL books)

Ensembles: Unit Objectives

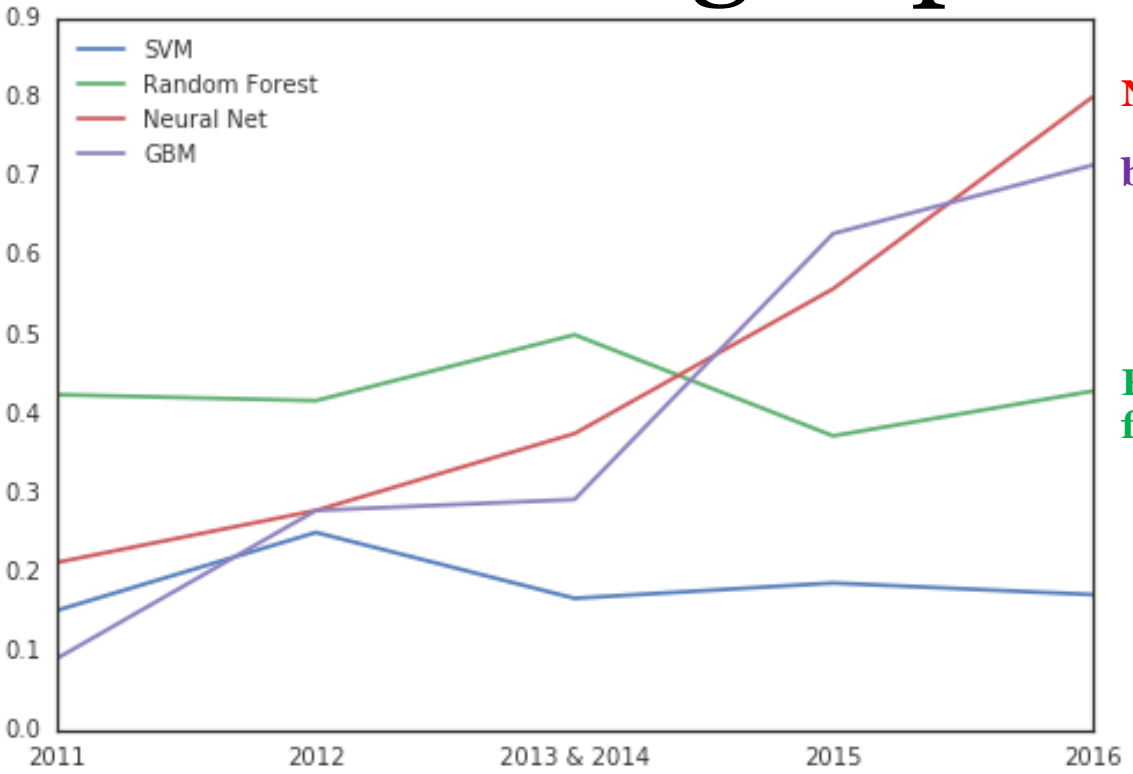
Big idea: We can improve performance by aggregating decisions from MANY predictors

- Prev. class: Predictors are **Independently** Trained
  - Using bootstrap samples of examples: "Bagging"
  - Using random subsets of features
  - Exemplary method: Random Forest / ExtraTrees
- Today: Predictors are **Sequentially** Trained
  - Each successive predictor "boosts" performance
  - How to use gradients to improve further
  - Exemplary method: XGBoost

Motivation: Boosting in practice

Fraction of competitions won by different methods

(some competitions have multiple winners, each with different strategies)



Neural nets

boosting

Random forests

Among 29 Kaggle competitions in 2015

- 17 / 29 (58%) used XGBoost
- 11 / 29 (37%) used deep neural networks

Source: https://www.kaggle.com/antgoldbloom/what-algorithms-are-most-successful-on-kaggle

Ensemble Method: **Sequentially** Predict Residual

- Model 1: Trained to predict original y in train set
  - `model1.fit(xtr_NF, ytr_N)`

```
# Compute the Residual Error
r1_N = ytr_N - model1.predict(xtr_NF)
```

Ensemble Method: **Sequentially** Predict Residual

- Model 1: Trained to predict original y in train set
  - `model1.fit(xtr_NF, ytr_N)`

- Model 2: Trained to predict residual from model 1
  - `r1_N = ytr_N - model1.predict(xtr_NF)`
  - `model2.fit(xtr_NF, r1_N)`

Ensemble Method: **Sequentially** Predict Residual

- Model 1: Trained to predict original y in train set
  - `model1.fit(xtr_NF, ytr_N)`
- Model 2: Trained to predict residual from model 1
  - `r1_N = ytr_N - model1.predict(xtr_NF)`
  - `model2.fit(xtr_NF, r1_N)`
- Model 3: Trained to predict residual from model 2
  - `r2_N = ytr_N - (model1.predict(xtr_NF) + model2.predict(xtr_NF))`
  - `model3.fit(xtr_NF, r2_N)`

Boosting for Regression Trees

ISL textbook

Algorithm 8.2 *Boosting for Regression Trees*

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.

2. For $b = 1, 2, \ldots, B$, repeat:

   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).

   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:

   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$

   (c) Update the residuals,

   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

Boosting with depth-1 tree

A depth-1 tree is called a "stump"

(Be careful, depth-1 still has branches, so maybe "sapling" is a better name than "stump")

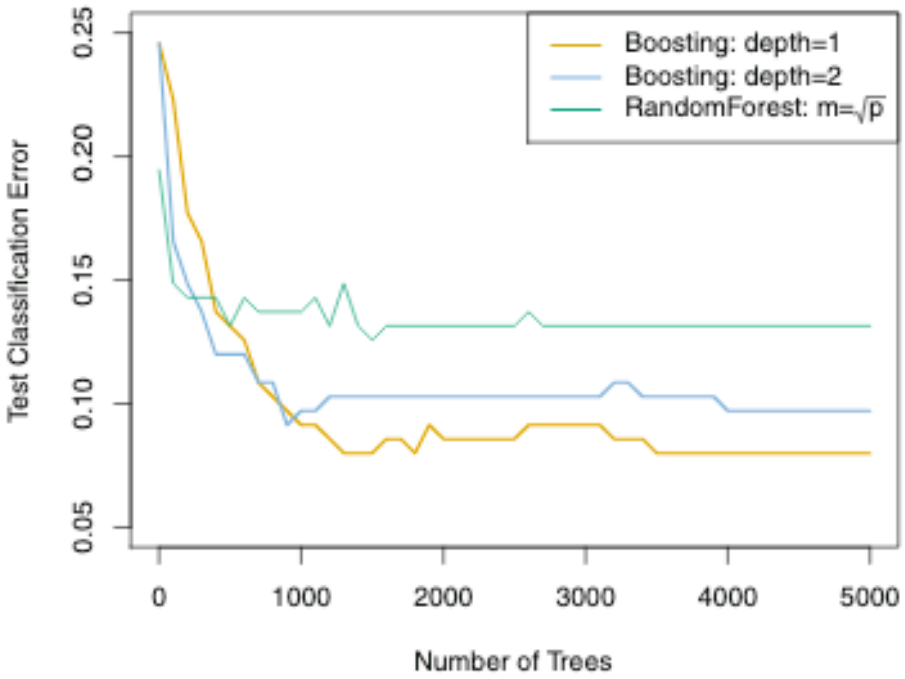


FIGURE 8.11. *Results from performing boosting and random forests on the 15-class gene expression data set in order to predict* cancer *versus* normal. *The test error is displayed as a function of the number of trees. For the two boosted models,* $\lambda = 0.01$. *Depth-1 trees slightly outperform depth-2 trees, and both outperform the random forest, although the standard errors are around 0.02, making none of these differences significant. The test error rate for a single tree is 24 %.*

Credit: ISL textbook

Regularization of boosted trees

Regularization

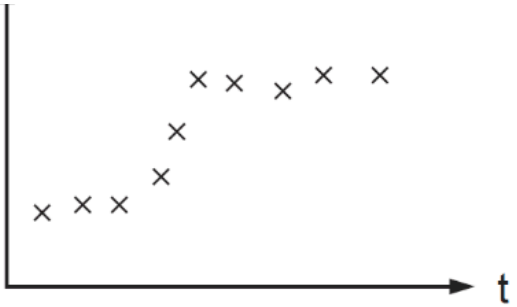
$$Obj(\Theta) = L(\Theta) + \Omega(\Theta)$$

Training Loss measures how well model fit on training data

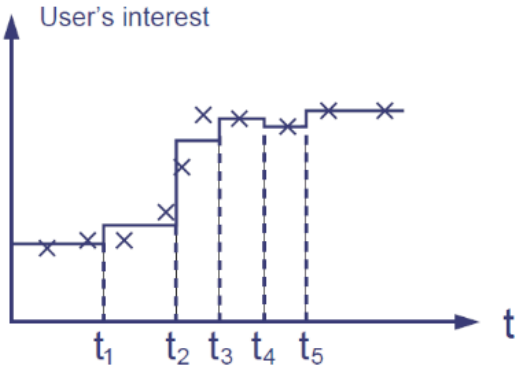
Regularization, measures complexity of model

How to measure complexity?

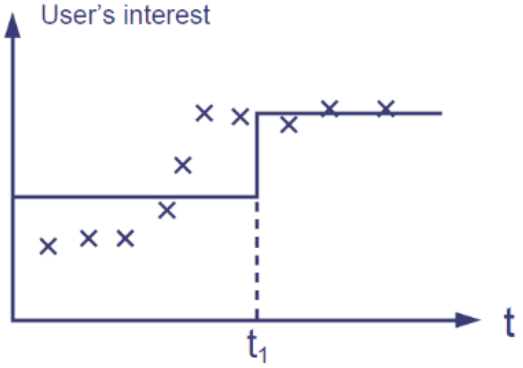
- Number of nodes in tree
- Depth of tree
- Scalar prediction in region (L2 penalty)



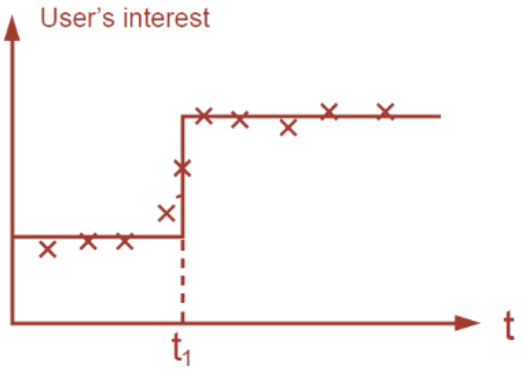
Observed user's interest on topic k against time t



☒ Too many splits, $\Omega(f)$ is high



☒ Wrong split point, $L(f)$ is high



☑ Good balance of $\Omega(f)$ and $L(f)$

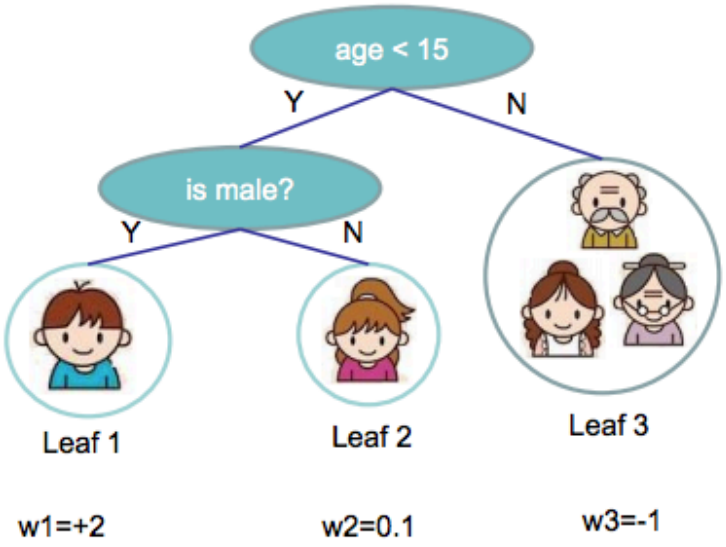
Credit: T. Chen
https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf

Example Regularization Term

- Define complexity as (this is not the only possible definition)

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Number of leaves **L2 norm of leaf scores**



$$\Omega = \gamma 3 + \frac{1}{2}\lambda(4 + 0.01 + 1)$$

Credit: T. Chen
https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf

Regularization when boosting

https://xgboost.readthedocs.io/

Minimization objective when adding tree *t*:

$$\text{obj}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$

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

Loss function

Regularization
(limit complexity of tree t)

Gradient boosting

Ensembles of *general* functions

Suppose we have M *functions* in an ensemble

$$f_M(x) = \sum_{m=1}^{M} f_m(x)$$

How to update the m-th function to reduce loss?

$$\hat{\mathbf{f}} = \arg\min_{\mathbf{f}} L(\mathbf{f})$$

$$\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m \mathbf{g}_m \qquad g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}$$

Motivates an update rule that could be applied to any differentiable loss

Gradient Boosting: Keep adding trees Fit each one to the gradient

"Standard" boosting: fit each tree to the residual

Might be **difficult** for many loss functions other than squared error

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)\right)$$

Gradient boosting: fit each tree to match the gradient wrt previous predictions

Should be **easy** for any differentiable loss function

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} \left(-g_{im} - T(x_i; \Theta)\right)^2$$

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

Gradient Boosting Algorithm

From ESL textbook

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
   Compute gradient
   At each training example
   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.
   Decide tree structure by fitting to gradients
   (c) For $j = 1, 2, \ldots, J_m$ compute
   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
   Decide leaf values by minimizing loss given structure
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
   Add up trees to get the final model
3. Output $\hat{f}(x) = f_M(x)$.

To Improve Gradient Boosting

Can extend gradient boosting with

- Second-order approximation of loss
- Smart approximate split finding (speed)
- Penalties on tree complexity
- Very smart practical implementation (speed)
  - Parallel computation, sparsity-awareness

Result: **Extreme** Gradient Boosting

aka XGBoost (T. Chen & C. Guestrin)

XGBoost:
Extreme Gradient Boosting

```
# XGBoost
from xgboost import XGBClassifier
clf = XGBClassifier()
# n_estimators = 100 (default)
# max_depth = 3 (default)
clf.fit(x_train,y_train)
clf.predict(x_test)
```

More details (beyond this class)

ESL textbook, Section 10.10

Good slide deck by T. Q. Chen (first author of XGBoost):

- https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf

Summary of Boosting

PRO

- Like all tree methods, invariant to scaling of inputs (no need for careful feature normalization)
- Can be scalable in practice
- Not too many hyperparameters (regularization)

CON

- Greedy sequential fit may not be globally optimal

IN PRACTICE

- XGBoost
  - Popular in many competitions and industrial applications