Ensemble

Big Data Lectures – Chapter 6

Lexin Li

Division of Biostatistics
University of California, Berkeley

Outline

- list of topics:
  - neural networks and deep learning
  - boosting
  - random forest
  - more ensemble methods

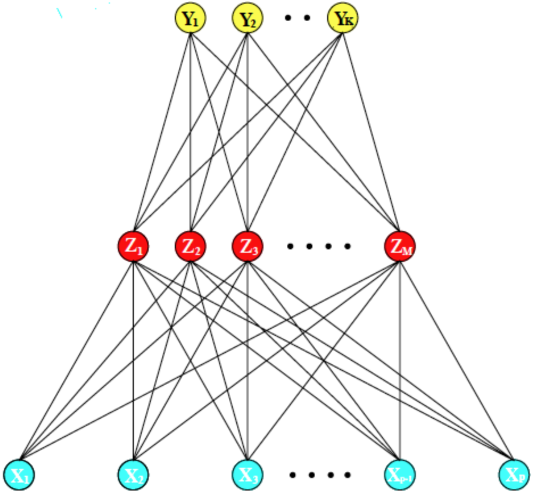
Neural Networks and Deep Learning

Neural networks

- neural networks:
  - the term has evolved to encompass a large class of models and learning methods, especially popular in the field of **artificial intelligence, computer vision, and automatic speech recognition**
  - we describe a widely used neural net, sometimes called the **single hidden layer back-propagation network**, or **single layer perceptron**
- examples:
  - a typical neural network used 1 to 10 million connections
  - Google Brain neural network used more than 1 billion connections developed a distributed computing infrastructure and spread the computation across 16,000 of our CPU cores

Neural networks

- key idea: sum of nonlinear functions of linear combinations of the inputs, typically represented by a network diagram



Neural networks

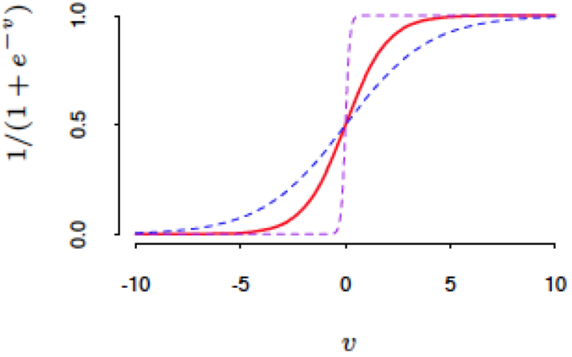
- neural networks model:

$$\begin{aligned} Z_m &= \sigma(\alpha_{0m} + \boldsymbol{\alpha}_m^\mathsf{T} \boldsymbol{X}), \quad m = 1, \ldots, M \\ T_k &= \beta_{0k} + \boldsymbol{\beta}_k^\mathsf{T} \boldsymbol{Z}, \quad k = 1, \ldots, K \\ f_k(\boldsymbol{X}) &= g_k(T), \quad k = 1, \ldots, K \end{aligned}$$

  - $\boldsymbol{Y} = (Y_1, \ldots, Y_K)$ are the K-dimensional output; e.g., for univariate response, $K = 1$; for K-class classification, kth unit models the probability of class k
  - $\boldsymbol{X} = (X_1, \ldots, X_p)$ are the p-dimensional input features
  - $\boldsymbol{Z} = (Z_1, \ldots, Z_M)$ are the **derived features** created from linear combinations of the inputs $\boldsymbol{X}$
  - $\boldsymbol{T} = (T_1, \ldots, T_K)$ are the output features that are directly associated with the outputs $\boldsymbol{Y}$ through output functions $g_k(\cdot)$
  - $g_k(T) = T$ for regression; $g_k(T) = e^{T_k} / \sum_{l=1}^K e^{T_l}$ for K-class classification

Neural networks

- activation function:
  - usually $\sigma(v) = \textit{sigmoid} = 1/(1+e^{-v})$
  - $\sigma(v) =$ a step function: human brain models where each unit represents a neuron, and the connections represent synapses; the neurons fired when the total signal passed to that unit exceeded a certain threshold



Neural networks

- loss function: residual sum of squares / deviance

$$L = \sum_{i=1}^{n} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2 , \qquad L = - \sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik} \log f_k(x_i)$$

- model fitting: back propagation – gradient descent
  - gradient descent update:

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{n} \frac{\partial R_i}{\partial \beta_{km}} \qquad \alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{n} \frac{\partial R_i}{\partial \alpha_{ml}}$$

where $R_i = \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2$

  - the derivatives: letting $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^{\mathsf{T}} x_i)$

$$\frac{\partial R_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i)) g_k'(\beta_k^{\mathsf{T}} z_i) z_{mi} \equiv \delta_{ki} z_{mi}$$

$$\frac{\partial R_i}{\partial \alpha_{ml}} = - \sum_{k=1}^{K} 2(y_{ik} - f_k(x_i)) g_k'(\beta_k^{\mathsf{T}} z_i) \beta_{km} \sigma'(\alpha_m^{\mathsf{T}} x_i) x_{il} \equiv s_{mi} x_{il}$$

Neural networks

- model fitting: back propagation (continued)
  - δ_{ki} and s_{mi} are the "errors" from the current model at the output and hidden layer units
  - back propagation equations

$$s_{mi} = \sigma'(\alpha_m^{\mathsf{T}} x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki}$$

  - two-pass updates:
    - forward pass: fix the current weights and compute the predicted values $\hat{f}_k(x_i)$
    - backward pass: compute the errors δ_{ki}, then back-propagate to compute the errors s_{mi}
    - update $\hat{\beta}_{km}$ and $\hat{\alpha}_{ml}$
  - γ_r is the learning rate
  - advantages: **simple and local** nature; each hidden unit passes and receives information only to and from units that share a connection; can be implemented efficiently on a parallel architecture computer

Neural networks

- practical issues:
  - starting values: usually starting values for weights are chosen to be random values near zero; hence the model starts out nearly linear, and becomes nonlinear as the weights increase
  - overfitting: early stopping; weight decay
  - scaling of the inputs: mean zero and standard deviation one
  - how many hidden units; how many hidden layers: guided by domain knowledge and experimentation
  - multiple minima: try with different starting values

Neural networks

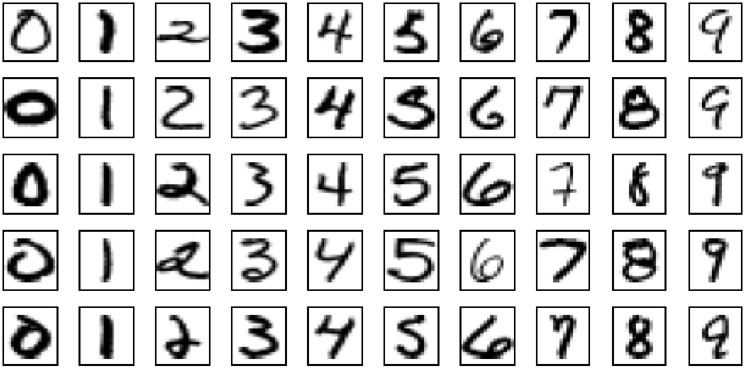
- practical issues:
  - starting values: usually starting values for weights are chosen to be random values near zero; hence the model starts out nearly linear, and becomes nonlinear as the weights increase
  - overfitting: early stopping; weight decay
  - scaling of the inputs: mean zero and standard deviation one
  - how many hidden units; how many hidden layers: guided by domain knowledge and experimentation
  - multiple minima: try with different starting values
- neural networks model is a **projection pursuit** type additive model:

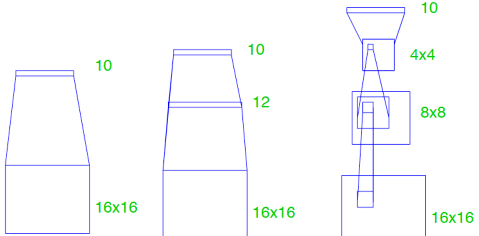
$$f(\boldsymbol{X}) = \beta_0 + \sum_{m=1}^{M} \beta_m \, \sigma(\alpha_{m0} + \boldsymbol{\alpha}_m^{\mathsf{T}} \boldsymbol{X})$$

- **view**: neural networks are not a fully automatic tool, as they are sometimes advertised; as with all statistical models, subject matter knowledge should and often be used to improve their performance

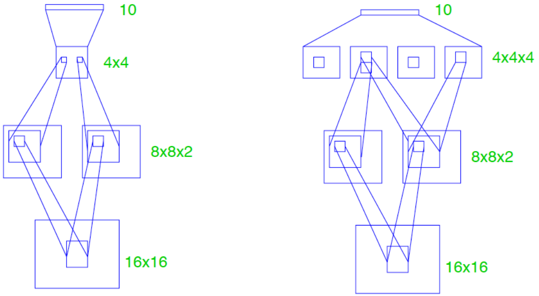
Example – handwritten digits

- ZIP code data:
  - input: 256 pixel values from 16×16 grayscale images; output: $0, 1, \ldots, 9$ 10-class classification
  - a modest experimental subset: 320 training digits and 160 testing digits





Net-1 Net-2 Net-3
Local Connectivity



Net-4 Shared Weights Net-5

Example – handwritten digits

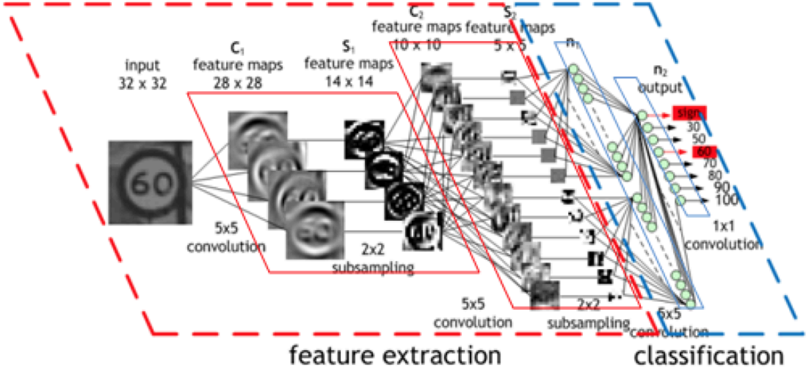
- neural networks used:
  - net-1: no hidden layer, equivalent to multinomial logistic regression
  - net-2: one hidden layer, 12 hidden units fully connected
  - net-3: two hidden layers locally connected
  - net-4: two hidden layers, locally connected with weight sharing
  - net-5: two hidden layers, locally connected, two levels of weight sharing (was the result of many person years of experimentation)

- results:

network	links	weights	accuracy
net 1	2570	2570	80.0%
net 2	3124	3214	87.0%
net 3	1226	1226	88.5%
net 4	2266	1131	94.0%
net 5	5194	1060	98.4%

Deep learning

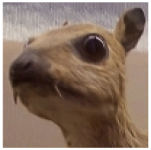
- deep learning is a class of machine learning algorithms that:
  - use a cascade of multiple layers of nonlinear processing units, and each successive layer uses the output from the previous layer as input



Deep learning

- **convolution**:
  - a mathematical operation that describes a rule of how to mix two functions or pieces of information
  - the feature map (the input data) + the convolution kernel $\Rightarrow$ a transformed feature map
  - convolution is often interpreted as a **filter**, where the kernel filters the feature map for information of a certain kind; e.g., one kernel might filter for edges and discard other information

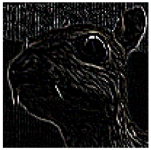
Input image

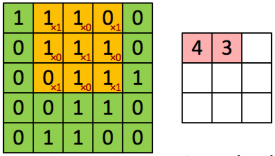


Convolution Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Feature map



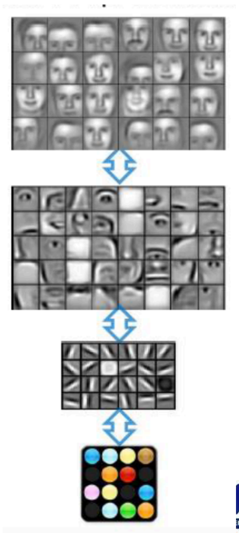


Image

Convolved Feature

Deep learning

- **hierarchical feature representation**:
  - form a hierarchical representation of the data, where higher level features are derived from lower level features
  - image recognition: pixel → edge → motif → part → object
  - speech: sample → spectral band → sound → phoneme → word
- feature engineering *vs* feature learning
  - **feature engineering**: extract useful features from the data for a specific learning task
  - **feature learning**: feature engineering **automatically done by algorithms** — the learnt features are usually difficult to interpret



Deep learning

- challenges of a deep network:
  - **computation** — hardware advancement, e.g., graphics processing unit (GPU)
  - **vanishing gradients**: the gradients became too small to provide a learning signal for very deep layers, thus making the deep architectures perform poorly when compared to shallow learning algorithms — pre-trained one level at a time through unsupervised learning, then further fine-tuned through supervised back-propagation, alternative activation functions, alternative network structures . . .
- deep architectures:
  - feed-forward: multilayer neural networks, convolutional neural networks
  - feed-back: stacked sparse coding, deconvolutional nets
  - bi-drectional: deep Boltzmann machines, stacked auto-encoders

Deep learning

- learning protocols:
  - purely supervised
    - initialize parameters randomly
    - train in supervised mode using back-propagation to compute gradients
    - used in most practical systems for speech and image recognition
  - unsupervised & layerwise + supervised classifier on top
    - train each layer unsupervised, one after the other
    - train a supervised classifier on top, keeping the other layers fixed
    - good when very few labeled samples are available
  - unsupervised & layerwise + global supervised fine-tuning
    - train each layer unsupervised, one after the other
    - add a classifier layer, and retrain the whole thing supervised
    - good when label set is poor

Boosting

Boosting

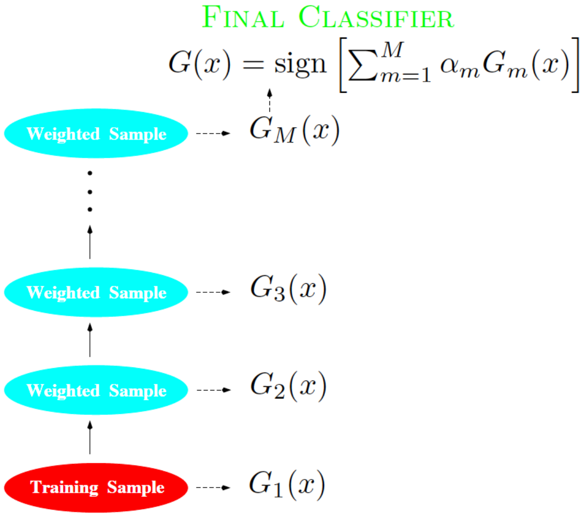
- introduction:
  - boosting is one of the most powerful learning ideas introduced in the last twenty years (early this century)
  - it was originally designed for classification problems, while it can profitably be extended to regression as well
  - motivation: combines the outputs of many "weak" classifiers (e.g., classification trees) to produce a powerful "committee"
- history and literatures
  - **AdaBoost**: the most popular boosting algorithm by Freund and Schapire (1997)
  - Breiman (NIPS Workshop, 1996) referred to AdaBoost with trees as the "best off-the-shelf classifier in the world"
  - Friedman et al. (2000) offered a statistical view of the AdaBoost: **a forward stagewise additive model with exponential loss**
  - Friedman (2001) developed **gradient boosting machine**
  - debate between Mease and Wyner (2008) and Friedman et al. (2008); review by Bühlmann and Hothorn (2007)

AdaBoost

- consider a two-class classification problem:
  - $Y \in \{-1, 1\}$, the classifier $G(\boldsymbol{x})$ has training error $\overline{\mathrm{err}} = \frac{1}{n}\sum_{i=1}^{n} I(y_i \neq G(\boldsymbol{x}_i))$
  - the expected error rate on future predictions is $E_{\boldsymbol{X},Y} I(Y \neq G(\boldsymbol{X}))$
- the procedure:
  - sequentially apply the weak classification algorithm to **repeatedly modified** versions of the data
  - produces a sequence of **weak classifiers** $G_m(\boldsymbol{x}), m = 1, 2, ..., M$
  - the predictions from G_m's are then combined through **a weighted majority vote** to produce the final prediction

$$G(\boldsymbol{x}) = \mathrm{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(\boldsymbol{x})\right)$$

- the weights $\alpha_1, ..., \alpha_M$ are computed by the boosting algorithm:
  - they weigh the contribution of each G_m
  - give higher influence to more accurate classifiers in the sequence


FINAL CLASSIFIER
$G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$
Weighted Sample
$G_M(x)$
Weighted Sample
$G_3(x)$
Weighted Sample
$G_2(x)$
Training Sample
$G_1(x)$

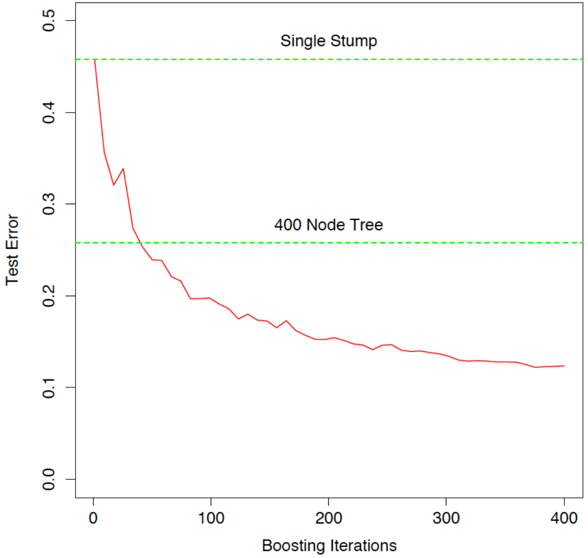
AdaBoost

- algorithm:

```
initialize: observation weights $w_i = 1/n, i = 1, ..., n$
**for** $m = 1, \dots, M$ **do**
    fit a classifier $G_m(\boldsymbol{x})$ to the training data using weights $w_i$
    compute the weighted error
        $$\mathrm{err}_m = \frac{\sum_{i=1}^n w_i I(y_i = G_m(\boldsymbol{x}_i))}{\sum_{i=1}^n w_i}$$
    compute the importance of $G_m$ as
        $$\alpha_m = \log(\frac{1 - \mathrm{err}_m}{\mathrm{err}_m})$$
    update $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(\boldsymbol{x}_i))], i = 1, ..., n$
**end for**
output $G(\boldsymbol{x}) = \mathrm{sign}\left(\sum_{m=1}^M \alpha_m G_m(\boldsymbol{x})\right)$
```

AdaBoost

- data modification in AdaBoost:
  - at each boosting step, apply weights $w_1, ..., w_n$ to the training observations $(\boldsymbol{x}_i, y_i), i = 1, ..., n$
  - initially, all of the weights are set to $w_i = 1/n$
  - at step $m = 2, 3, ..., M$, those observations misclassified by $G_{m-1}(\boldsymbol{x})$ have weights increased; observations correctively classified by $G_{m-1}(\boldsymbol{x})$ have weights decreased
  - observations that are difficult to correctly classify receive ever increasing influence
  - each successive classifier is forced to concentrate on those training observations that are misclassified by previous classifiers
- an illustrative example:
  - ten features $X_1, ..., X_{10} \sim N(0, 1)$
  - two classes: $Y = 2 \cdot I(\sum_{j=1}^{10} X_j^2 > \chi^2_{10}(0.5)) - 1$
  - training size $n = 2000$, test size $10,000$
  - weak classifier: stump (a two-terminal node classification tree)
  - misclassification error rate: stump 46%; 400-mode tree 26%; boosting 12.2%


Single Stump
400 Node Tree
Test Error
Boosting Iterations
0.0
0.1
0.2
0.3
0.4
0.5
0
100
200
300
400

Boosting as additive model

- boosting is an additive model:

$$G(\boldsymbol{x}) = \mathrm{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(\boldsymbol{x})\right).$$

  - fitting an additive expansion in a set of "basis" functions $G_m(\boldsymbol{x})$

- **additive models**:
more generally, basis function expansions take the form

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} \beta_m b(\boldsymbol{x}; \gamma_m)$$

  - β_m's are expansion coefficients
  - $b(\boldsymbol{x}, \gamma_m)$ are simple functions of $\boldsymbol{x}$, characterized by parameters γ_m

Boosting as additive model

- model estimation via a loss function:
  - squared-error or likelihood-based loss functions
  - typically, the model is obtained by minimizing the loss function averaged over the training data

$$\min_{\beta_m,\gamma_m{}_1^M} \sum_{i=1}^{n} L\left(y_i, \sum_{m=1}^{M} \beta_m b(\boldsymbol{x}_i;\gamma_m)\right)$$

  - it is feasible to rapidly solve the subproblem of fitting just a **single basis**

$$\min_{\beta,\gamma} \sum_{i=1}^{n} L\left(y_i, \beta b(\boldsymbol{x}_i;\gamma)\right)$$

- examples:
  - CART uses step basis functions, γ parametrizes the split variables and points at the internal nodes, and the predictions at the terminal nodes

Boosting as additive model

- forward *stagewise* additive modeling:
  - approximate the function by sequentially adding new basis functions to the expansion **without** adjusting the parameters and coefficients of those that have been added.
  - at iteration m, one solves for the optimal basis function $b(\boldsymbol{x}, \boldsymbol{\gamma}_m)$ and corresponding coefficient β_m to add to the current expansion $f_{m-1}(\boldsymbol{x})$
- algorithm:

```
initialize $f_0(\boldsymbol{x}) = 0$
for $m = 1, \ldots, M$ do
    compute
        $$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{n} L\left(y_i, f_{m-1}(\boldsymbol{x}_i) + \beta b(\boldsymbol{x}_i; \boldsymbol{\gamma})\right)$$
    set $f_m(x) = f_{m-1}(\boldsymbol{x}) + \beta_m b(\boldsymbol{x}; \boldsymbol{\gamma}_m)$
end for
```

AdaBoost – a statistical view

- what is AdaBoost in a statistical view:
  - AdaBoost builds an **additive logistic regression model**

$$f(\boldsymbol{x}) = \log \frac{\Pr(Y = 1|\boldsymbol{x})}{\Pr(Y = -1|\boldsymbol{x})} = \sum_{m=1}^{M} \beta_m G_m(\boldsymbol{x})$$

  - model fitting: **forward stagewise additive modeling**
  - loss function: the **exponential loss** function

$$L(y, f(\boldsymbol{x})) = \exp[-yf(\boldsymbol{x})]$$

  - the basis functions are individual classifiers $G_m(\boldsymbol{x}) \in \{-1, 1\}$.

- remarks:
  - the success of boosting is not that mysterious
  - why equivalent; how connects to a more familiar stat model; why exponential loss function

AdaBoost – a statistical view

- compared to slide-25:
  - one needs to solve (β_m, G_m) to minimize

$$\sum_{i=1}^{n} \exp[-y_i(f_{m-1}(\boldsymbol{x}_i) + \beta G(\boldsymbol{x}_i))] = \sum_{i=1}^{n} w_i^{(m)} \exp[-\beta y_i G(\boldsymbol{x}_i))]$$

  where $w_i^{(m)} = \exp[-y_i f_{m-1}(\boldsymbol{x}_i)]$

  - $w_i^{(m)}$ depends only on $f_{m-1}(\boldsymbol{x}_i)$, not on β or $G(\boldsymbol{x})$; weights
  - the solution is updated as $f_m(\boldsymbol{x}) = f_{(m-1)}(\boldsymbol{x}) + \beta_m G_m(\boldsymbol{x})$:

$$\begin{aligned} G_m &= \arg\min_G \sum_{i=1}^{n} w_i^{(m)} I(y_i \neq G(\boldsymbol{x}_i)) \\ \beta_m &= \frac{1}{2} \log \frac{1 - \mathrm{err}_m}{\mathrm{err}_m} = \frac{1}{2}\alpha_m \end{aligned}$$

  - the weights for the next iteration are:

$$\begin{aligned} w_i^{(m+1)} &= w_i^{(m)} \exp(-\beta_m y_i G_m(\boldsymbol{x})) \\ &= w_i^{(m)} \exp(\alpha_m I(y_i \neq G(\boldsymbol{x}_i))) \exp(-\beta_m) \end{aligned}$$

AdaBoost – a statistical view

- what AdaBoost actually estimates:

$$f^*(\boldsymbol{x}) = \arg\min_f E_{Y|\boldsymbol{x}}[e^{-Yf(\boldsymbol{x})}] = \frac{1}{2}\log\frac{\Pr(Y=1|\boldsymbol{x})}{\Pr(Y=-1|\boldsymbol{x})}$$

  - equivalently, $\Pr(Y=1|\boldsymbol{x}) = [1+\exp\{-2f^*(\boldsymbol{x})\}]^{-1}$
  - the additive expansion produced by AdaBoost is estimating one half of the log-odds of $\Pr(Y=1|\boldsymbol{x})$
- another loss criterion with the same population minimizer:
  - the binomial negative log-likelihood (deviance / cross-entropy)
  - let $Y' = (Y+1)/2 \in \{0,1\}$, then the binomial negative log-likelihood loss function is

$$\begin{aligned} -l(Y, p(\boldsymbol{x})) &= -[Y'\log p(\boldsymbol{x}) + (1-Y')\log(1-p(\boldsymbol{x}))] \\ &= \log(1+\exp(-2Yf(\boldsymbol{x}))) \end{aligned}$$

    where $p(\boldsymbol{x}) = [1+e^{-2f(\boldsymbol{x})}]^{-1}$
  - the population minimizers of the deviance $E_{Y|\boldsymbol{x}}[-l(Y, f(\boldsymbol{x})]$ and $E_{Y|\boldsymbol{x}}[e^{-Yf(\boldsymbol{x})}]$ are the same.

Gradient boosting machine

- functional optimization view:
  - boosting as an **optimization algorithm in functional space**
  - this interpretation opens the door to understand what boosting does and brings boosting from classification to regression, survival ...
- **gradient boosting machine**: functional gradient descent

```
initialize $f_0(\boldsymbol{x}) = 0$ or an offset value
for $m = 1, \ldots, M$ do
    compute the negative gradient and evaluate at $f_{m-1}(\boldsymbol{x}_i)$
        $$g_{mi} = \left[ -\frac{\partial L(y, f(\boldsymbol{x}))}{\partial f(\boldsymbol{x})} \right]_{f(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}_i)}$$
    fit $g_{mi}$ to $\boldsymbol{x}_i$ with a base learner: $\hat{g}_m(\boldsymbol{x}, \gamma_m) : \boldsymbol{x}_i \to g_{mi}$
    line search:
        $$\rho_m = \arg\min_{\rho} L(y, f_{m-1}(\boldsymbol{x}) + \rho \hat{g}_m(\boldsymbol{x}, \gamma_m))$$
    update $f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \nu \rho_m \hat{g}_m(\boldsymbol{x}, \gamma_m)$
end for
```

Gradient boosting machine

- gradient for some common loss functions:

algorithm	loss function	negative gradient
AdaBoost	$\exp[-yf(\boldsymbol{x})]$	$y\exp[-yf(\boldsymbol{x})]$
LogitBoost	$\log\{1+\exp[-2yf(\boldsymbol{x})]\}$	$2y/\{1+\exp[2yf(\boldsymbol{x})]\}$
L_2Boost	$\frac{1}{2}[y-f(\boldsymbol{x})]^2$	$y-f(\boldsymbol{x})$
LADBoost	$\lvert y-f(\boldsymbol{x})\rvert$	$\mathrm{sign}(y-f(\boldsymbol{x}))$

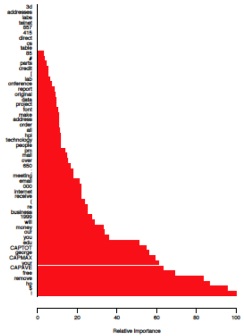
- base learner:
  - **component-wise** linear model: regression of g_{mi} on x_{ij} for $j=1,\dots,p$
  - **component-wise** smoothing splines
  - **trees**
- regularization:
  - **early stopping**: control the number of boosting iterations
  - **shrinkage**: a small shrinkage coefficient $0<\nu<1$
- theoretical investigation: Bühlmann and Yu (2003, JASA)

Gradient boosting machine

- interpretation: predictor relative importance

$$\left\{ E\left[\frac{\partial \hat{f}(\boldsymbol{X})}{\partial X_j}\right]^2 \operatorname{var}(X_j) \right\}^{1/2} \qquad j = 1, \ldots, p$$

- spam email data revisit:
    - boosting: 4.5%, GAM: 5.5%, CART: 8.7%



Random Forests

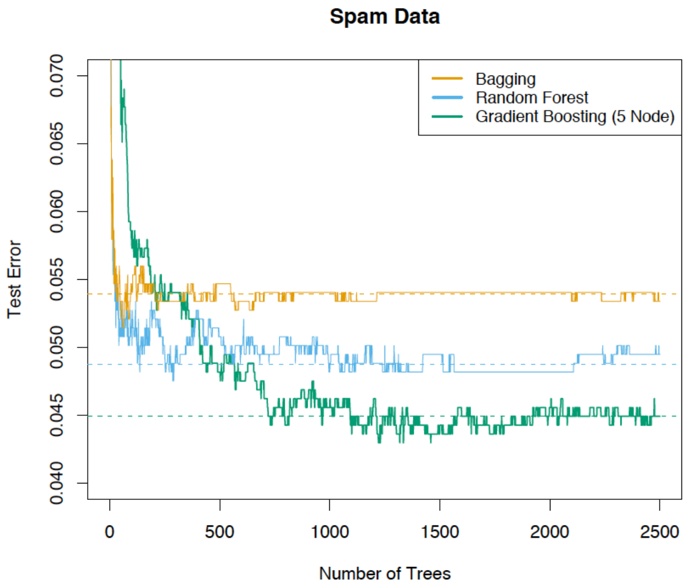
Random forests

- **bagging**:
  - an additive model / committee method for reducing the **variance** of an estimated prediction function
  - simply fit the same regression tree many times to bootstrap sampled versions of the training data, and average the result
  - an average of B **i.i.d.** random variables, each with variance σ^2, has variance $\frac{1}{B}\sigma^2$
  - an average of B **i.d.** random variables, with positive pairwise correlation ρ, has variance $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$
- boosting:
  - the committee of weak learners **evolves** over time
  - the members cast a **weighted** vote
- **random forest**:
  - a substantial modification of bagging that builds a large collection of **de-correlated** trees, and then averages them
  - bagging is often dominated by boosting, whereas random forests enjoy a similar performance as boosting, and are simpler to train and tune

Random forests

- algorithm:
  - for $b = 1$ to B:
    - draw a bootstrap sample of size n from the training data
    - grow a tree T_b to the bootstrapped data, by recursively repeating for each terminal node of the tree
      - select m variables at random from the p variables (de-correlation)
      - pick the best variable/split-point among them
      - split the node into two daughter nodes
  - output the ensemble of trees $\{T_b\}_1^B$
  - regression: $\hat{f}_B(\boldsymbol{x}) = \frac{1}{B}\sum_{b=1}^{B} T_b(\boldsymbol{x})$
  - classification: $\hat{C}_B(\boldsymbol{x}) =$ majority vote $\{\hat{C}_b(T_b(\boldsymbol{x}))\}_1^B$

- remarks:
  - correlation reduction is achieved in the tree-growing process through random selection of the input variables
  - **not all** estimators can be improved by shaking up the data like this; it seems that highly nonlinear estimators, such as trees, benefit the most


Spam Data
Bagging
Random Forest
Gradient Boosting (5 Node)
Test Error
0.040
0.045
0.050
0.055
0.060
0.065
0.070
0
500
1000
1500
2000
2500
Number of Trees

Ensemble Methods

Ensemble methods

- introduction:
  - build a prediction model by combining (the strengths of) a collection of base models
  - two tasks: developing a number of base learners from the training data + combining them to form the composite predictor
- learning the ensemble:
  - lasso-based approach:

$$\min_{\alpha} \sum_{i=1}^{n} L[y_i, \alpha_0 + \sum_{m=1}^{M} \alpha_m T_m(\boldsymbol{x}_i)] + \lambda \sum_{m=1}^{M} |\alpha_m|$$

  - super learner

Additional readings

- Hastie, T., Tibshirani, R., and Friedman, J. (2001). *Elements of Statistical Learning*. Springer. Chapters 9, 10, 11, 15, 16
- Friedman, J., Hastie, T. and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (with discussion), *Annals of Statistics*, **28**, 337-307
- Friedman, J. (2001). Greedy function approximation: A gradient boosting machine, *Annals of Statistics*, **29**, 1189-1232
- Bühlmann, P. and Hothorn, T. (2007). Boosting algorithms: regularization, prediction and model fitting (with discussion), *Statistical Science*, **22**, 477-505
- Mease, D. and Wyner, A. (2008). Evidence contrary to the statistical view of boosting (with discussion), *Journal of Machine Learning Research*, **9**, 131-156
- Le, Q., Ranzato, M.A., Monga, R., Devin, M., Chen, K., Corrado, G., Jeffrey Dean, Andrew Ng. (2012). Building high-level features using large scale unsupervised learning. *Proceedings of the 29th International Conference on Machine Learning*.
- LeCun, Y. (2013). Deep Learning Tutorial. *ICML 2013*
- Hinton, G., Bengio Y., and LeCun, Y. (2015). Deep Learning Tutorial. *NIPS 2015*.
- Dettmers, T. (2015). Deep Learning in a Nutshell.