LEC 7: Classification trees and ensemble learning

Guangliang Chen

Nov. 19, 2016

Outline

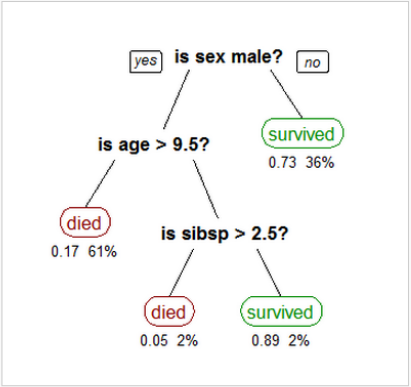
- Classification trees
- Ensemble learning
  - Bagging
  - Random forest
  - Boosting
- Summary

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]
- 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

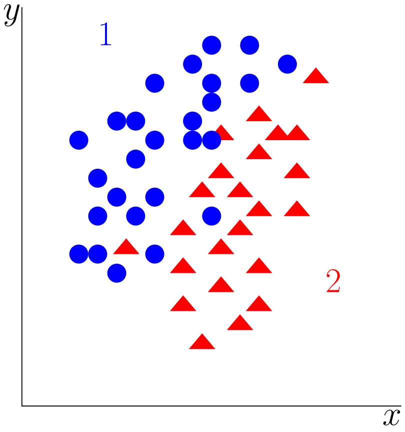
What is a classification tree?



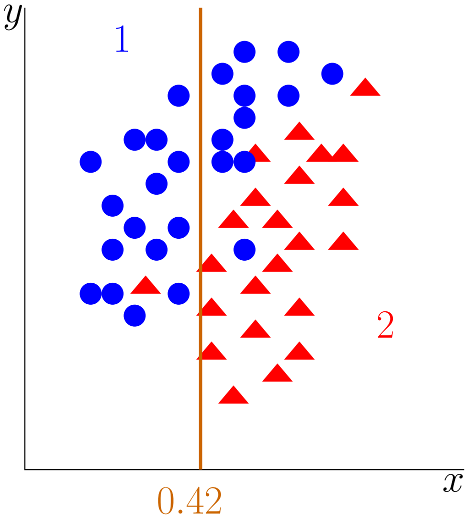
A tree showing survival of passengers on the Titanic ("sibsp" is the number of spouses or siblings aboard). The figures under the leaves show the probability of survival and the percentage of observations in the leaf.

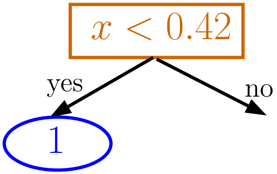
- A series of binary splits based on training data.
- Each internal node represents a query on one of the variables.
- The terminal/leaf nodes are the decision nodes, typically dominated by one of the classes.
- New observations are classified in the respective terminal nodes through majority vote.

Demo: Building a classification tree

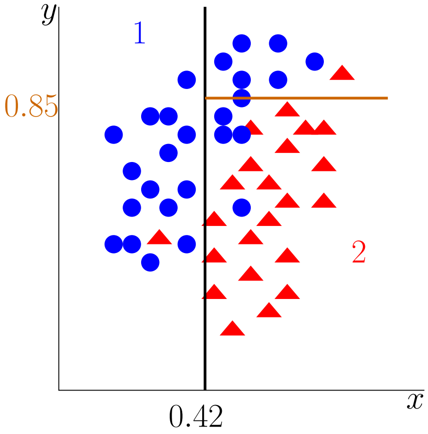


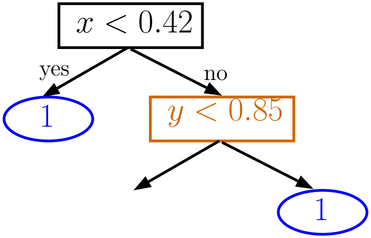
Classification trees and ensemble learning



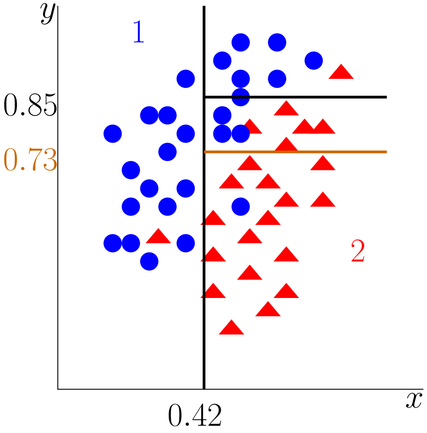


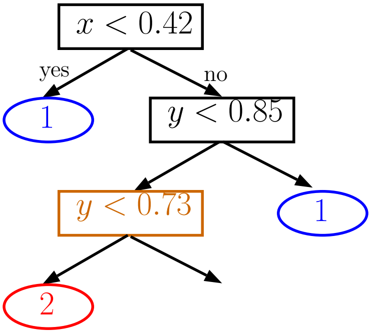
Classification trees and ensemble learning



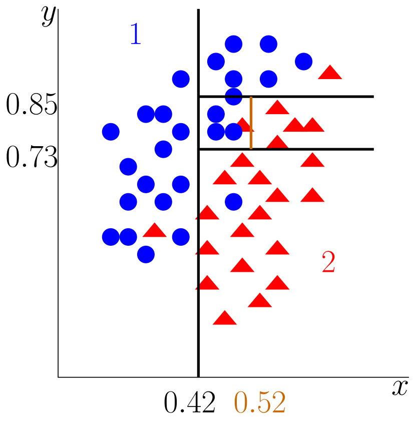


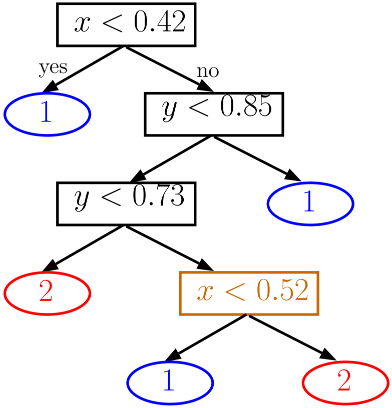
Classification trees and ensemble learning



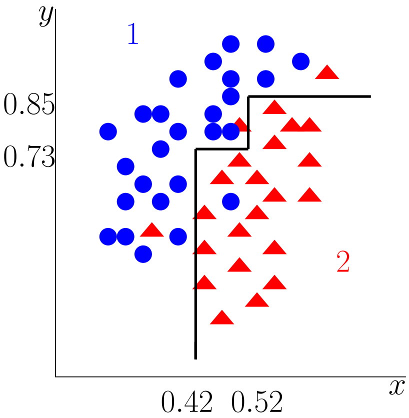


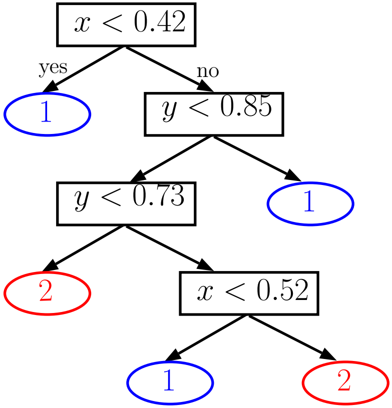
Classification trees and ensemble learning



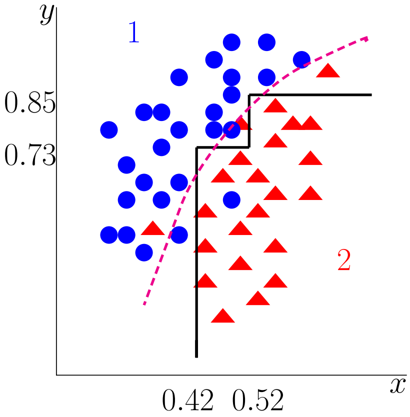


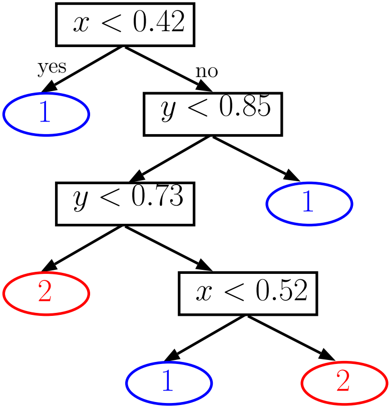
Classification trees and ensemble learning



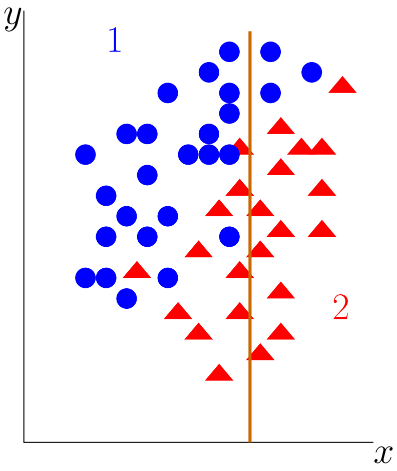


Classification trees and ensemble learning





Splitting criterion: Gini impurity score



The common approach is to minimize the **Gini diversity index** over all possible splits (variable + cutoff):

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

where

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

MATLAB function for classification trees

```
t = fitctree(trainX, trainLabels);

view(t, 'mode', 'graph') % to visualize the tree


pred = predict(t, testX);
```

(Matlab demonstration)

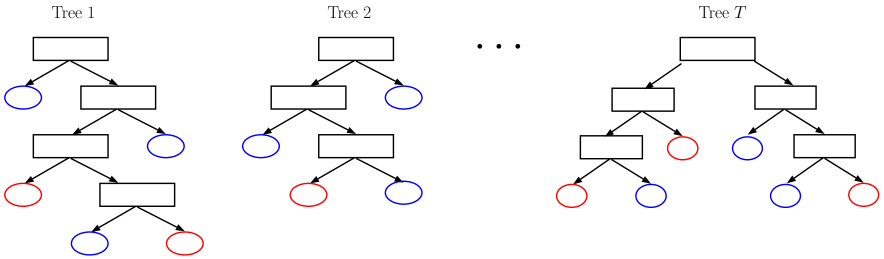
Some remarks about classification trees

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)
- No formal distributional assumptions (non-parametric)
- Works in the same way in multiclass settings
- Can handle large data sets
- Can handle categorical variables naturally

Ensemble methods

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



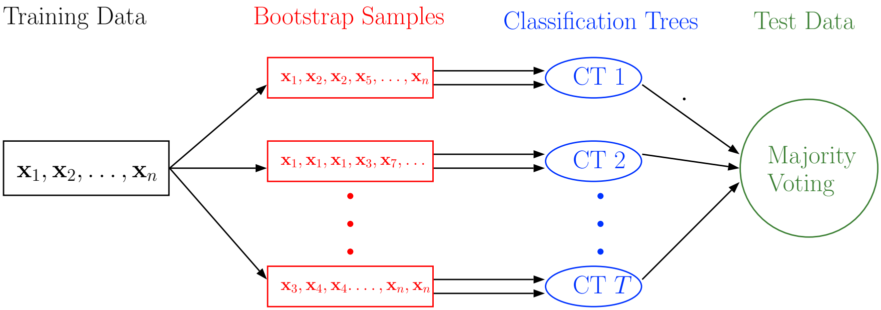
Classification trees and ensemble learning

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

Out-of-bag

- Drawing n out of n observations with replacement omits on average **36.8%** $(= e^{-1})$ of observations for each decision tree.
- We say that those samples left out by a tree are **out-of-bag (oob)** with respect to the tree.
- For each training example, we may vote the predictions of all the trees (for which the observation is oob) and compare with the true label to compute an average oob error.
- 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.

Some further comments on bagging

- 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.
- An optimal value of T can be found using cross-validation, or by observing the out-of-bag error.

MATLAB function for bagging

% train 50 trees with training data X, y and make oob predictions in the meantime
TB = TreeBagger(50, X, y, '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

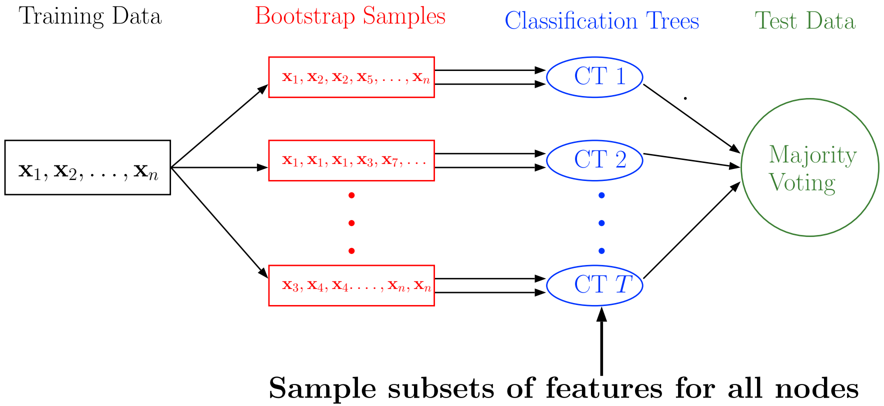
Random forest

Random forests improve bagging by **allowing every tree in the ensemble to randomly select predictors for 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
Sample subsets of features for all nodes

MATLAB function 1 for random forest

% train 50 trees with feature bagging + oob prediction
TB = TreeBagger(50, X, y, 'OOBPrediction', 'on', 'NumPredictorsToSample', 40, 'MinLeafSize', 1);
% default value of 'NumPredictorsToSample' $= \sqrt{d}$, so you don't really need to specify this option

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

MATLAB function 2 for random forest

```
% build a random forest of 50 trees for classification
ens = fitensemble(X, y, 'bag', 50, 'Tree', 'type', 'classification');

% or instead
temp = templateTree('NumVariablesToSample', √d);
ens = fitensemble(X,y,'bag',50,temp,'type','classification');


% make predictions on test data
pred = predict(ens, testX);
```

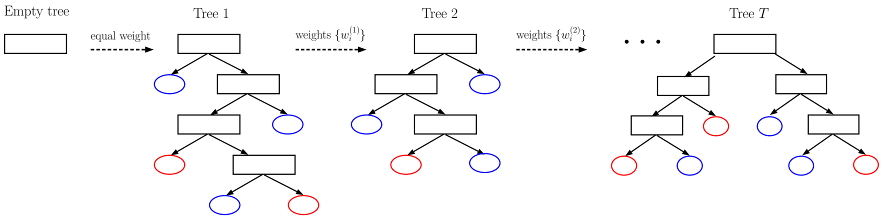
Python function for random forest

See documentation at

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

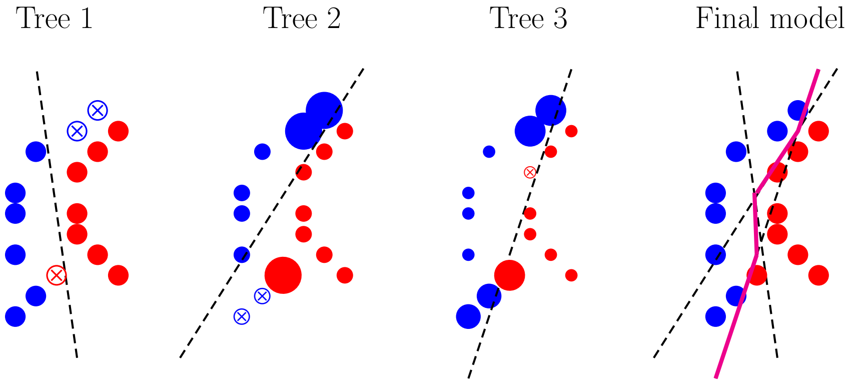
AdaBoost (Adaptive Boosting)

Main idea: build tree classifiers sequentially by“ focusing more attention” on training errors made by the preceding trees and then add their predictions.



One way to realize this idea is to **reweight the training data** for each new tree based on the training error of the previous trees.

Demo: an ensemble of 3 boosted trees



Observe that no point was predicted wrong more than once!

How to choose weights

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

For any subsequent tree $t > 1$,

- Compute the training error by first $t-1$ trees, denoted as ϵ_t
- Choose $\alpha_t = \log \frac{1-\epsilon_t}{\epsilon_t}$
- Modify the weights of the misclassified points by $w_i^{(t)} = w_i^{(t-1)} \cdot \exp(\alpha_t)$ (no change for the correctly classified points)
- Renormalize all $w_i^{(t)}$ to have a sum of 1.

How to use weights

Consider a weighted training set $(\mathbf{x}_i, y_i, w_i^{(t)}), 1 \leq i \leq n, t \geq 1$. 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

The final prediction is made based on the learned function

$$f(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t C_t(\mathbf{x})$$

by voting with weights $\alpha_t = \log \frac{1-\epsilon_t}{\epsilon_t}$ for the different trees:

$$j^* = \operatorname{argmax}_j \sum_{t \,:\, C_t(\mathbf{x})=j} \alpha_t$$

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.
- Excellent generalization performance
- Can handle many features easily
- In general, AdaBoost $\succ$ random forest $\succ$ bagging $\succ$ single tree
- Lots of variants since AdaBoost (first significant boosting method)
  - GentleBoost, LogitBoost, etc (Matlab implementation available)
  - GradientBoost

MATLAB function for AdaBoost

% build an ensemble of 50 trees by adaptive boosting
ens = fitensemble(X, y, ‘AdaBoostM1’, 50, ’Tree’); % for 2 classes
ens = fitensemble(X, y, ‘AdaBoostM2’, 50, ’Tree’); % for 3 or more classes

% make predictions on test data
pred = predict(ens, testX);

Python function for AdaBoost

See documentation at

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

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

Practice problems

1. Apply bagging with 500 trees to the USPS handwritten digits. Plot the out-of-bag error and use it to select a reasonable number of trees. Apply bagging again but with this size instead to the dataset. How does the corresponding test error compare with that you obtained with 500 trees?

2. Repeat Question 1 with random forest instead of bagging.

3. Repeat Question 1 with adaptive boosting instead of bagging.