Topics in Data Science

Lecture 14

Department of Mathematics and Statistics

PURDUE UNIVERSITY NORTHWEST

Random Forest Classifier

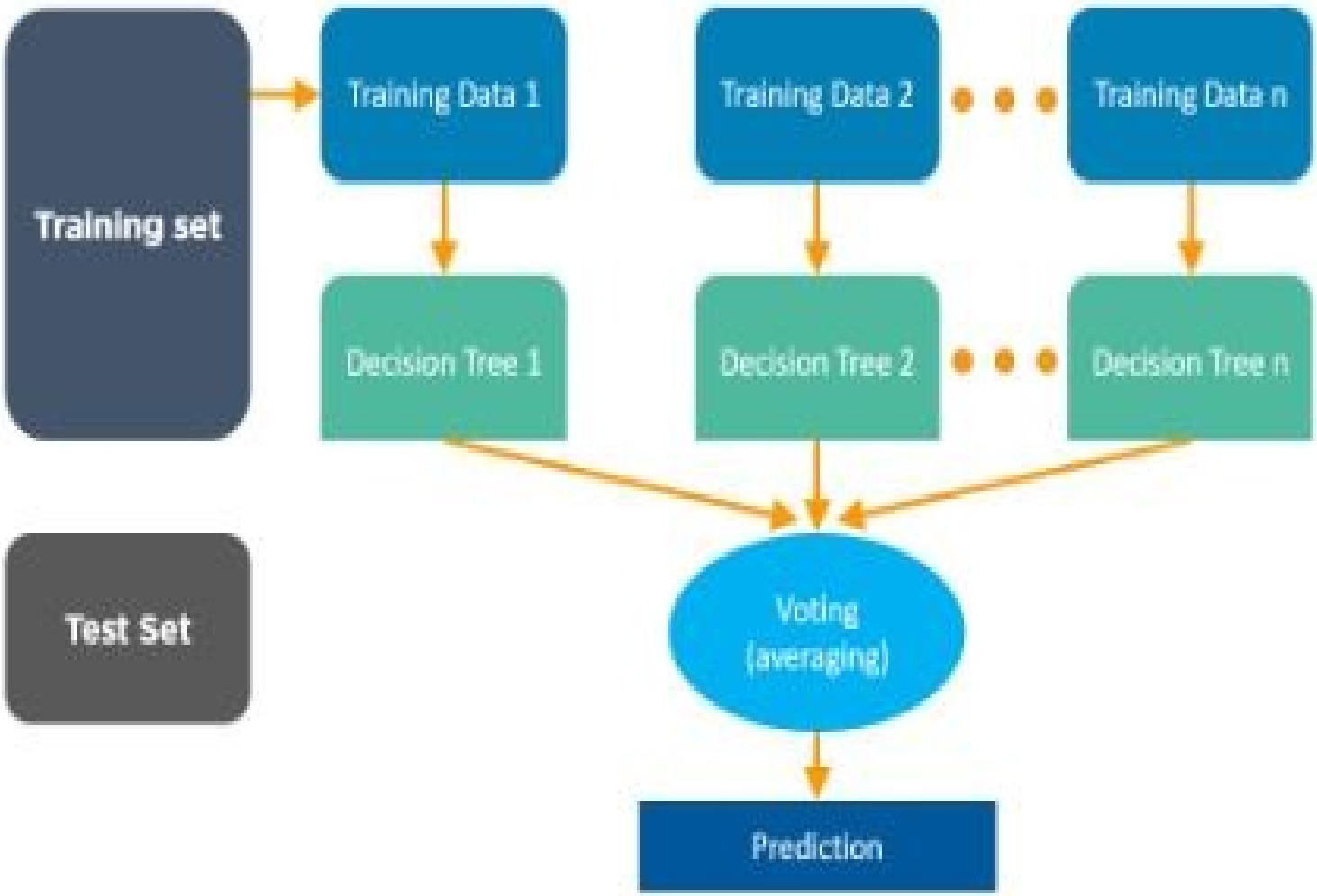
- Random forest approach is supervised nonlinear classification and regression algorithm.
- The random forest algorithm relies on multiple decision trees and accepts the results of the predictions from each tree. Based on the majority votes of predictions, it determines the final result.
- It uses a collection of decision trees providing more flexibility, accuracy, and ease of access in the output. This algorithm dominates over decision trees algorithm as decision trees provide poor accuracy as compared to the random forest algorithm
- Being a supervised learning algorithm, random forest uses the bagging method in decision trees and as a result, increases the accuracy of the learning model.
- Avoids overfitting.

One of the most important features of the Random Forest Algorithm is that it can handle the data set containing continuous variables, as in the case of regression, and categorical variables, as in the case of classification

Assumptions of Random Forest

- **Independence of Trees:** The decision trees in the forest should be independent of each other. We achieve this through bootstrap sampling and feature randomness.
- **Sufficient Data:** Random Forest requires a large amount of data to build diverse trees and achieve optimal performance.
- **Balanced Trees:** The algorithm assumes that individual trees grow sufficiently deep to capture the underlying patterns in the data.
- **Noisy Data Handling:** Random Forest can handle noisy data, but assumes that the noise is randomly distributed and is not systematic.

Random Forest- Diagram



Random Forest-History

- Introduction of the Random Subspace Method
  - "Random Decision Forests" [Ho, 1995] and "The Random Subspace Method for Constructing Decision Forests" [Ho, 1998]
  - Motivation:
    - Trees derived with traditional methods often cannot be grown to arbitrary complexity for possible loss of generalization accuracy on unseen data.
    - The essence of the methods are to build multiple trees in randomly selected subspaces of the feature space.
  - Trees in, different subspaces generalize their classification in complementary ways, and their combined classification can be monotonically improved.

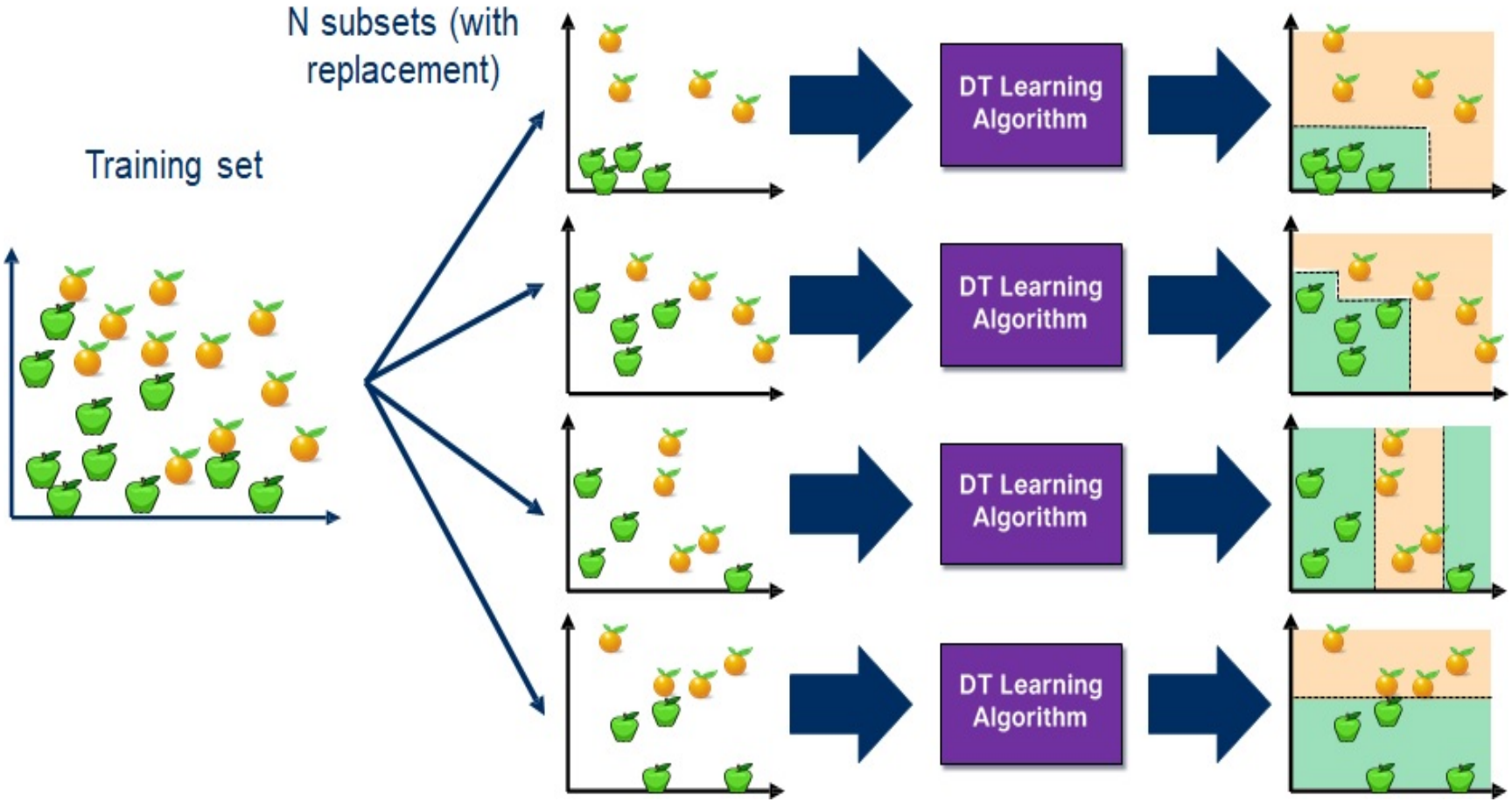
Random Forest

We have a single data set, so how do we obtain slightly different trees?

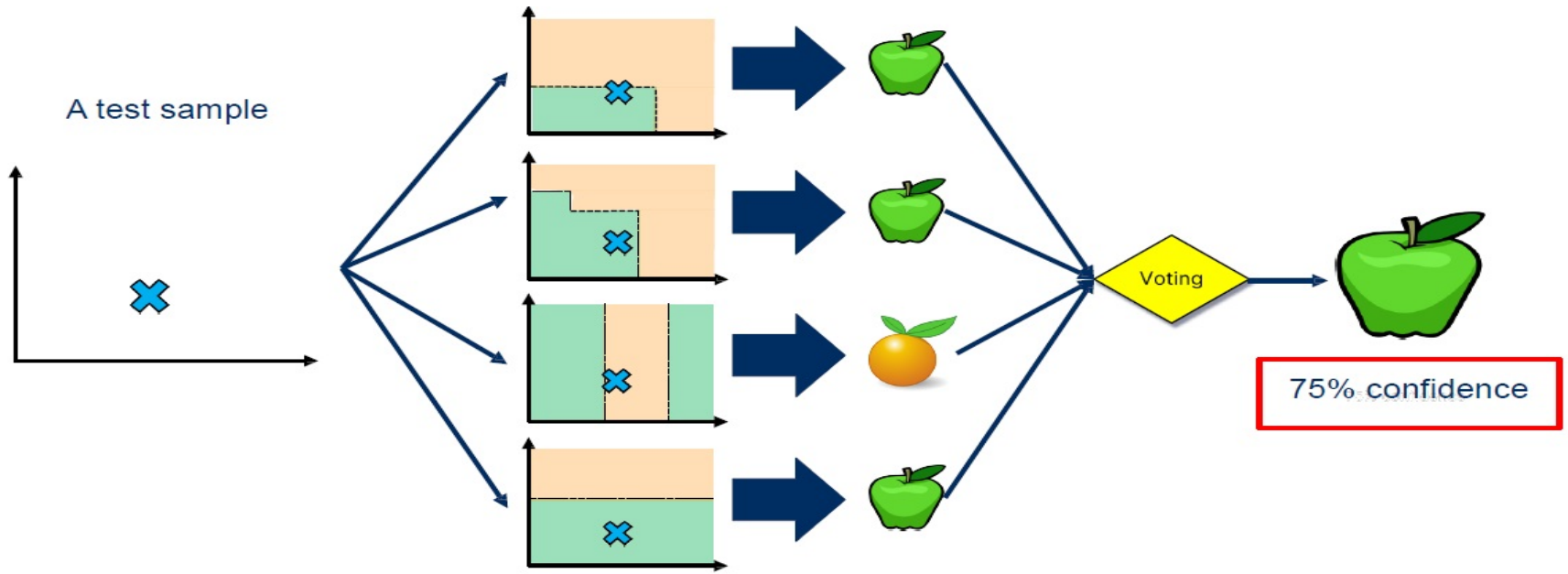
- 1. Bagging (Bootstrap Aggregating):
  - Take random subsets of data points from the training set to create N smaller data sets
  - Fit a decision tree on each subset
  - Results in low variance.
- 2. Random Subspace Method (also known as Feature Bagging):
  - Fit N different decision trees by constraining each one to operate on a random subset of features

We use row sampling with replacement to generate various models and use the training data to make the prediction from each model. Finally, we aggregate the results and make the decision using the voting classifier.

Random Forest-Bagging



Random Forest-Bagging



Key Concepts in Random Forest

- **Ensemble Learning:** This technique combines multiple models to improve performance. Random Forest is an example of ensemble learning where multiple decision trees work together to produce a more accurate and stable prediction.
- **Bagging (Bootstrap Aggregating):** In Random Forest, the algorithm creates multiple subsets of the original dataset by sampling with replacement (bootstrapping). Each tree is trained on a different subset, which helps in reducing variance and preventing overfitting.
- **Feature Randomness:** During the construction of each tree, Random Forest randomly selects a subset of features to consider at each split. This ensures that the trees are diverse and reduces the likelihood that a few strong predictors dominate the model.

High variance in decision trees

- Decision trees such as regression or classification trees suffer from high variance.
- This means that if we split the training data into two parts at random and fit a decision tree to both halves, the results can be quite different.
- In contrast, a procedure with low variance will yield similar results if applied repeatedly to distinct data sets; linear regression tends to have low variance, if the ratio of n to p is moderately large
- Bootstrap method is widely used

Bootstraping

- Bootstrapping is a technique used in inferential statistics that work on building random samples of single datasets again and again.
- Bootstrapping allows calculating measures such as mean, median, mode, confidence intervals, etc. of the sampling.
- In statistical methods, assumptions about data distributions are often necessary but not always accurate in real-world scenarios. Bootstrapping provides a flexible alternative by resampling data to estimate the sampling distribution of a statistic, useful when distribution assumptions are uncertain or sample sizes are small.
- The term "bootstrapping" was coined by Bradley Efron in 1979, who introduced the technique as a way to estimate the variability of a statistical estimate by pulling oneself up by one's bootstraps. The term is derived from the phrase "to pull oneself up by one's bootstraps," which means to use one's own resources to achieve something. In the context of statistics, bootstrapping refers to the process of using the original dataset as a resource to estimate the variability of a statistical estimate.

Bootstraping

Bootstrapping can be used in the following scenarios:

- Small sample sizes with unknown population distribution
- When assumption of normality does not hold
- Skewed data
- A non-linear combination of variables (Eg. A ratio)
- A location statistic other than the mean (Median, Difference in median)

Bagging

- Bootstrap aggregation or bagging is a general-purpose procedure for reducing the variance of a statistical learning method.
- We know that for a set of independent observations $X_1, X_2, \cdots, X_n$ each with variance σ^2, the variance of $\bar{X}$ is σ^2/n. This means that averaging a set of observations reduces variance
- One natural way to reduce the variance and hence increase the prediction accuracy of a statistical learning method is to take many training sets from the population and build a separate prediction model using each training set. Then we average the resulting prediction.

Unfortunately, we don't usually have access to many training data sets to develop the model. However, we can bootstrap!!

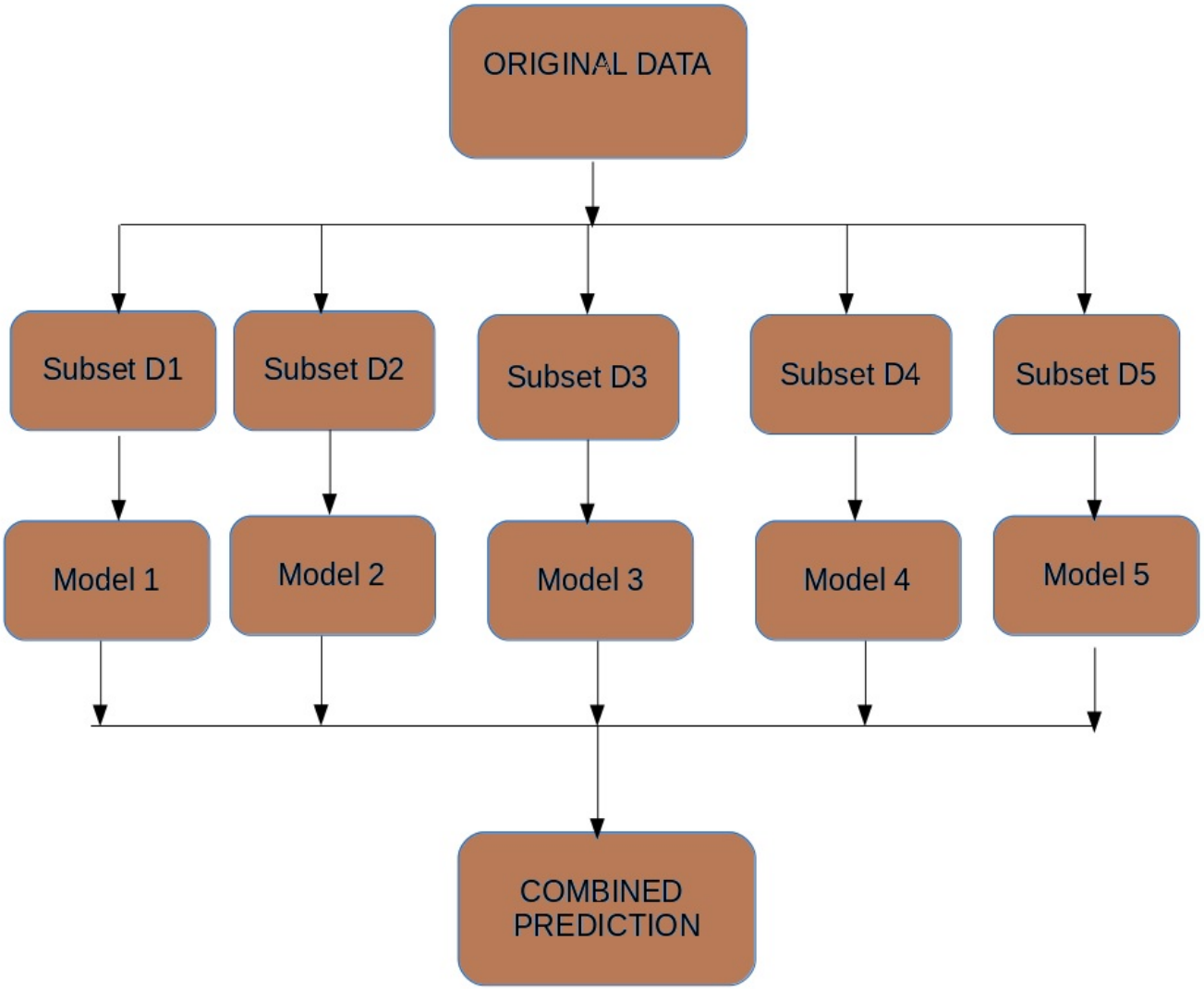
Bagging

- We can take repeated samples from the (single) training data set.
- We will generate B different bootstrapped training data sets.
- We then train our method on the bth bootstrapped training set in order to get $\hat{f}^{*b}(x)$
- We average all the predictions , to obtain

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x)$$

This is called bagging.

Bagging


ORIGINAL DATA
Subset D1
Subset D2
Subset D3
Subset D4
Subset D5
Model 1
Model 2
Model 3
Model 4
Model 5
COMBINED PREDICTION

Bagging for Regression Trees

- To apply bagging to regression trees do the following
  - construct B regression trees using B bootstrapped training sets
  - Average the resulting predictions
- Regression trees are grown deep, and are not pruned.
- Each individual tree has high variance, but low bias. Thus, a veraging these B trees reduces the variance.
- Bagging has been demonstrated to give impressive improvements in accuracy by combining together hundreds or even thousands of trees into a single procedure.

Interpretability of Bagged Trees

- Recall that bagging typically results in improved accuracy over prediction using a single tree.
- Unfortunately, however, it can be difficult to interpret the resulting model.
- But one main attraction of decision trees is their intrepretability!
- However, when we bag a large number of trees, it is no longer possible to represent the resulting statistical learning procedure using a single tree, and it is no longer clear which variables are most important to the procedure.
- Thus, bagging improves prediction accuracy at the expense of interpretability.

Bagging: Effect of Correlation

- Problem: the datasets are not independent, so we don't get the $1/m$ variance reduction.
  Possible to show that if the sampled predictions have variance σ^2 and correlation ρ, then

$$Var\left(\frac{1}{m}\sum_{i=1}^{m} y_i\right) = \frac{1}{m}(1-\rho)\sigma^2 + \rho\sigma^2$$

- Ironically, it can be advantageous to introduce additional variability into your algorithm, as long as it reduces the correlation between samples.
  - Intuition: you want to invest in a diversified portfolio, not just one stock.
  - Can help to use average over multiple algorithms, or multiple configurations of the same algorithm

Variable Importance Measures

- Although the collection of bagged trees is much more difficult to interpret than a single tree, one can obtain an overall summary of the importance of each predictor using the RSS (for bagging regression trees)
- In the case of bagging regression trees, we can record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor.

Bagging for Classification Trees

- For a given test observation, we can record the class predicted by each of the B trees and take a majority vote.
- A majority vote is simply the overall prediction is the most commonly occurring class among the B predictions.

Out-of-Bag Error Estimation

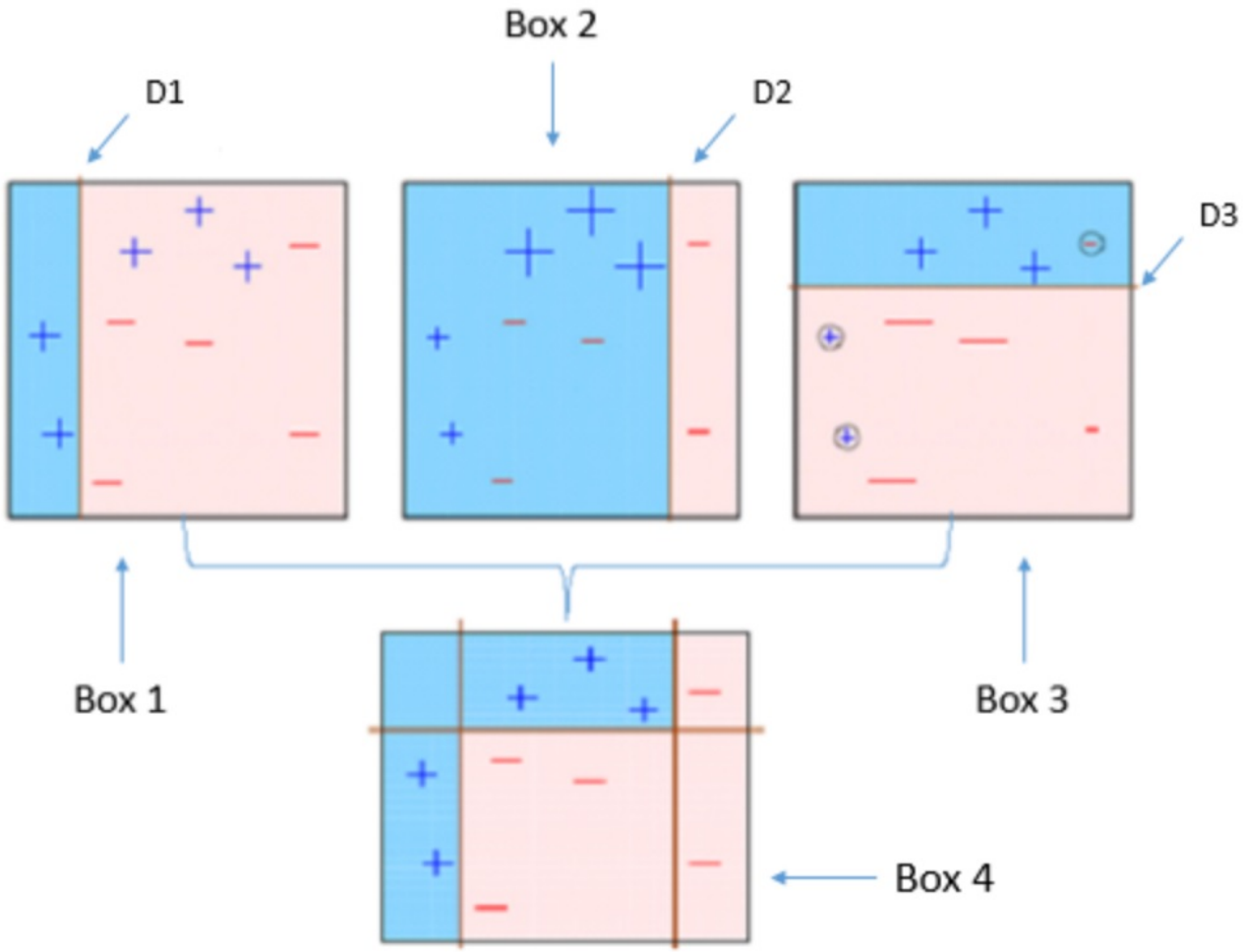
- Note that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations.
- One can show that on average, each bagged tree makes use of around two-thirds of the observations.
- The remaining one-third of the observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations.
- We can predict the response for the ith observation using each of the trees in which that observation was OOB. This will yield around B/3 predictions for the ith observation

Boosting

Boosting is an ensemble learning technique that sequentially combines multiple weak classifiers to create a strong classifier. It is done by training a model using training data and is then evaluated. The next model is built on that which tries to correct the errors present in the first model. This procedure is continued and models are added until either the complete training data set is predicted correctly or predefined number of iterations is reached.

- Boosting is another approach for improving the predictions resulting from a decision tree.
- Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification trees.

Boosting Algorithm



Boosting Type

- AdaBoost (Adaptive Boosting)
- Gradient Boosting
- XGBoost (Extreme Gradient Boosting)
- CatBoost (Categorical Boosting)
- LightGBM (Light Gradient Boosting Machine)

Adaboosting

Consider a data set with 7 observations and 3 features describing the illness status.

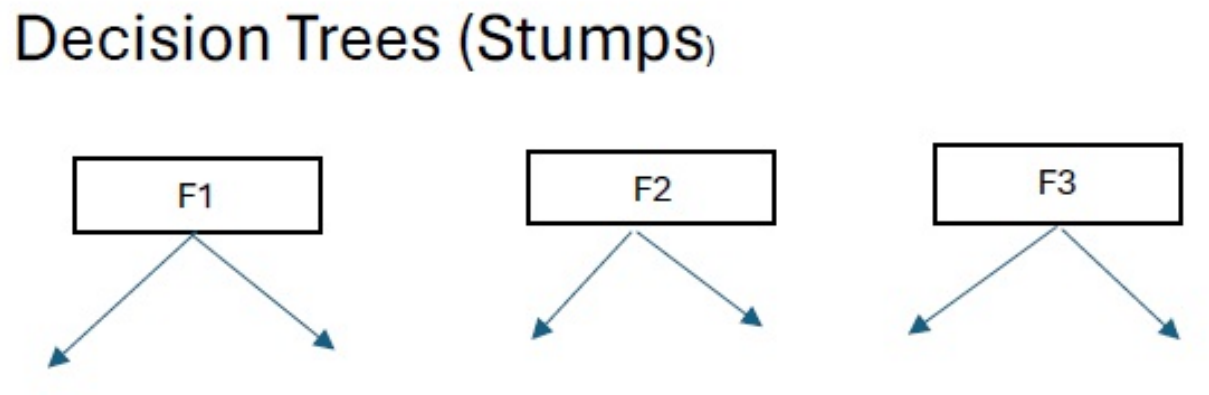
ID	*Gender*	*Age*	*Weight*	Illness
1	M	34	170	N
2	F	25	134	Y
3	F	46	150	N
4	M	19	200	N
5	F	55	122	Y
6	M	49	150	N
7	M	31	190	Y

Adaboosting

Consider a data set with 7 observations and 3 features. At the beginning each observations have the same weight 1/7.

ID	*Gender*	*Age*	*Weight*	Illness	sample weight
1	M	34	170	N	1/7
2	F	25	134	Y	1/7
3	F	46	150	N	1/7
4	M	19	200	N	1/7
5	F	55	122	Y	1/7
6	M	49	150	N	1/7
7	M	31	190	Y	1/7

Stage 1:



We will choose the stump with lowest entropy or Gini index

Adaboost

Suppose the gender has been chosen

ID	*Gender*	*Age*	*Weight*	Illness	sample weight	Prediction
1	M	34	170	N	1/7	N
2	F	25	134	Y	1/7	Y
3	F	46	150	N	1/7	Y
4	M	19	200	N	1/7	N
5	F	55	122	Y	1/7	Y
6	M	49	150	N	1/7	N
7	M	31	190	Y	1/7	N

Performance of the stump

$$\alpha = \frac{1}{2} log\left(\frac{1 - \text{Total Error}}{\text{Total Error}}\right)$$

Total Error = The sum of all the weights of the missclassified sample

Since there are two misclssified samples the total error is

$$\text{Total Error} = 2/7 = 0.285$$

Therefore, $\alpha = 0.2$ Therefore, the quantity of performance or say of the gender on the total prediction is 0.2
Based on the performance or alpha value we need to calculate the new weights

$$\text{new weight} = \text{old weight} * e^{\pm alpha}$$

Therefore, for successfully classified sample we have

$$\text{new weight} = 1/7 * e^{-0.2} = 0.117$$

for misclassified sample

$$\text{new weight} = 1/7 * e^{0.2} = 0.174$$

Adaboost Example-Contd.

ID	*Gender*	*Age*	*Weight*	Illness	sample weight	New weight
1	M	34	170	N	1/7	0.117
2	F	25	134	Y	1/7	0.117
3	F	46	150	N	1/7	0.174
4	M	19	200	N	1/7	0.117
5	F	55	122	Y	1/7	0.117
6	M	49	150	N	1/7	0.117
7	M	31	190	Y	1/7	0.174

Note that the total sum of the new weight is 0.933. However we need to make the total weigh 1. Therefore we divede each new weight by 0.933 and the resulting values are

ID	*Gender*	*Age*	*Weight*	Illness	sample weight	New weight
1	M	34	170	N	1/7	0.1254
2	F	25	134	Y	1/7	0.1254
3	F	46	150	N	1/7	0.1865
4	M	19	200	N	1/7	0.1254
5	F	55	122	Y	1/7	0.1254
6	M	49	150	N	1/7	0.1254
7	M	31	190	Y	1/7	0.1865

Adaboost example-contd.

Now, we need to create the new Buckets

ID	*Gender*	*Age*	*Weight*	Illness	A. weight	N. weight	New Bucket
1	M	34	170	N	1/7	0.1254	0 to 0.1254
2	F	25	134	Y	1/7	0.1254	0.1254 to 0.251
3	F	46	150	N	1/7	0.1865	0.251 to 0.437
4	M	19	200	N	1/7	0.1254	0.437 to 0.563
5	F	55	122	Y	1/7	0.1254	0.563 to 0.688
6	M	49	150	N	1/7	0.1254	0.688 to 0.814
7	M	31	190	Y	1/7	0.1865	0.814 to 1

Note that the bucket for incorrectly classified records is wider than for correctly classified records.
Now, we will generate 7 random numbers between 0 and 1. Suppose we got the values below:
0.9, 0.541,0.418,0.715,0.272,0.518 and 0.100

Adaboot Example -contd.

	Gender	*Age*	*Weight*	Illness	New Bucket
7	M	31	190	Y	0.814 to 1
4	M	19	200	N	0.437 to 0.563
3	F	46	150	N	0.251 to 0.437
6	M	49	150	N	0.688 to 0.814
3	F	46	150	N	0.251 to 0.437
4	M	19	200	N	0.437 to 0.563
1	M	34	170	N	0 to 0.125

Boosting for Regression Trees

- Unlike fitting a single large decision tree to the data, which amounts to fitting the data hard and potentially overfitting, the boosting approach instead learns slowly.
- In general, statistical learning approaches that learn slowly tend to perform well. Note that in boosting, unlike in bagging, the construction of each tree depends strongly on the trees that have already been grown.
- Given the current model, we fit a decision tree to the residuals from the model. That is, we fit a tree using the current residuals r rather than the outcome Y, as the response.
- We then add this new decision tree into the fitted function in order to update the residuals.

Weighted Training set

https://www.cs.toronto.edu/~rgrosse/courses/csc311_f20/slides/lec06.pdf

Boosting for regression Trees

- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.
- By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well.
- The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals

Boosting Algorithm for Regression Trees

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training dataset.
2. For $b = 1, 2, \cdots, B$ repeat
   - fit a tree $\hat{f}^b$ with d splits ((d+1) terminal nodes) to the training data (X, r)
   - Update $\hat{f}(x)$ by adding the shrunken version of the new tree:

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

   - Update the residuals

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

3. Output the boosted model

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

Tuning Parameters

There are three tuning parameters that we must carefully consider.

1. The number of trees B.
   Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all
2. The shrinkage parameter λ:
   This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the application. Very small λ can require using a very large value of B in order to achieve good performance.
3. The number d splits in each tree.
   This controls the complexity of the boosted ensemble. Often $d = 1$ works well, in which case each tree is a stump, consisting of a single split.

Bagging and Boosting

- Growing a random forest proceeds in exactly the same way, except that we use a smaller value of the mtry argument.
- By default, randomForest() uses $p/3$ variables when building a random forest of regression trees, and $\sqrt{p}$ variables when building a random forest of classification trees.

Bagging and Boosting using R

```
library(MASS)
data("Boston")
attach(Boston)
train=sample(1:nrow(Boston), nrow(Boston)*0.7)
tree.boston=tree(medv~., data=Boston, subset=train)
plot(tree.boston)
text(tree.boston)
yhat=predict(tree.boston,newdata=Boston[-train,])
boston.test=Boston[-train,"medv"]
mean((yhat-boston.test)^2)

##################
cv.boston=cv.tree(tree.boston)
plot(cv.boston, type="b")
prune.boston=prune.tree(tree.boston, best=7)
yhat=predict(prune.boston,newdata=Boston[-train,])
mean((yhat-boston.test)^2)
```

Bagging and Boosting using R

```
################### Random Forest and Bagging
library(randomForest)
bag.boston=randomForest(medv~.,data=Boston, subset=train, mtry=4, importance=T)
bag.boston
yhat.bag=predict(bag.boston, newdata=Boston[-train,])
mean((yhat.bag-boston.test)^2)
varImpPlot(bag.boston)

###########Boosting
library(gbm)
boost.boston=gbm(medv~., data=Boston[train,], distribution = "gaussian",
                 n.trees=5000, interaction.depth = 4  )
yhat.boost=predict(boost.boston, newdata=Boston[-train,], n.trees=5000)
mean((yhat.boost-boston.test)^2)


boost.boston=gbm(medv~., data=Boston[train,], distribution = "gaussian",
                    n.trees=5000, interaction.depth = 4, shrinkage = 0.2 )
yhat.boost=predict(boost.boston, newdata=Boston[-train,], n.trees=5000)
mean((yhat.boost-boston.test)^2)
```