San José State University

Math 251: Statistical and Machine Learning Classification

Classification Trees and Ensemble Learning

Dr. Guangliang Chen

Outline

- Classification trees
- Ensemble learning
  - Bagging
  - Random forest
  - Boosting (AdaBoost, LogitBoost)
- Assignment 5

Main references

- "Trees, Bagging, Random Forests and Boosting", lecture slides by Trevor Hastie of Stanford University[1]
- "Trees and Random Forests", lecture slides by Adele Cutler of Utah State University[2]
- "Boosting", lecture notes by Kilian Weinberger of Cornell University[3]
- Chapter 8 of Textbook 1

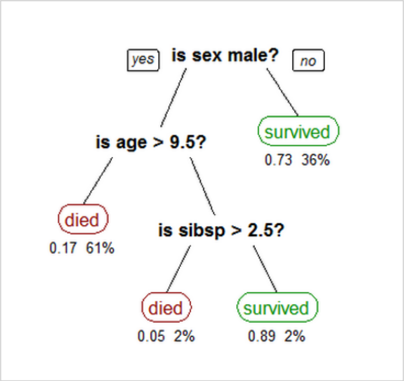
[1] http://jessica2.msri.org/attachments/10778/10778-boost.pdf

[2] http://www.math.usu.edu/adele/randomforests/uofu2013.pdf

[3] https://www.cs.cornell.edu/courses/cs4780/2021fa/lectures/lecturenote19.html

What is a classification tree?

- A series of binary splits based on training data.
- Each internal node represents a query on one of the variables.
- The terminal/leaf nodes are the decision nodes, typically dominated by one of the classes.
- New observations are classified in the respective terminal nodes through majority vote.



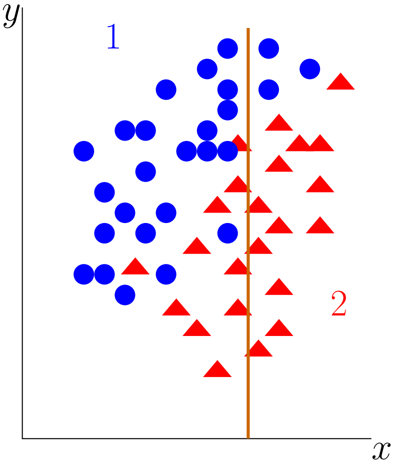
A tree showing survival of passengers on the Titanic ("sibsp" is the number of spouses or siblings aboard). The figures under the leaves show the probability of survival and the percentage of observations in the leaf.

Splitting criterion: Gini impurity score

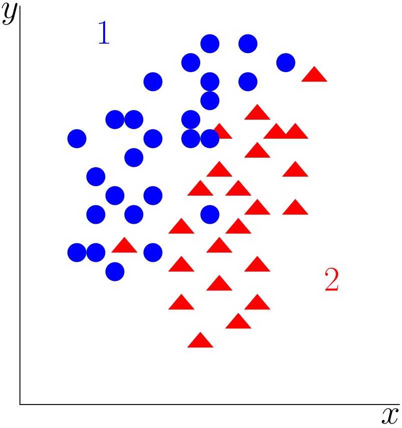
Optimal splitting in each code is found by minimizing the **Gini diversity index** over all possible splits (variable + cutoff):

$$n^{(\mathrm{L})}\sum_{j=1}^{c} p_j^{(\mathrm{L})}(1-p_j^{(\mathrm{L})})+n^{(\mathrm{R})}\sum_{j=1}^{c} p_j^{(\mathrm{R})}(1-p_j^{(\mathrm{R})})$$

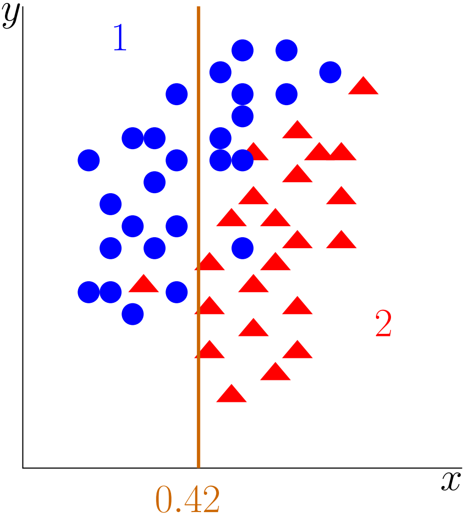
- $n^{(\mathrm{L})}$ ($n^{(\mathrm{R})}$): #training examples in left (right) node
- $p_i^{(\mathrm{L})}$ ($p_i^{(\mathrm{R})}$): proportion of class i in left (right) node

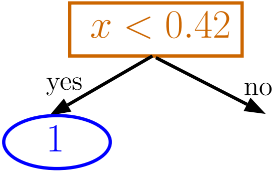


Demo: Building a classification tree

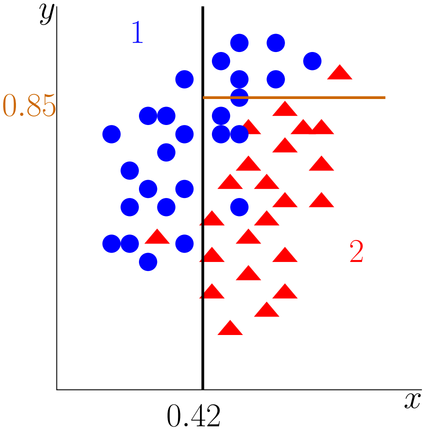


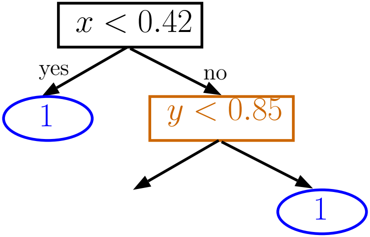
Classification trees and ensemble learning



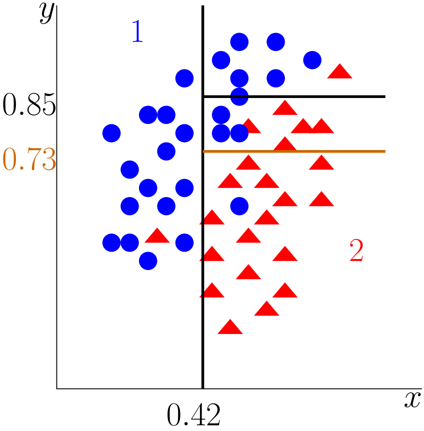


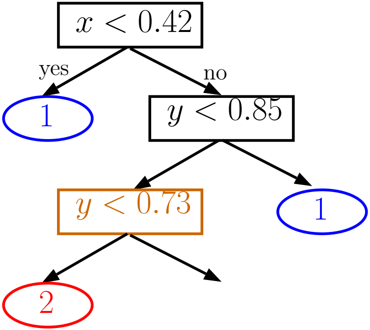
Classification trees and ensemble learning



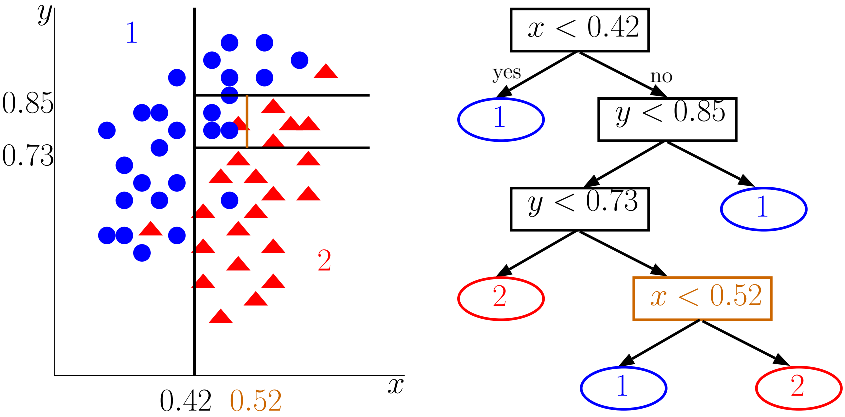


Classification trees and ensemble learning

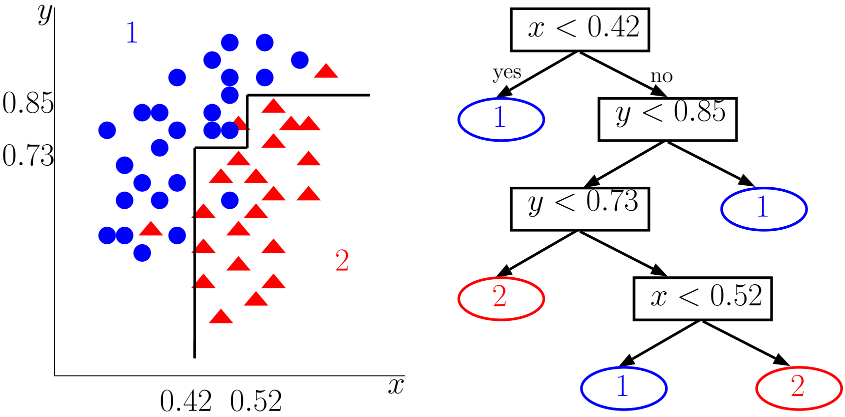




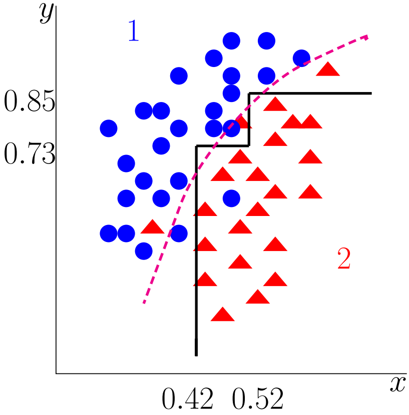
Classification trees and ensemble learning

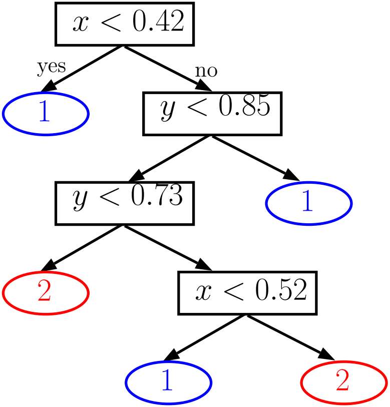


Classification trees and ensemble learning



Classification trees and ensemble learning

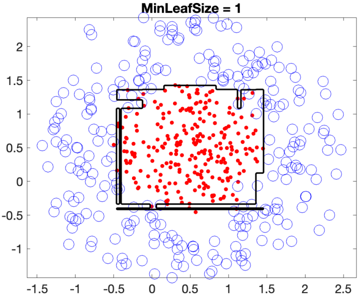


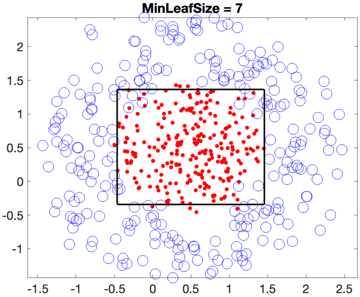


MATLAB function (with default values)

```
mdl = fitctree(trainX, trainLabels,
      ‘MinParentSize’, 10,
      ‘MinLeafSize’ , 1,
      ‘MaxNumSplits’ , n − 1,
      ‘NumVariablesToSample’ , ‘all’)
view(mdl, 'mode', 'graph') % to visualize the tree
pred = predict(mdl, Xtst);
```

Demonstration





Remark. Classification trees, though easy to build, are considered as weak learners:

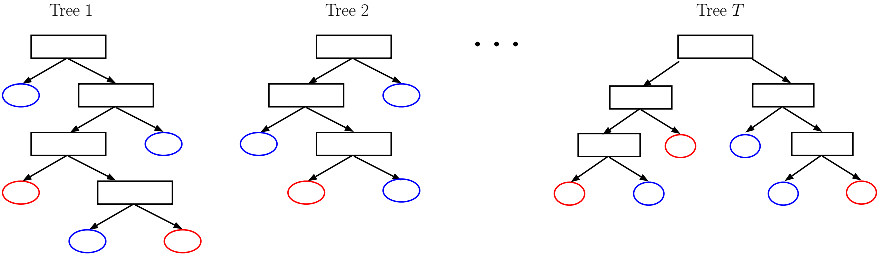
- The decision boundary is piecewise linear
- Unstable (if we change the data a little, the tree may change a lot)
- Prediction performance is often poor (due to high variance)

Advantages of classification trees

- No distribution assumptions (non-parametric)
- Works in the same way in multiclass settings
- Can handle large data sets
- Can handle categorical variables naturally
- Can deal with missing values elegantly

Ensemble methods

Ensemble methods train many weak classifiers (e.g., trees) and combine their predictions to enhance the performance of a single weak learner:

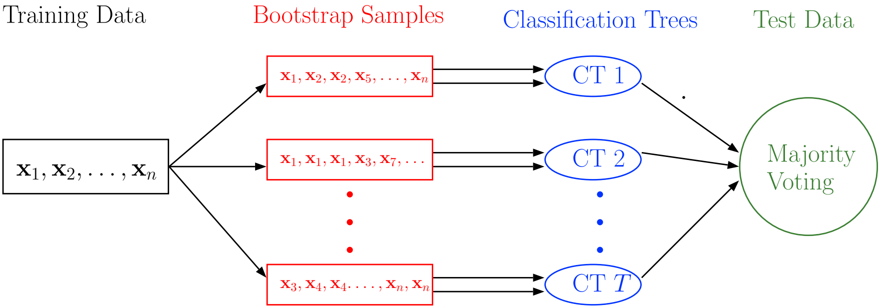


To be covered in this course:

- **Bagging (Bootstrap AGGregatING)**: build many trees independently from different bootstrap samples of the training data and then vote their predictions to get a final prediction
- **Random Forest**: a variant of bagging by allowing to use different subsets of variables at the nodes of any tree in the ensemble
- **Boosting**: build many trees adaptively and then add their predictions
  - AdaBoost (Adaptive Boosting)
  - GradientBoost (Gradient boosting)

Bagging

To build a classification tree on each separate **bootstrap sample** of the training data, i.e., a **random sample with replacement**, and then use majority vote.


Training Data
Bootstrap Samples
Classification Trees
Test Data
$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$
$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_2, \mathbf{x}_5, \ldots, \mathbf{x}_n$
$\mathbf{x}_1, \mathbf{x}_1, \mathbf{x}_1, \mathbf{x}_3, \mathbf{x}_7, \ldots$
$\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_4. \ldots, \mathbf{x}_n, \mathbf{x}_n$
CT 1
CT 2
CT T
Majority Voting

Remark.

- "Averaging" many trees decreases the variance of the model, without increasing the bias (as long as the trees are not correlated)
- Bootstrap sampling is a way of de-correlating the trees (as simply training many trees on a single training set would give strongly correlated trees).
- The number of bootstrap samples/trees, T, is a free parameter. Typically, a few hundred to several thousand trees are used, depending on the size and nature of the training set.

Out-of-bag (oob)

It turns out that drawing n out of n observations with replacement omits on average **36.8%** of observations for each tree:

$$\left(1-\frac{1}{n}\right)^n \approx \frac{1}{e} = 0.3678794$$

We say that those samples left out by a tree are **out-of-bag (oob)** with respect to that tree.

classification trees

data	1	2	3	$\cdots$	T
$\mathbf{x}_1$	*	***	oob		oob
$\mathbf{x}_2$	**	oob	*		oob
$\mathbf{x}_3$	oob	*	oob		*
$\mathbf{x}_4$	oob	oob	**		**
$\mathbf{x}_5$	*	oob	oob		oob
$\mathbf{x}_6$	oob	oob	***		*
$\mathbf{x}_7$	oob	*	*		***
$\vdots$					
$\mathbf{x}_n$	*	oob	*		**

* selected by the tree

Out-of-bag error

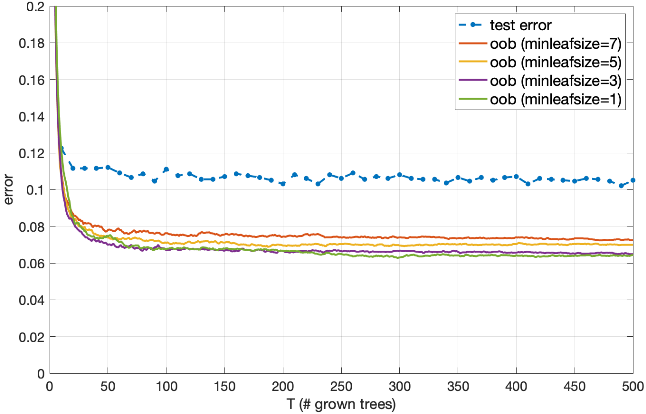
For each training example $(\mathbf{x}_i, y_i)$, we may vote the predictions of all the oob trees to obtain an estimate of the true label y_i, denoted as $\hat{y}_i^{(\mathrm{oob})}$. The overall oob error is computed as follows:

$$e^{(\mathrm{oob})}(T) = \sum_{i=1}^{n} 1_{y_i \neq \hat{y}_i^{(\mathrm{oob})}}$$

Such measure is an unbiased estimator of the true ensemble error and it does not require an independent validation dataset for evaluating the predictive power of the model.

An optimal value of T can be found by observing the out-of-bag error.

OOB and test errors of bagging on the *usps* data set



MATLAB function for bagging

```
% Train T trees and make oob predictions in the meantime
TB = TreeBagger(T, Xtr, ytr, 'OOBPrediction', 'on', 'NumPredictorsToSample', 'all', 'MinLeafSize', 1);

% Plot out-of-bag errors
figure; plot(oobError(TB))

pred = predict(TB, testX); % make prediction
pred = cellfun(@str2num, pred); % convert string cell output to vector
```

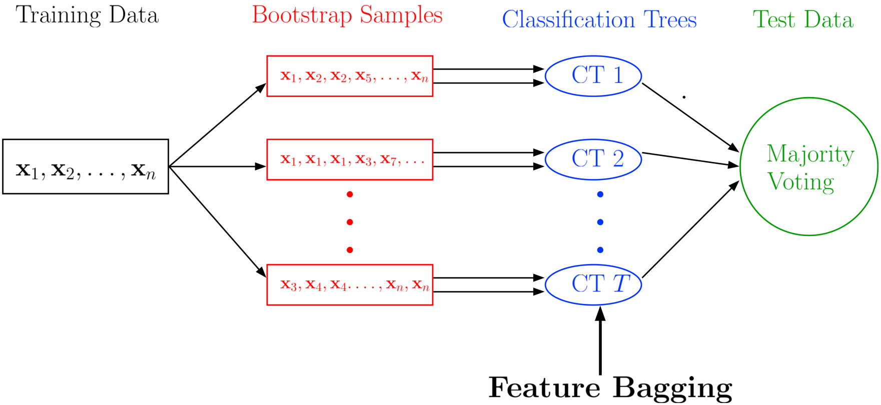
Random forest

Random forests improve bagging by allowing every tree in the ensemble to randomly select predictors for each of its nodes. This process is sometimes called "**feature bagging**".

The motivation is to further de-correlate the trees in bagging: If one or a few features are very strong predictors for the class label, these features will be selected in many of the trees, causing them to become correlated.

Typically, for a classification problem with d features, $\sqrt{d}$ features (selected at random) are used in each split.

Classification trees and ensemble learning



Training Data

Bootstrap Samples

Classification Trees

Test Data

$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$

$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_2, \mathbf{x}_5, \ldots, \mathbf{x}_n$

$\mathbf{x}_1, \mathbf{x}_1, \mathbf{x}_1, \mathbf{x}_3, \mathbf{x}_7, \ldots$

$\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_4 . \ldots, \mathbf{x}_n, \mathbf{x}_n$

CT 1

CT 2

CT T

Majority Voting

Feature Bagging

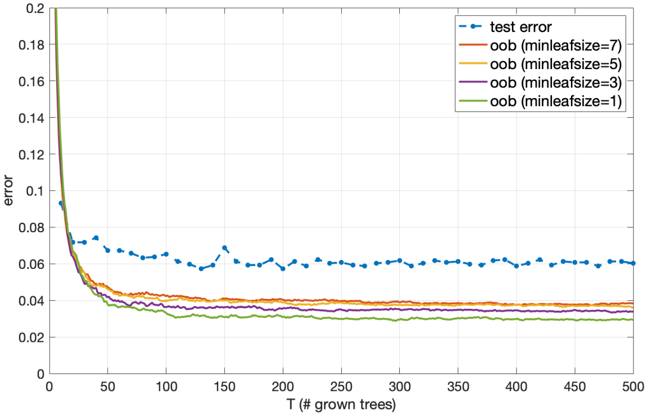
MATLAB function for random forest

% Train T trees with feature bagging (s) + oob prediction
TB = TreeBagger(T, Xtr, ytr, 'OOBPrediction', 'on', 'NumPredictorsToSample', s, 'MinLeafSize', 1);
% Default value of 'NumPredictorsToSample' $= \sqrt{d}$, so it is already random forest unless you set s = ‘all’

figure; plot(oobError(TB))

pred = predict(TB, testX);
pred = cellfun(@str2num, pred); % convert string cell to vector

OOB and test errors of random forest on the *usps* data set



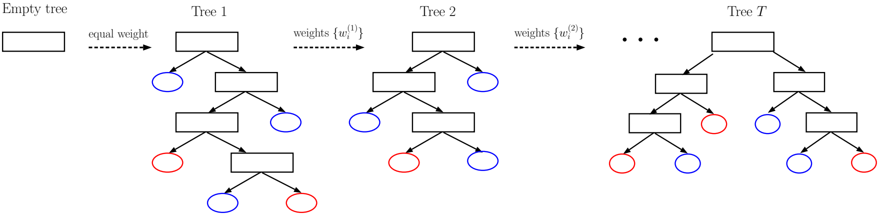
Python function for random forest

See documentation at

`http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`

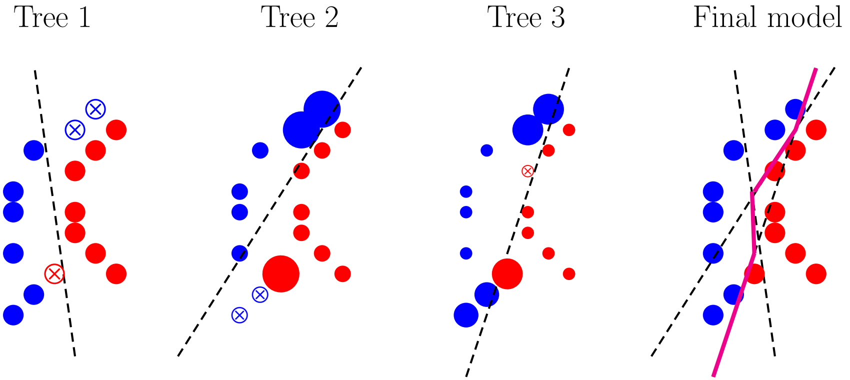
AdaBoost (Adaptive Boosting)

Main idea: Build tree classifiers sequentially by“ focusing more attention” on training errors made by the preceding trees.



One way to realize this idea is to use the output of the current ensemble to **reweight the training data** for the next tree.

Demo: an ensemble of 3 boosted trees



How to choose weights

Initially, all training examples have an equal weight, i.e., $w_i^{(0)} = \frac{1}{n}, \forall\, i$.

For each $t = 1 : T$, do the following sequentially:

- Fit a tree classifier $C_t(\mathbf{x})$ to the training data using weights $w_i^{(t-1)}$ from previous iteration
- Compute the weighted error of $C_t(\mathbf{x})$:

$$\epsilon_t = \sum_{i=1}^{n} w_i^{(t-1)} I_{y_i \neq C_t(\mathbf{x}_i)}$$

Classification trees and ensemble learning

- Set $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$, which is weight to be assigned to C_t

- Modify the weights of the training data points as follows:

$$w_i^{(t)} = w_i^{(t-1)} \cdot e^{-\alpha_t y_i C_t(\mathbf{x}_i)} = \begin{cases} w_i^{(t-1)} e^{-\alpha_t}, & C_t(\mathbf{x}_i) = y_i \\ w_i^{(t-1)} e^{\alpha_t}, & C_t(\mathbf{x}_i) \neq y_i \end{cases}$$

- Re-normalize the new weights $w_i^{(t)}$ to have a sum of 1 for the next iteration.

Remark. The classifier in the procedure does not need to be a decision tree (they can be kNN, or logistic regression instead)

How to use weights

Consider a weighted training set $(\mathbf{x}_i, y_i, w_i^{(t)}), 1 \le i \le n, t \ge 0$. In all calculations (wherever used), every training point $\mathbf{x}_i$ will count as "$w_i^{(t)}$ points".

For example, when computing the Gini impurity for a candidate split

$$n^{(\mathrm{L})} \sum_{j=1}^{k} p_j^{(\mathrm{L})}(1 - p_j^{(\mathrm{L})}) + n^{(\mathrm{R})} \sum_{j=1}^{k} p_j^{(\mathrm{R})}(1 - p_j^{(\mathrm{R})})$$

we will use instead

$$n^{(\mathrm{L})} = \sum_{i \in \text{left node}} w_i^{(t)} \qquad \text{and} \quad p_j^{(\mathrm{L})} = \frac{1}{n^{(\mathrm{L})}} \sum_{i \in (\text{class } j \cap \text{left node})} w_i^{(t)}$$

How to combine the trees

Classifying a test point $\mathbf{x}_0$ is through voting with weights

$$\alpha_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$$

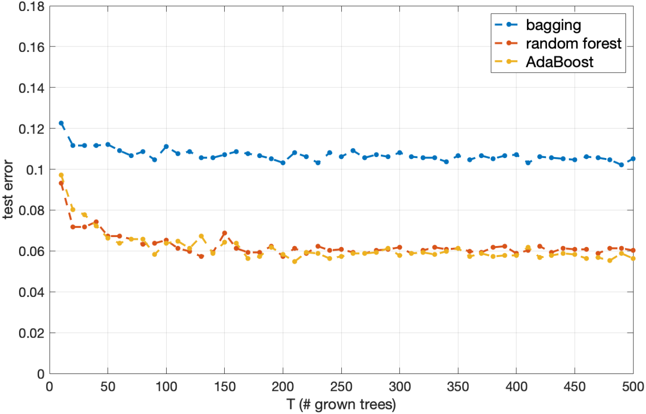
for the different trees:

$$\hat{y}_0 = \operatorname{argmax}_j \sum_{t\,:\,C_t(\mathbf{x}_0)=j} \alpha_t$$

Remark. For binary classification with labels $y_i = \pm 1$, the above decision rule can be written as

$$\hat{y}_0 = \operatorname{sgn}\left(\sum_{t=1}^{T} \alpha_t C_t(\mathbf{x}_0)\right)$$

Test errors of bagging, random forest, and AdaBoost on the *usps* data set



Mathematical interpretation of AdaBoost

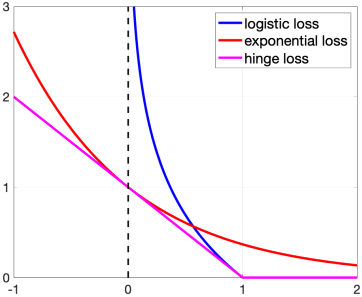
AdaBoost is a member of the gradient boosting family of classifiers. It uses the exponential loss.

In the setting of binary labels, i.e., $y = \pm 1$, let f be a binary classifier or a continuous function.

The exponential loss of f at a given location $\mathbf{x}$ is

$$\ell(y, \hat{y}) = e^{-y\hat{y}}, \quad \hat{y} = f(\mathbf{x}).$$

3 loss functions when $y = 1$:



The total loss of f on a training set $\{(\mathbf{x}_i, y_i)\}_{1\le i\le n}$, where $y_i = \pm 1$, is

$$\ell_n(f) = \sum_{i=1}^n \ell(y_i, \hat{y}_i) = \sum_{i=1}^n e^{-y_i \hat{y}_i}, \quad \hat{y}_i = f(\mathbf{x}_i)$$

It is an estimate of the expected loss at a random location $(\vec{X}, Y)$:

$$\mathrm{E}(\ell(Y, f(\vec{X}))) = \mathrm{E}\left(e^{-Yf(\vec{X})}\right) \quad \leftarrow \quad \frac{1}{n}\ell_n(f)$$

It can be shown that the minimizer of this loss is

$$\arg\min_f \mathrm{E}\left(e^{-Yf(\vec{X})}\right) \quad \rightarrow \quad f(\vec{X}) = \frac{1}{2}\log\frac{P(Y=1\mid\vec{X})}{P(Y=-1\mid\vec{X})}$$

To see this, write

$$\mathrm{E}\left(e^{-Yf(\vec{X})}\right) = \mathrm{E}_{\vec{X}}\left[\mathrm{E}_Y\left(e^{-Yf(\vec{X})} \mid \vec{X}\right)\right]$$

Next,

$$\begin{aligned}\mathrm{E}_Y\left(e^{-Yf(\vec{X})} \mid \vec{X}=\mathbf{x}\right) &= \mathrm{E}_Y\left(e^{-Yf(\mathbf{x})} \mid \mathbf{x}\right)\\ &= e^{-f(\mathbf{x})}P(Y=1\mid\mathbf{x}) + e^{f(\mathbf{x})}P(Y=-1\mid\mathbf{x}).\end{aligned}$$

Differentiating this with respect to $f(\mathbf{x})$ and setting it to zero gives that

$$-e^{-f(\mathbf{x})}P(Y=1\mid\mathbf{x}) + e^{f(\mathbf{x})}P(Y=-1\mid\mathbf{x}) = 0$$

The solution of the equation (in $f(\mathbf{x})$) is

$$f(\mathbf{x}) = \frac{1}{2}\log\frac{P(Y=1\mid\mathbf{x})}{P(Y=-1\mid\mathbf{x})}$$

This shows that the minimizer of the expected exponential loss is half of the log-odds function at each location.

Mathematical derivation of AdaBoost

Recall that the decision rule of the AdaBoost classifier is

$$y_0 = \operatorname{sgn}(f(\mathbf{x}_0)), \quad f(\mathbf{x}_0) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}_0)$$

where h_t is the classification tree built in iteration t, with weight α_t.

The decision function $f(\mathbf{x})$ is trained iteratively using gradient descent, based on the exponential loss. The procedure is outlined next.

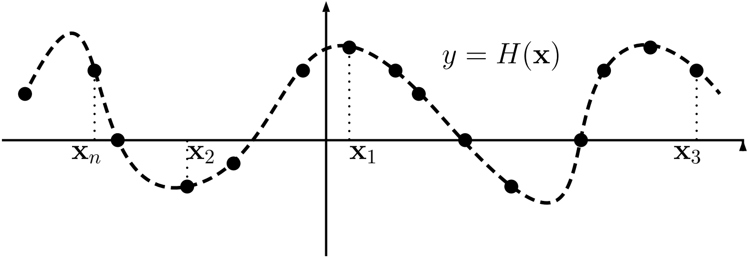
Suppose $H = H_t$ is the current ensemble learner (at iteration t),

$$H(\mathbf{x}) = \sum_{j=1}^{t} \alpha_j h_j(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^d$$

The loss of H on the training data is

$$\ell_n(H) = \sum_{i=1}^{n} \ell\left(y_i, H(\mathbf{x}_i)\right) = \sum_{i=1}^{n} e^{-y_i H(\mathbf{x}_i)}$$

which can be regarded as a function of $(H(\mathbf{x}_1), \ldots, H(\mathbf{x}_n))$.



In the next iteration, we want to add one more weak learner $h = h_{t+1}$ of the same kind, with weight $\alpha = \alpha_{t+1}$, so that the new ensemble learner $H + \alpha h$ will improve H in terms of the total loss, i.e.,

$$\ell_n(H + \alpha h) < \ell_n(H)$$

To choose h and α cleverly, we need to compute the gradient of $\ell_n(H)$ with respect to $(H(\mathbf{x}_1), \ldots, H(\mathbf{x}_n))$, i.e.,

$$\frac{\partial \ell_n(H)}{\partial H} \equiv \nabla \ell_n(H) = \left(\frac{\partial \ell_n(H)}{\partial H(\mathbf{x}_1)}, \ldots, \frac{\partial \ell_n(H)}{\partial H(\mathbf{x}_n)} \right),$$

in order to carry out gradient descent in the functional space.

Gradient boosting in general

Assumption: Suppose that we are given

- a space $\mathbb{H}$ of weak learners (e.g., classification trees) that have large bias and high training error, and
- a convex and differentiable loss function ℓ.

Question (Kearns, 1988): Can weak learners from $\mathbb{H}$ be combined to generate a strong learner with low bias?

Answer (Schapire, 1990): Yes

Solution: Create an ensemble learner iteratively:

$$f(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}), \quad \alpha_t \in \mathbb{R},\ h_t \in \mathbb{H}$$

That is, letting

$$H = \sum_{j=1}^{t} \alpha_j h_j(\mathbf{x})$$

be the current ensemble learner (after t iterations), we want to add in iteration $t+1$ one more weak learner h (with certain weight α) to the ensemble as follows:

$$\min_{h \in \mathbb{H}} \ell_n(H + \alpha h)$$

However, it may be too difficult to exactly solve the minimization problem.

So we instead seek a weak learner $h \in \mathbb{H}$ to simply reduce the loss a bit, i.e.,

$$\ell_n(H + \alpha h) < \ell_n(H)$$

If α is sufficiently small, then using Taylor's expansion we get

$$\ell_n(H + \alpha h) \approx \ell_n(H) + \alpha \langle \nabla \ell_n(H), h \rangle$$

Thus, to decrease the loss, we just need to select h such that

$$\langle \nabla \ell_n(H), h \rangle < 0$$

or equivalently,

$$\sum_{i=1}^{n} \frac{\partial \ell_n(H)}{\partial H(\mathbf{x}_i)}\, h(\mathbf{x}_i) < 0$$

Letting

$$r_i = \frac{\partial \ell_n(H)}{\partial H(\mathbf{x}_i)},$$

this condition can be rewritten as

$$\sum_{i=1}^{n} r_i\, h(\mathbf{x}_i) < 0.$$

Note that the choice of h does not need to be great. As long as the sum is negative, h would work.

Back to AdaBoost

Recall the setting:

- Binary labels: $y_i = \pm 1$, $i = 1, \ldots, n$
- Weak learners: $\mathbb{H} =$ space of binary classification trees, $h(\mathbf{x}) = \pm 1$
- Exponential loss function:

$$\ell_n(H) = \sum_{i=1}^{n} e^{-y_i H(\mathbf{x}_i)}$$

where H is the current ensemble learner at iteration t.

To find the next weaker learner $h \in \mathbb{H}$, we compute the components of the gradient $\nabla \ell_n(H)$:

$$r_i = \frac{\partial \ell_n(H)}{\partial H(\mathbf{x}_i)} = -y_i e^{-y_i H(\mathbf{x}_i)}$$

Denote

$$w_i = \frac{1}{Z} e^{-y_i H(\mathbf{x}_i)}, \quad 1 \le i \le n$$

where

$$Z = \sum_{i=1}^{n} e^{-y_i H(\mathbf{x}_i)} = \ell_n(H) \quad \text{such that } \sum_{i=1}^{n} w_i = 1$$

It follows that

$$r_i = -y_i w_i Z.$$

Now,

$$\sum_{i=1}^{n} r_i\, h(\mathbf{x}_i) = -Z \sum_{i=1}^{n} w_i y_i h(\mathbf{x}_i) = -Z \left[\sum_{h(\mathbf{x}_i)=y_i} w_i - \sum_{h(\mathbf{x}_i)\neq y_i} w_i \right]$$

$$= -Z \left[1 - 2 \sum_{h(\mathbf{x}_i)\neq y_i} w_i \right] = Z \left[2 \sum_{h(\mathbf{x}_i)\neq y_i} w_i - 1 \right]$$

In order to make progress (i.e., $\sum_{i=1}^{n} r_i\, h(\mathbf{x}_i) < 0$), we just need the new weak learner h to achieve a weighted classification error (better than random guessing):

$$\epsilon = \sum_{h(\mathbf{x}_i)\neq y_i} w_i < 0.5$$

After we have found the new weak leaner h achieving $\epsilon < 0.5$, we can select the optimal step size α as follows:

$$\min_{\alpha} \ell_n(H + \alpha h)$$

Write

$$\begin{aligned}
\ell_n(H + \alpha h) &= \sum_{i=1}^{n} e^{-y_i[H(\mathbf{x}_i) + \alpha h(\mathbf{x}_i)]} = Z \sum_{i=1}^{n} w_i e^{-\alpha y_i h(\mathbf{x}_i)} \\
&= Z \left[\sum_{h(\mathbf{x}_i) = y_i} w_i e^{-\alpha} + \sum_{h(\mathbf{x}_i) \neq y_i} w_i e^{\alpha} \right] \\
&= Z \left[e^{-\alpha}(1 - \epsilon) + e^{\alpha} \epsilon \right]
\end{aligned}$$

Differentiating it with respect to α and setting it equal to zero gives that

$$0 = Z\left[-e^{-\alpha}(1-\epsilon) + e^{\alpha}\epsilon\right]$$

The solution is

$$\alpha = \frac{1}{2}\log\frac{1-\epsilon}{\epsilon}$$

Lastly, after obtaining the new ensemble learner $H + \alpha h$, we need to re-compute and re-normalize the weights:

$$w_i' = e^{-y_i[H(\mathbf{x}_i)+\alpha h(\mathbf{x}_i)]} = w_i e^{-\alpha y_i h(\mathbf{x}_i)} = \begin{cases} w_i e^{-\alpha}, & h(\mathbf{x}_i) = y_i \\ w_i e^{\alpha}, & h(\mathbf{x}_i) \neq y_i \end{cases}$$

The new normalizing constant is

$$\begin{aligned} Z' &= \sum_{h(\mathbf{x}_i)=y_i} w_i e^{-\alpha} + \sum_{h(\mathbf{x}_i)\neq y_i} w_i e^{\alpha} \\ &= e^{-\alpha}(1-\epsilon) + e^{\alpha}\epsilon \\ &= \left(\frac{1-\epsilon}{\epsilon}\right)^{-1/2}(1-\epsilon) + \left(\frac{1-\epsilon}{\epsilon}\right)^{1/2}\epsilon \\ &= 2\sqrt{\epsilon(1-\epsilon)}. \end{aligned}$$

Thus, the new normalized weights are

$$w_i' = \frac{w_i e^{-\alpha y_i h(\mathbf{x}_i)}}{2\sqrt{\epsilon(1-\epsilon)}} = \begin{cases} w_i \cdot \frac{1}{2(1-\epsilon)}, & h(\mathbf{x}_i) = y_i \\ w_i \cdot \frac{1}{2\epsilon}, & h(\mathbf{x}_i) \neq y_i \end{cases}$$

Some comments on AdaBoost

- Given any weak learner better than random guess (i.e., error rate < 0.5), AdaBoost can achieve an arbitrarily high accuracy on the training data.
- Fast convergence
- Excellent generalization performance
- Can handle many features easily
- In general, AdaBoost $\succ$ random forest $\succ$ bagging $\succ$ single tree

LogitBoost (adaptive logistic regression)

AdaBoost is gradient boosting coupled with the exponential loss.

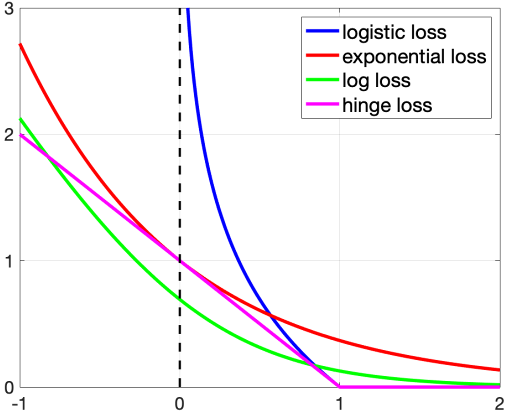
There are lots of other variants for gradient boosting such as LogitBoost for binary classification with labels $y = \pm 1$.

It works similarly to AdaBoost, expect it minimizes binomial deviance, also called log loss:

$$\ell(y, \hat{y}) = \log\left(1 + e^{-2y\hat{y}}\right)$$

where $\hat{y}$ is the output of a classifier or continuous function.

When $y = 1$:


3
2
1
0
-1
0
1
2
logistic loss
exponential loss
log loss
hinge loss

Why do the two loss functions (exponential and log) have so similar graphs?

This can be inferred from the Taylor series of $\log(1+x)$:

$$\log(1+x) = x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \cdots$$

When x is close to 0, we have

$$\log(1+x) \approx x$$

Thus, when the exponential loss $e^{-y\hat{y}}$ is small, the log loss behaves like the squared exponential loss:

$$\log\left(1+e^{-2y\hat{y}}\right) \approx e^{-2y\hat{y}} = \left(e^{-y\hat{y}}\right)^2$$

On a training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ where $y_i = \pm 1$, the total log loss is

$$\text{logLoss} = \sum_{i=1}^n \log\left(1 + e^{-2y_i \hat{y}_i}\right)$$

where $\hat{y}_i = f(\mathbf{x}_i)$ are predictions by f (e.g., an ensemble learner).

It is an estimate of the expected log loss at a random location $(\vec{X}, Y)$:

$$\mathrm{E}\left[\log\left(1 + e^{-2Yf(\vec{X})}\right)\right] \quad \leftarrow \quad \frac{1}{n}\text{logLoss}$$

It can be similarly shown [homework] that the minimizer of this loss is

$$\arg\min_f \mathrm{E}\left[\log\left(1 + e^{-2Yf(\vec{X})}\right)\right] \quad \longrightarrow \quad f(\vec{X}) = \frac{1}{2}\log\frac{P(Y=1 \mid \vec{X})}{P(Y=-1 \mid \vec{X})}$$

When linear functions are used, i.e.,

$$f(\mathbf{x}_i) = \frac{1}{2}\boldsymbol{\theta} \cdot \mathbf{x}_i,$$

the log loss is in fact identical to the logistic loss for the logistic regression model (with $0/1$ labels):

$$\text{logisticLoss} = \sum_{i=1}^{n} -y_i \log p(\mathbf{x}_i; \boldsymbol{\theta}) - (1 - y_i) \log(1 - p(\mathbf{x}_i; \boldsymbol{\theta}))$$

where

$$p(\mathbf{x}_i; \boldsymbol{\theta}) = \frac{1}{1 + e^{-\boldsymbol{\theta} \cdot \mathbf{x}_i}}.$$

Verify:

$$\begin{aligned}
\text{logisticLoss} &= \sum_{i=1}^{n} -y_i \log p(\mathbf{x}_i;\boldsymbol{\theta}) - (1-y_i)\log(1-p(\mathbf{x}_i;\boldsymbol{\theta})) \\
&= \sum_{i:y_i=1} -\log p(\mathbf{x}_i;\boldsymbol{\theta}) + \sum_{i:y_i=0} -\log(1-p(\mathbf{x}_i;\boldsymbol{\theta})) \\
&= \sum_{i:y_i=1} \log(1+\exp(-\boldsymbol{\theta}\cdot\mathbf{x}_i)) + \sum_{i:y_i=0} \log(1+\exp(\boldsymbol{\theta}\cdot\mathbf{x}_i)) \\
&= \sum_{i:y_i=1} \log(1+\exp(-\boldsymbol{\theta}\cdot\mathbf{x}_i)) + \sum_{i:y_i=-1} \log(1+\exp(\boldsymbol{\theta}\cdot\mathbf{x}_i)) \\
&= \sum_{i=1}^{n} \log(1+\exp(-y_i\boldsymbol{\theta}\cdot\mathbf{x}_i)) \;\longleftarrow\; \hat{y}_i = \frac{1}{2}\boldsymbol{\theta}\cdot\mathbf{x}_i
\end{aligned}$$

The process of training LogitBoost is very similar to that of AdaBoost, by iteratively re-weighting the training data.

Assume a space $\mathbb{H}$ of weak learners (e.g., classification trees), and let the current ensemble learner be

$$H(\mathbf{x}) = \sum_{j=1}^{t} \alpha_j h_j(\mathbf{x}), \quad h_j \in \mathbb{H}$$

with log loss:

$$\ell_n(H) = \sum_{i=1}^{n} \log\left(1 + e^{-2y_i H(\mathbf{x}_i)}\right)$$

To find the next weaker learner $h \in \mathbb{H}$, we compute the components of the gradient $\nabla \ell_n(H)$:

$$\begin{aligned} r_i = \frac{\partial \ell_n(H)}{\partial H(\mathbf{x}_i)} &= \frac{1}{1+e^{-2y_i H(\mathbf{x}_i)}} e^{-2y_i H(\mathbf{x}_i)}(-2y_i) \\ &= -2y_i \frac{1}{1+e^{2y_i H(\mathbf{x}_i)}} \\ &= -w_i y_i \end{aligned}$$

where

$$w_i = \frac{2}{1+e^{2y_i H(\mathbf{x}_i)}}, \quad 1 \le i \le n$$

The rest of the process is then the same:

$$\sum_{i=1}^{n} r_i \, h(\mathbf{x}_i) = -\sum_{i=1}^{n} w_i y_i h(\mathbf{x}_i) = 2 \sum_{h(\mathbf{x}_i) \neq y_i} w_i - 1.$$

In order to make progress (i.e., $\sum_{i=1}^{n} r_i \, h(\mathbf{x}_i) < 0$), we just need the new weak learner h to achieve the following weighted error:

$$\epsilon = \sum_{h(\mathbf{x}_i) \neq y_i} w_i < 0.5$$

MATLAB function for AdaBoost and LogitBoost

```
temp = templateTree(‘NumVariablesToSample’, ‘all’, ‘minleafsize’, 1, ‘MaxNumSplits’, 10);
mdl = fitcensemble(Xtr, ytr, ‘Method’, option, ‘NumLearningCycles’, T, ‘Learners’, temp);
% Set option = ‘AdaBoostM1’ (2 classes), or ‘AdaBoostM2’ (> 2 classes),
% or ‘LogitBoost’, or ‘Bag’ (bagging, random forest)

% make predictions on test data
pred = predict(mdl, Xtst);
```

Python functions for AdaBoost (and GradientBoost)

AdaBoost[4]

GradientBoost[5]

[4] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

[5] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

Ensemble methods on the MNIST digits

See poster at

`https://www.sjsu.edu/faculty/guangliang.chen/Math285S16/poster-Ensemble.pdf`

Summary

- Classification trees (weak learner)
- Ensemble methods
  - Independent trees: bagging, random forest
  - Adaptive trees: boosting such as AdaBoost (and GradientBoost)
- Many advantages:
  - Simple and fast
  - Can handle large data well (with automatic feature selection)
  - Excellent performance

Assignment 5

1. Apply random forest with 500 trees to the fashion-MNIST data and report the accuracy. Plot the out-of-bag error and use it to select a reasonable number of trees (T). Apply random forest again but with this size instead to the dataset. How does the corresponding test error compare with that you obtained with 500 trees?

2. Apply bagging with the value of T used in Question 1 to the fashion-MNIST data and report the accuracy. How does it compare with random forest with the same value of T?

3. Repeat Question 2 with adaptive boosting instead of bagging.

4. Implement the one-versus-one extension of LogitBoost and apply it to fashion-MNIST. Discuss your results.

Bonus (5 points). Prove the minimizer of the theoretical log loss is also half of the log odds function (see the bottom of slide 57).