Machine Learning Notes 12

Ensemble Methods

Ensemble methods

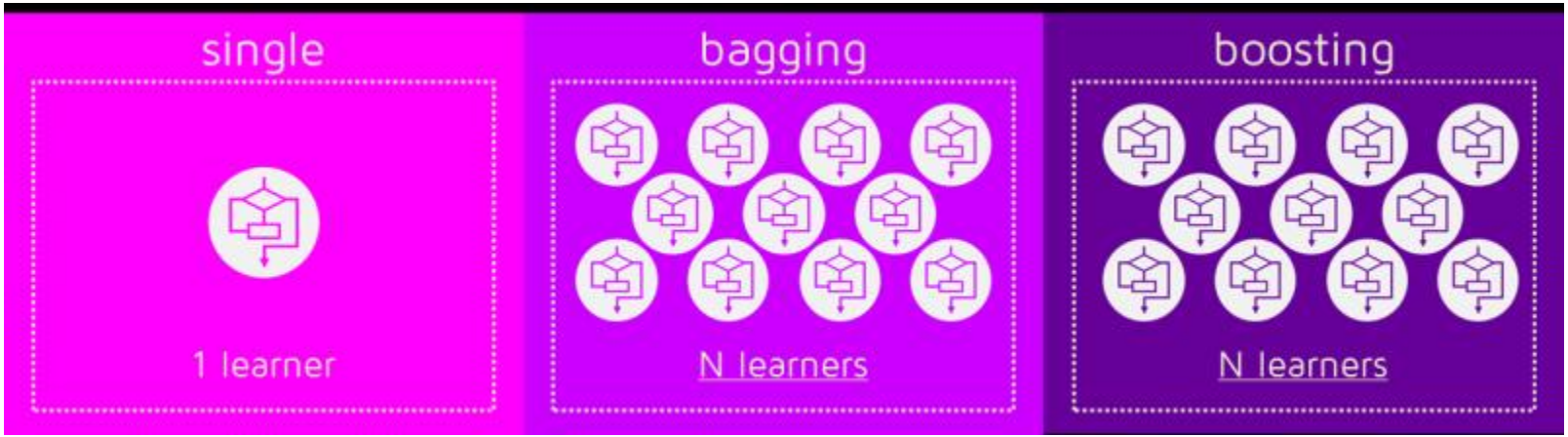
- Ensemble is a Machine Learning concept in which the idea is to train multiple models using the same learning algorithm
- The ensembles take part in a bigger group of methods, called multiclassifiers, where a set of hundreds or thousands of learners with a common objective are fused together to solve the problem.
- The second group of multiclassifiers contain the hybrid methods. They use a set of learners too, but they can be trained using different learning techniques. **Stacking** is the most well-known.

Ensemble methods

- The main causes of error in learning are due to noise, bias and variance.
- Ensemble helps to minimize these factors. These methods are designed to improve the stability and the accuracy of Machine Learning algorithms.
- Combinations of multiple classifiers decrease variance, especially in the case of unstable classifiers, and may produce a more reliable classification than a single classifier.

Ensemble methods

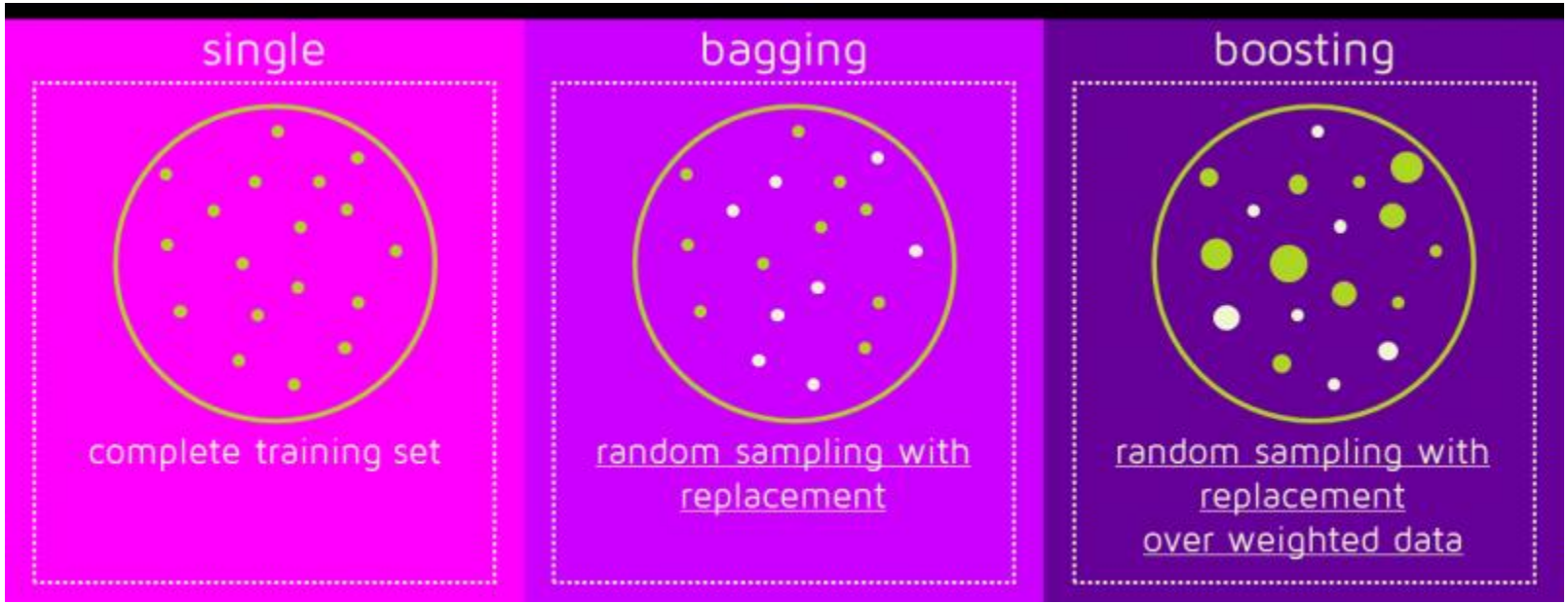
- To use Bagging or Boosting you must select a base learner algorithm. For example, if we choose a classification tree, Bagging and Boosting would consist of a pool of trees as big as we want.



Ensemble methods

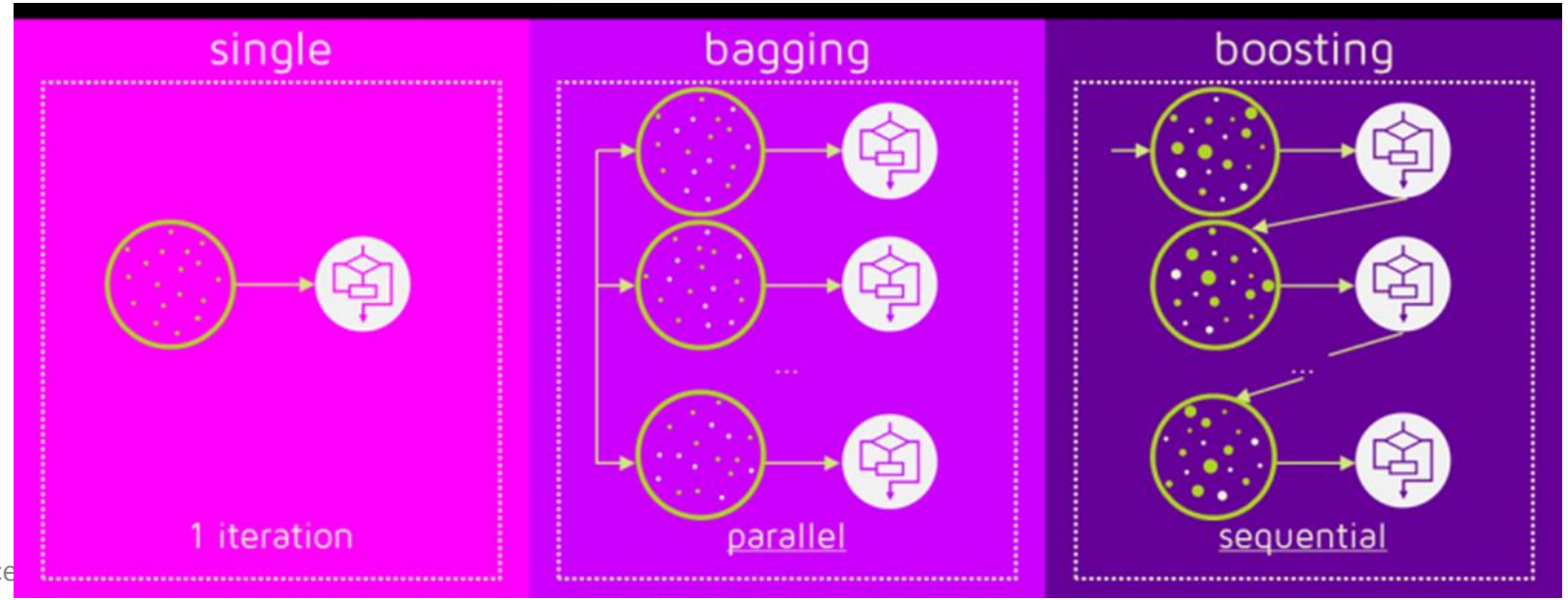
- Bagging and Boosting get N learners by generating additional data in the training stage. N new training data sets are produced by random sampling with replacement from the original set. By sampling with replacement some observations may be repeated in each new training data set.

Ensemble methods



Ensemble methods

- While the training stage is parallel for Bagging (i.e., each model is built independently), Boosting builds the new learner in a sequential way:



Ensemble methods

- In Boosting algorithms each classifier is trained on data, taking into account the previous classifiers' success. After each training step, the weights are redistributed. Misclassified data increases its weights to emphasise the most difficult cases.

Bagging (Bootstrap AGGregation)

- Bagging is used when the goal is to reduce the variance of a decision tree classifier. Here the objective is to create several subsets of data from training sample chosen randomly with replacement. Each collection of subset data is used to train their decision trees

Bagging (Bootstrap AGGregation)

Partitioning of data	Random
Goal to achieve	Minimum variance
Methods used	Random subspace
Functions to combine single model	Weighted average
Example	Random Forest

Bagging Steps

- Suppose there are N observations and M features in training data set. A sample from training data set is taken randomly with replacement.
- A subset of M features are selected randomly and whichever feature gives the best split is used to split the node iteratively.
- The tree is grown to the largest.
- Above steps are repeated n times and prediction is given based on the aggregation of predictions from n number of trees.

Advantages vs Disadvantages

- **Advantages:**
  - Reduces over-fitting of the model.
  - Handles higher dimensionality data very well.
  - Maintains accuracy for missing data.
- **Disadvantages:**
  - Since final prediction is based on the mean predictions from subset trees, it won't give precise values for the classification and regression model.

Python

- rfm = RandomForestClassifier(n_estimators=80, oob_score=True, n_jobs=-1, random_state=101, max_features = 0.50, min_samples_leaf = 5)
- fit(x_train, y_train)
- predicted = rfm.predict_proba(x_test)

Boosting

- In Boosting algorithms each classifier is trained on data, taking into account the previous classifiers' success. After each training step, the weights are redistributed. Misclassified data increases its weights to emphasise the most difficult cases.

Boosting

Partitioning of data	Higher vote to misclassified samples
Goal to achieve	Increase accuracy
Methods used	Gradient descent
Functions to combine single model	Weighted majority vote
Example	Ada Boost

Boosting

- Boosting is used to create a collection of predictors. In this technique, learners are learned sequentially with early learners fitting simple models to the data and then analysing data for errors. Consecutive trees (random sample) are fit and at every step, the goal is to improve the accuracy from the prior tree. When an input is misclassified by a hypothesis, its weight is increased so that next hypothesis is more likely to classify it correctly. This process converts weak learners into better performing model.

Boosting Steps:

- Draw a random subset of training samples d1 without replacement from the training set D to train a weak learner C1
- Draw second random training subset d2 without replacement from the training set and add 50 percent of the samples that were previously falsely classified/misclassified to train a weak learner C2
- Find the training samples d3 in the training set D on which C1 and C2 disagree to train a third weak learner C3
- Combine all the weak learners via majority voting.

Advantages vs Disadvantages

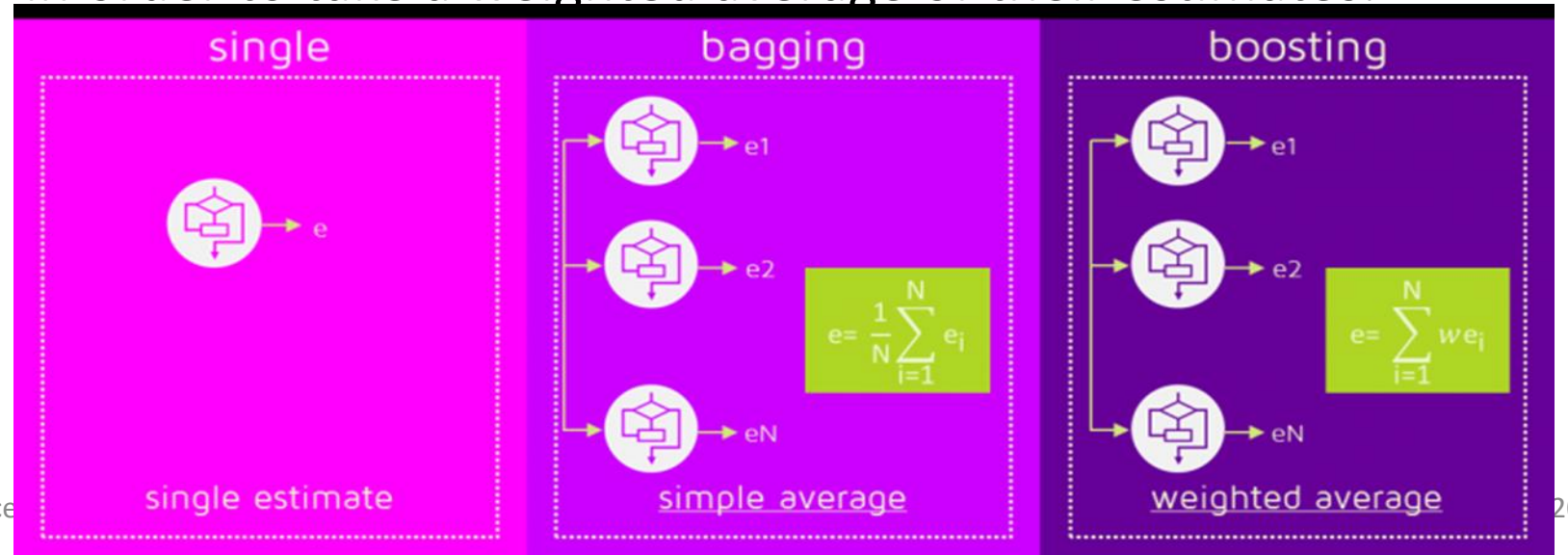
- **Advantages:**
  - Supports different loss function (we have used ‘binary:logistic’ for this example).
  - Works well with interactions.
- **Disadvantages:**
  - Prone to over-fitting.
  - Requires careful tuning of different hyper-parameters.

Python

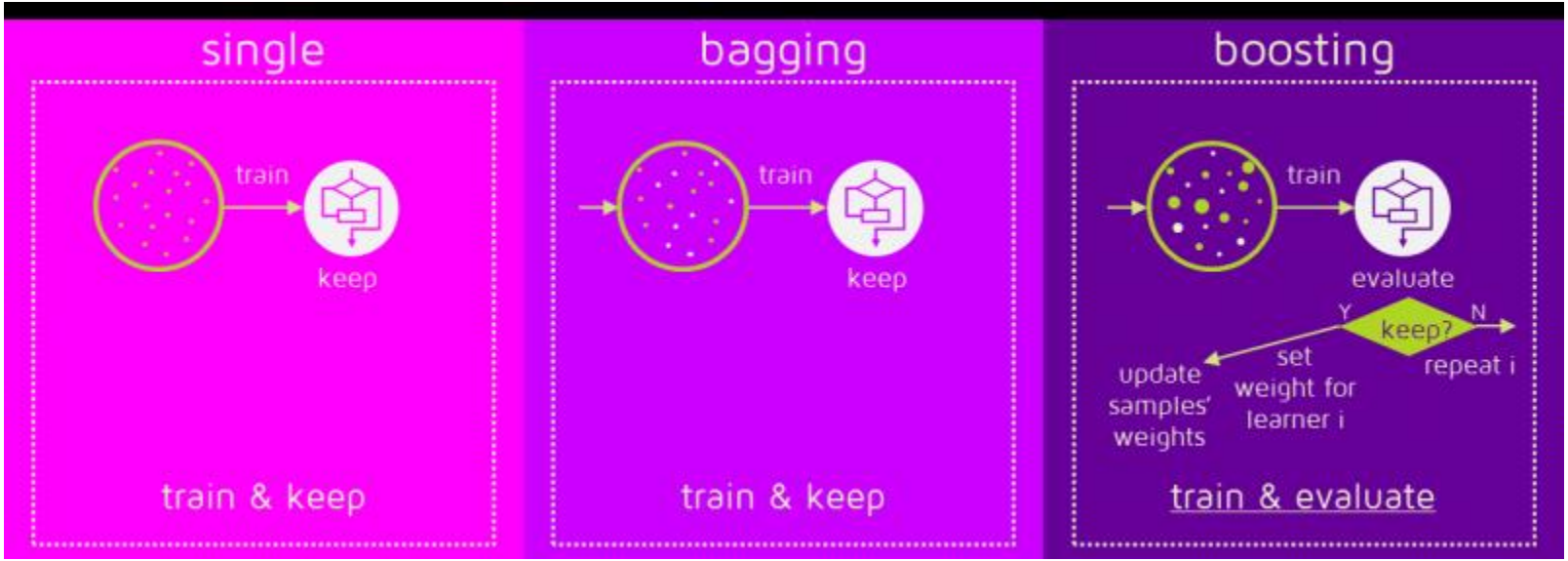
- from xgboost import XGBClassifier
- xgb = XGBClassifier(objective='binary:logistic', n_estimators=70, seed=101)
- fit(x_train, y_train)
- predicted = xgb.predict_proba(x_test)

How does the classification stage work

- In Bagging the result is obtained by averaging the responses of the N learners (or majority vote). However, Boosting assigns a second set of weights, this time for the N classifiers, in order to take a weighted average of their estimates.



- In the Boosting training stage, the algorithm allocates weights to each resulting model. A learner with good a classification result on the training data will be assigned a higher weight than a poor one. So when evaluating a new learner, Boosting needs to keep track of learners' errors, too. Let's see the differences in the procedures:
- Some of the Boosting techniques include an extra-condition to keep or discard a single learner. For example, in AdaBoost, the most renowned, an error less than 50% is required to maintain the model; otherwise, the iteration is repeated until achieving a learner better than a random guess.


single
train
keep
train & keep
bagging
train
keep
train & keep
boosting
train
evaluate
Y
keep?
N
update samples' weights
set weight for learner i
repeat i
train & evaluate

Random Forest

Information Gain

- Slides from https://www.cs.cmu.edu
- Information gain is one criteria to decide on the
- attribute.

Information

- Imagine:
- 1. Someone is about to tell you your own name
- 2. You are about to observe the outcome of a dice roll
- 2. You are about to observe the outcome of a coin flip
- 3. You are about to observe the outcome of a biased coin flip

- Each situation have a different *amount of uncertainty*
- as to what outcome you will observe.

Information

- Information:
- reduction in uncertainty (amount of surprise in the outcome)

$$I(E) = \log_2 \frac{1}{p(x)} = -\log_2 p(x)$$

If the probability of this event happening is small and it happens the information is large.

- Observing the outcome of a coin flip is head ⟶ $I = -\log_2 1/2 = 1$

2. Observe the outcome of a dice is 6 ⟶ $I = -\log_2 1/6 = 2.58$

Entropy

- The *expected amount of information* when observing the output of a random variable X

$$H(X) = E(I(X)) = \sum_i p(x_i) I(x_i) = -\sum_i p(x_i) \log_2 p(x_i)$$

If there X can have 8 outcomes and all are equally likely

$$H(X) == -\sum_i 1/8 \log_2 1/8 = 3 \quad \text{bits}$$

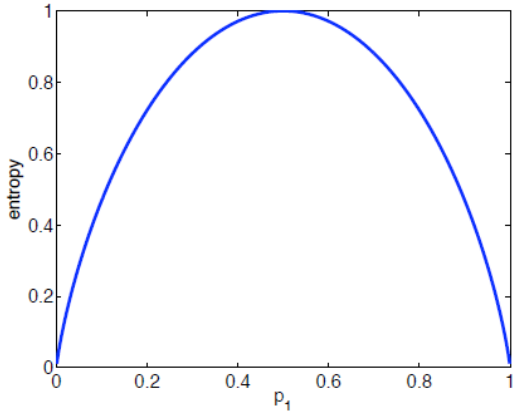
Entropy

- If there are *k* possible outcomes

$$H(X) \leq \log_2 k$$

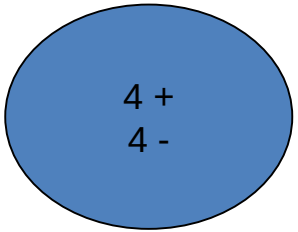
- Equality holds when all outcomes are equally likely

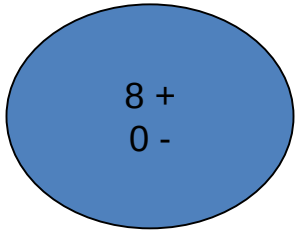
- The more the probability dis
- the deviates from
- uniformity
- the lower the entropy



Entropy, purity

- Entropy measures the purity





The distribution is less uniform
Entropy is lower
The node is purer

Conditional entropy

$$H(X) = -\sum_i p(x_i) \log_2 p(x_i)$$

$$H(X \mid Y) = -\sum_j p(y_j) H(X \mid Y = y_j)$$

$$= -\sum_j p(y_j) \sum_i p(x_i \mid y_j) \log_2 p(x_i \mid y_j)$$

Information gain

- IG(X,Y)=H(X)-H(X|Y)

Reduction in uncertainty by knowing Y

Information gain:
(information before split) – (information after split)

Information Gain

- Information gain:
- (information before split) – (information after split)

Example

Attributes Labels

X1	X2	Y	Count
T	T	+	2
T	F	+	2
F	T	-	5
F	F	+	1

Which one do we choose X1 or X2?

$IG(X1,Y) = H(Y) - H(Y|X1)$

$H(Y) = -(5/10)\log(5/10) - 5/10\log(5/10) = 1$

$H(Y|X1) = P(X1=T)H(Y|X1=T) + P(X1=F)H(Y|X1=F)$

$= 4/10\,(1\log 1 + 0\log 0) + 6/10\,(5/6\log 5/6 + 1/6\log 1/6)$

$= 0.39$

Information gain (X1,Y)= 1-0.39=0.61

Which one do we choose?

X1	X2	Y	Count
T	T	+	2
T	F	+	2
F	T	-	5
F	F	+	1

Information gain (X1,Y)= 0.61

Information gain (X2,Y)= 0.12

Pick the variable which provides the most information gain about Y

Pick X1

Recurse on branches

X1	X2	Y	Count
T	T	+	2
T	F	+	2
F	T	-	5
F	F	+	1

One branch

The other branch

Caveats

- The number of possible values influences the information gain.

- The more possible values, the higher the gain (the more likely it is to form small, but pure partitions)

Purity (diversity) measures

- Purity (Diversity) Measures:
- – Gini (population diversity)
- – Information Gain
- – Chi-square Test

Overfitting

- You can perfectly fit to any training data
- Zero bias, high variance

- Two approaches:
- 1. Stop growing the tree when further splitting the data does not yield an improvement
- 2. Grow a full tree, then prune the tree, by eliminating nodes.

Bagging

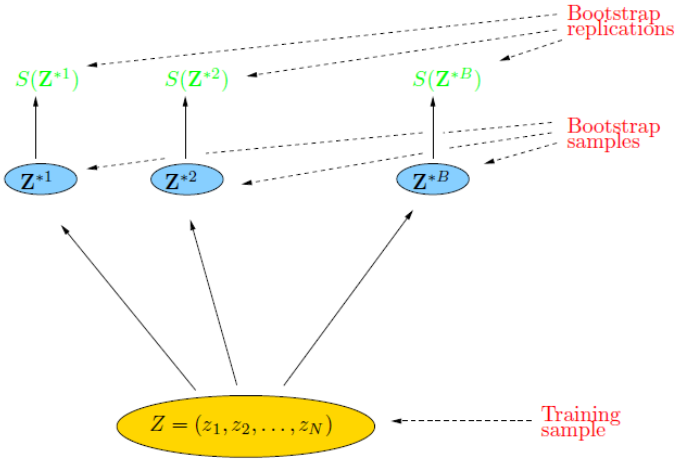
- Bagging or *bootstrap aggregation* a technique for reducing the variance of an estimated prediction function.

- For classification, a *committee* of trees each
- cast a vote for the predicted class.

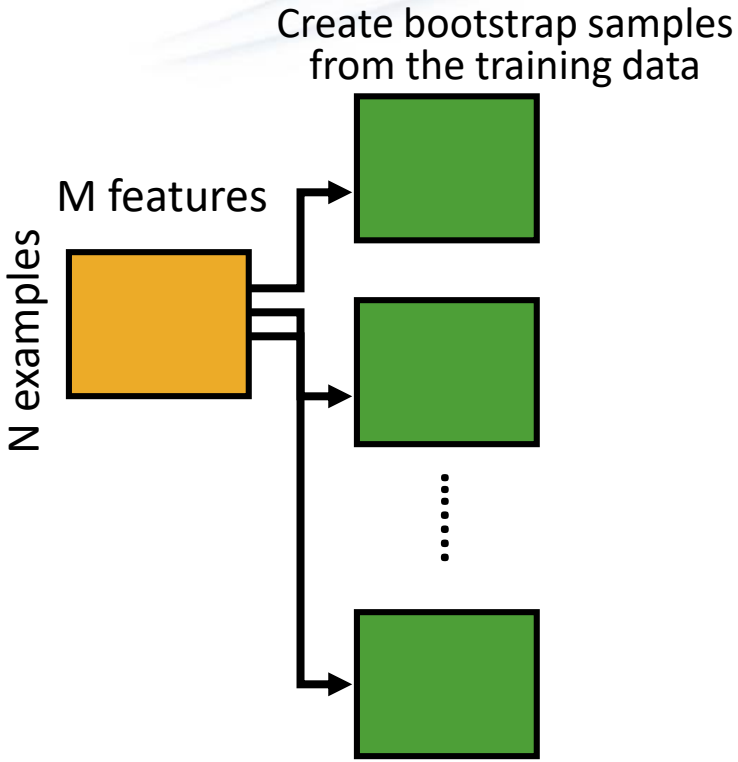
Bootstrap

The basic idea:

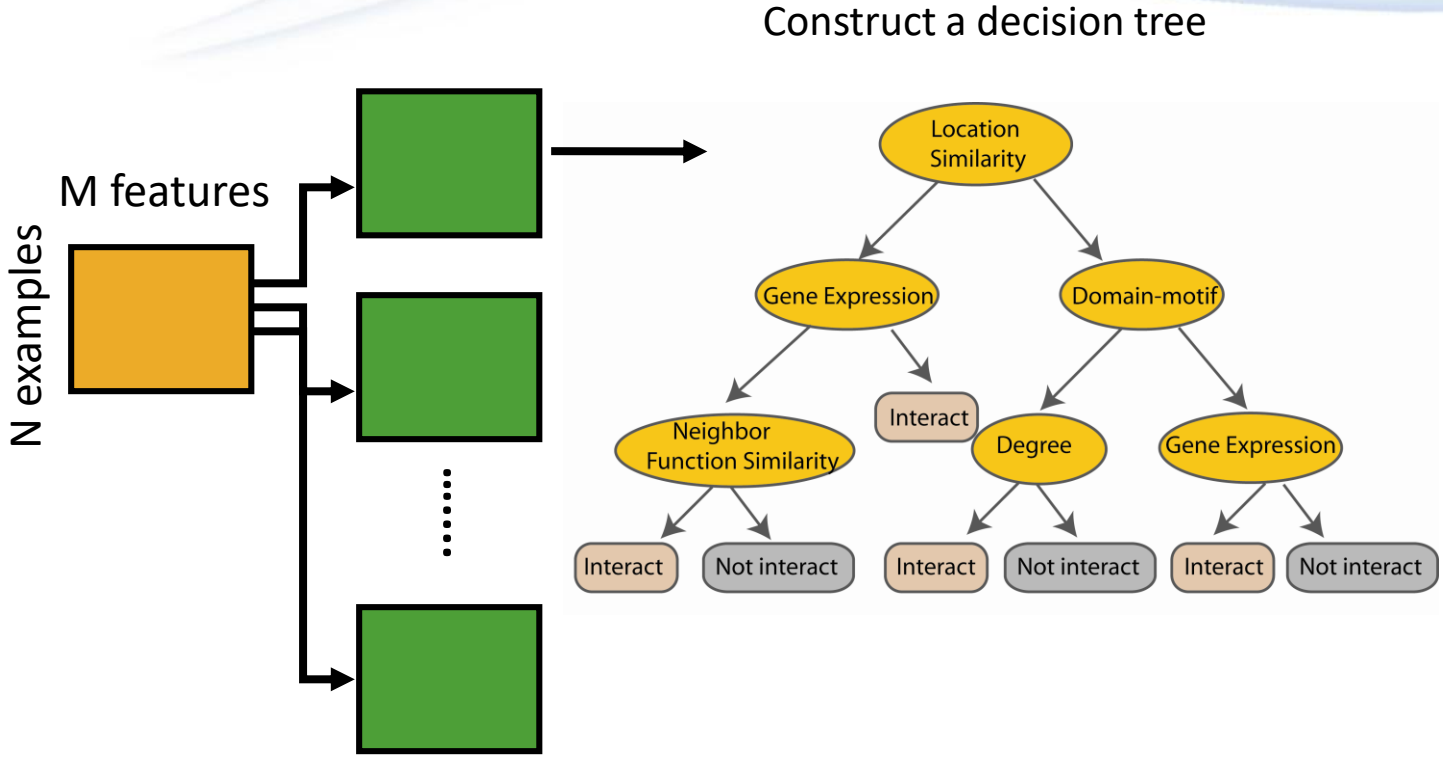
randomly draw datasets *with replacement* from the training data, each sample *the same size as the original training set*



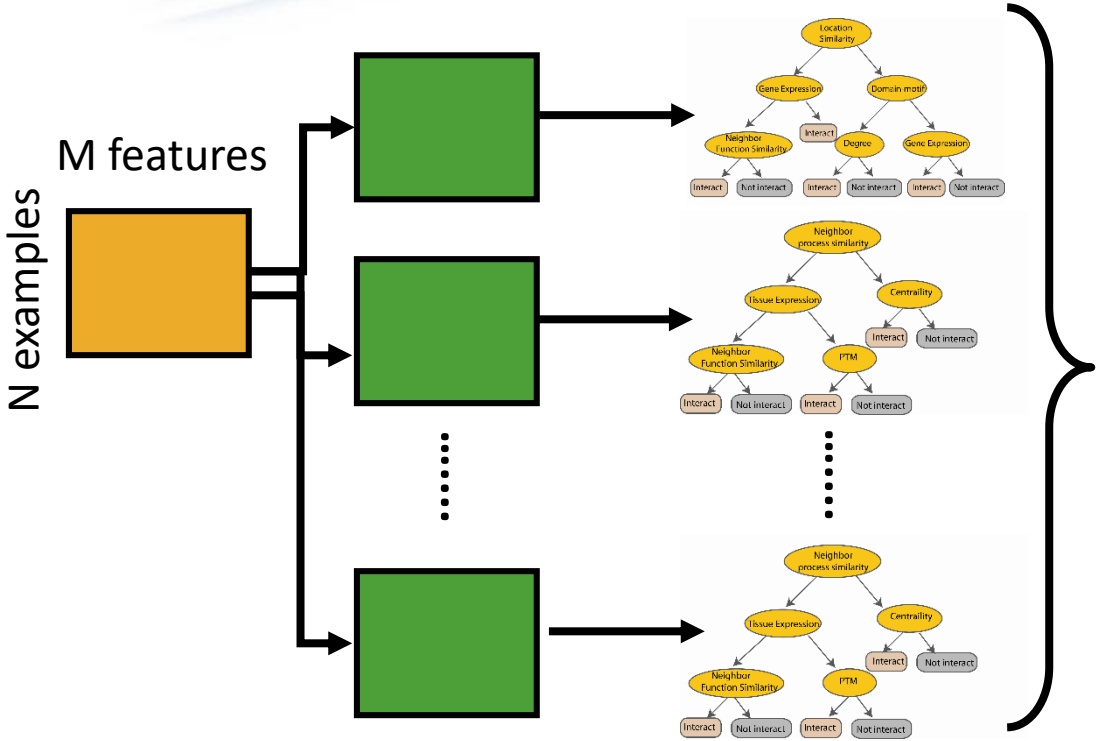
Bagging


Create bootstrap samples
from the training data
M features
N examples

Random Forest Classifier



Random Forest Classifier



Bagging

$\mathbf{Z} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$

Z^{*b} where = 1,.., B..

The prediction at input x when bootstrap sample *b* is used for training

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

http://www-stat.stanford.edu/~hastie/Papers/ESLII.pdf (Chapter 8.7)

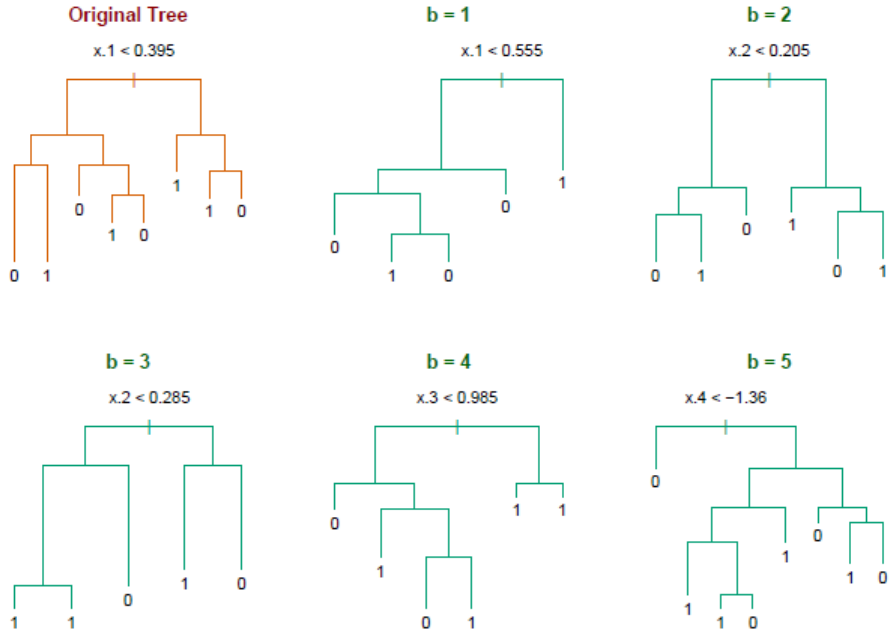
Bagging : an simulated example

- Generated a sample of size $N = 30$, with two
- classes and $p = 5$ features, each having a
- standard Gaussian distribution with pairwise
- Correlation 0.95.

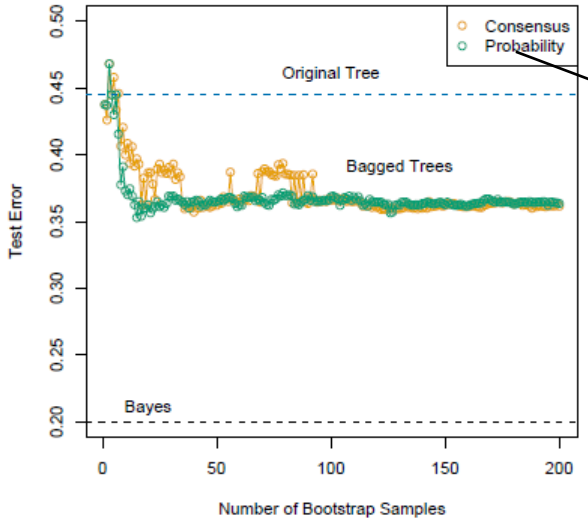
- The response Y was generated according to
- $\Pr(Y = 1 / x1 \leq 0.5) = 0.2$,
- $\Pr(Y = 0 / x1 > 0.5) = 0.8$.

Bagging

Notice the bootstrap trees are different than the original tree



Bagging



Treat the voting Proportions as probabilities

FIGURE 8.10. *Error curves for the bagging example of Figure 8.9. Shown is the test error of the original tree and bagged trees as a function of the number of bootstrap samples. The orange points correspond to the consensus vote, while the green points average the probabilities.*

bagging helps under squared-error loss, in short because averaging reduces

Hastie

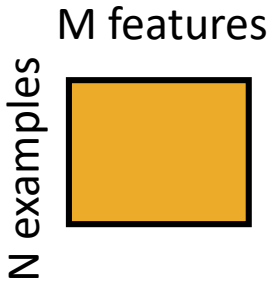
http://www-stat.stanford.edu/~hastie/Papers/ESLII.pdf Example 8.7.1

Random forest classifier

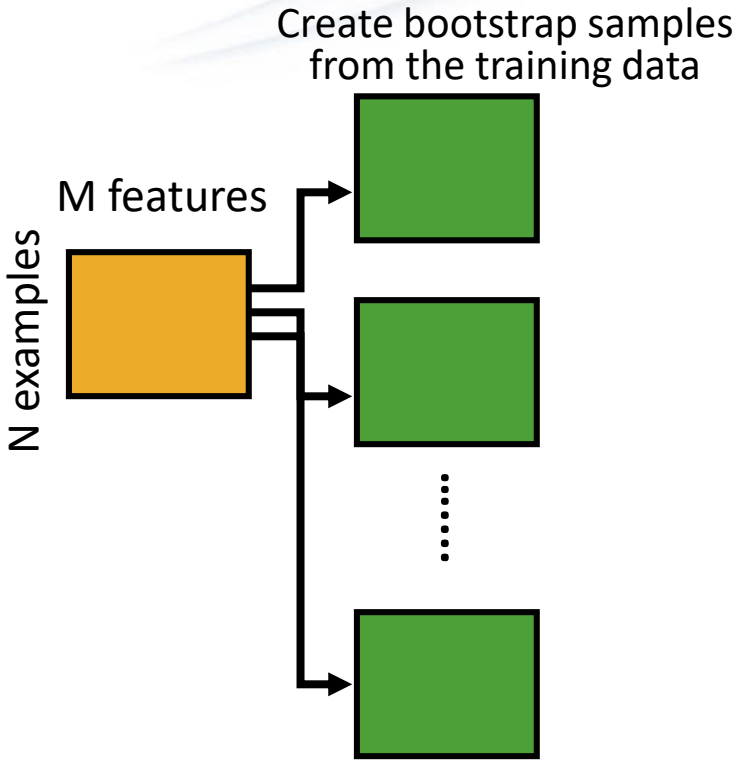
- Random forest classifier, an extension to
- bagging which uses *de-correlated* trees.

Random Forest Classifier

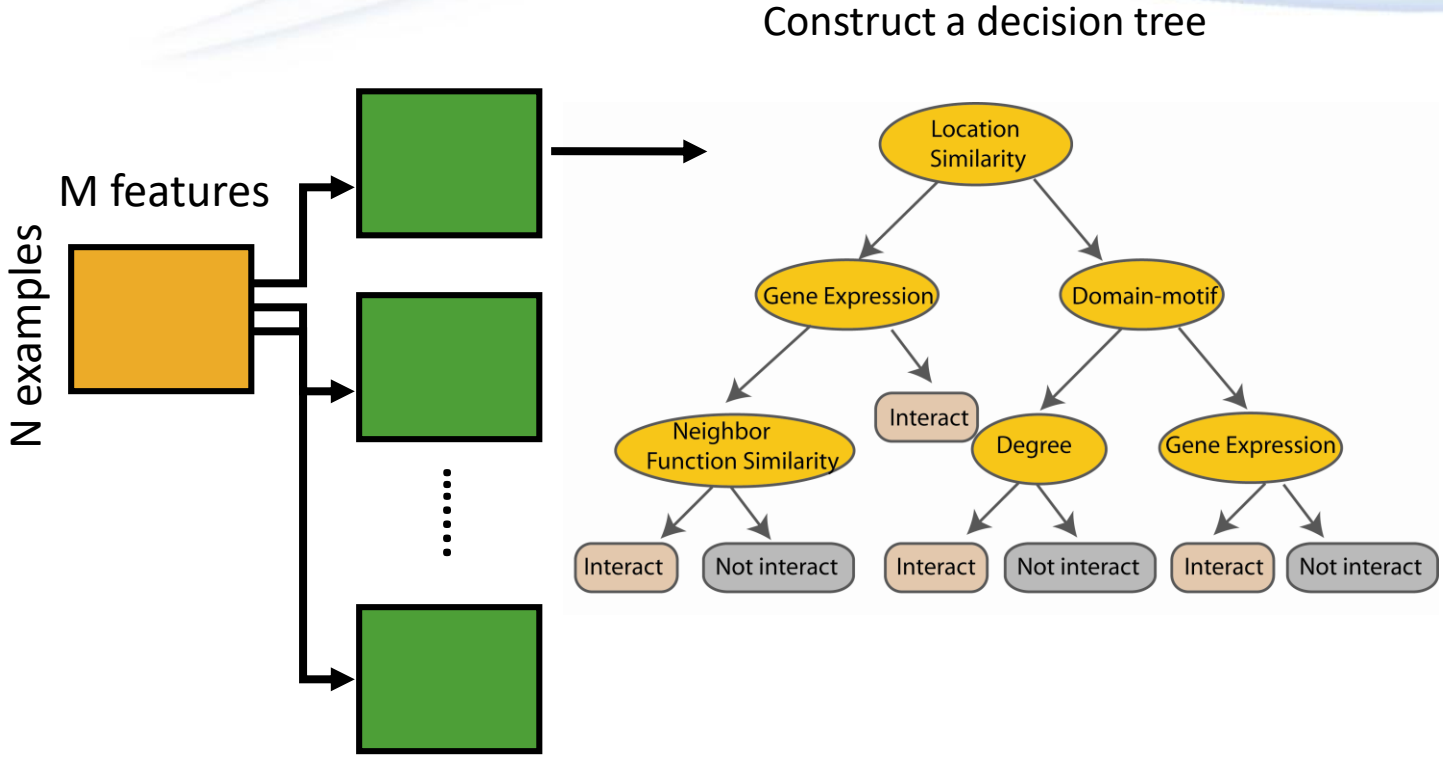
Training Data



Random Forest Classifier

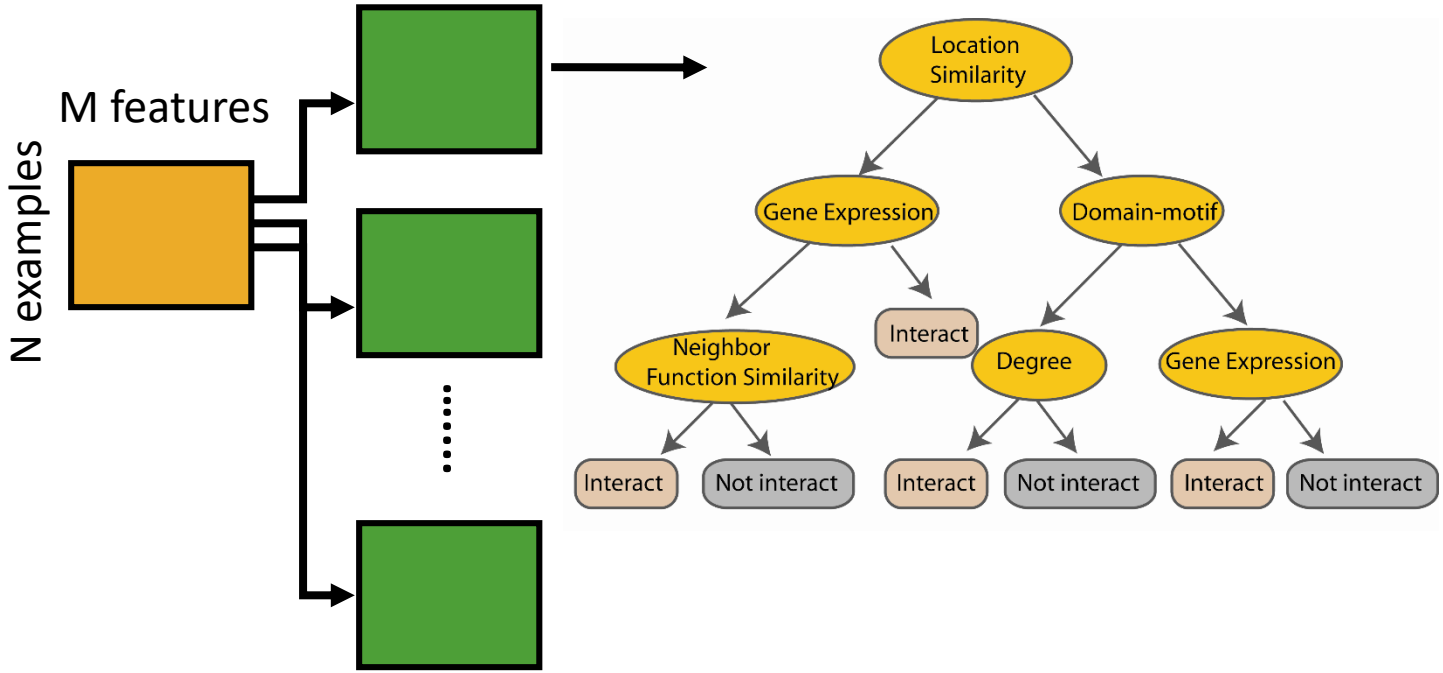


Random Forest Classifier

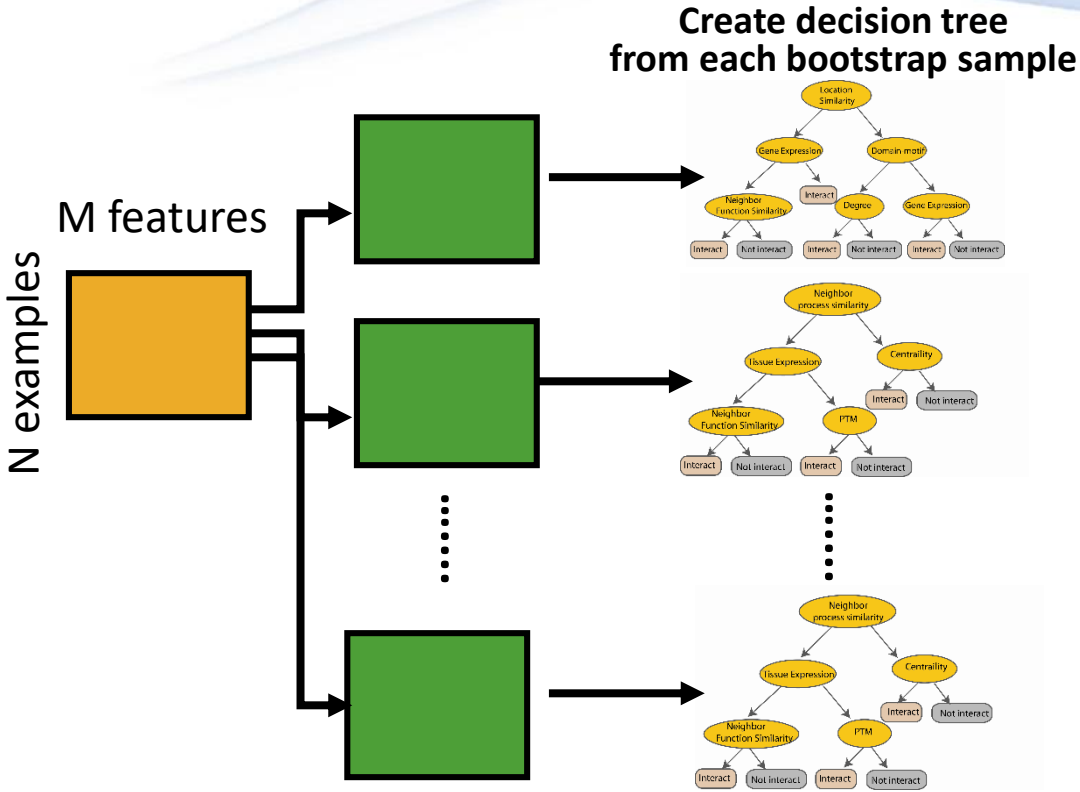


Random Forest Classifier

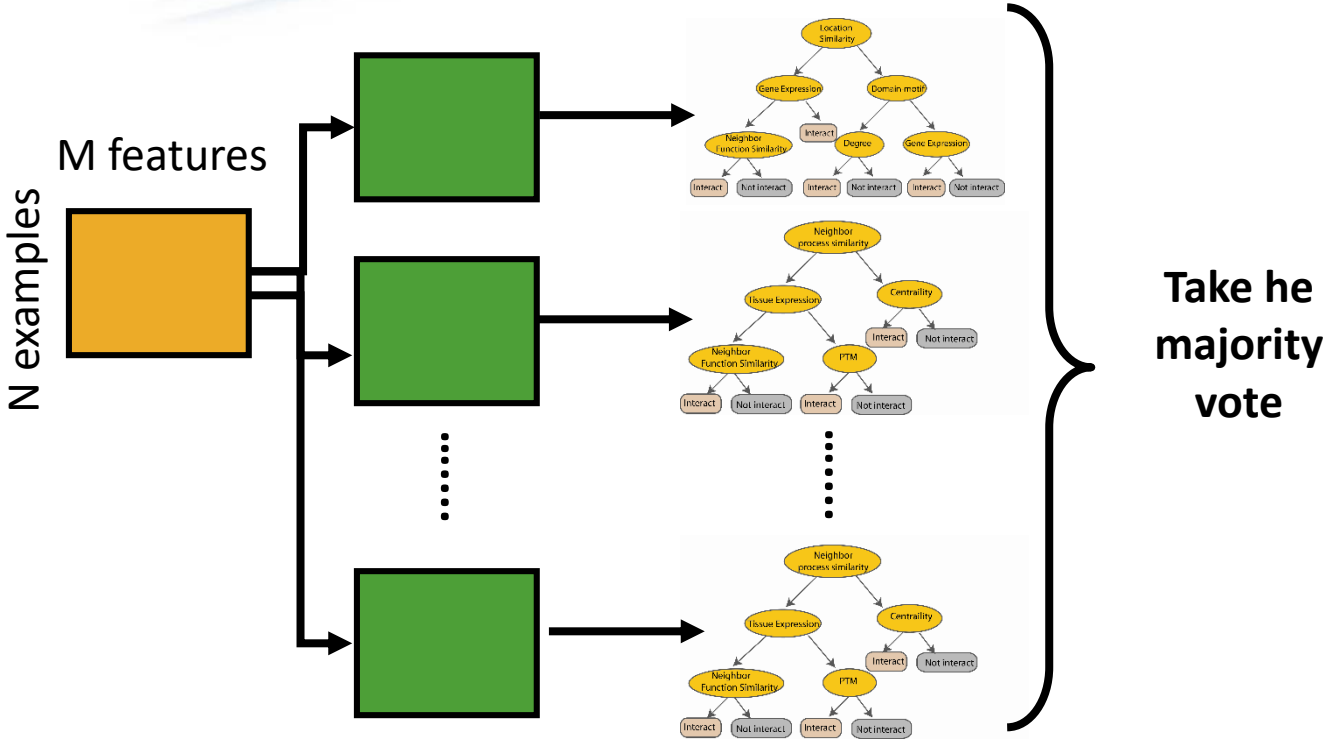
At each node in choosing the split feature
choose only among $m<M$ features



Random Forest Classifier



Random Forest Classifier



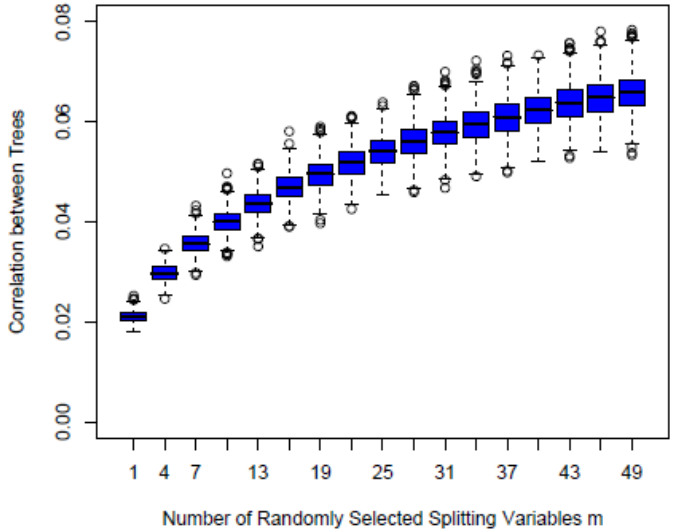


FIGURE 15.9. *Correlations between pairs of trees drawn by a random-forest regression algorithm, as a function of m. The boxplots represent the correlations at 600 randomly chosen prediction points x.*

Random forest

- Available package:
- http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

- To read more:

- http://www-stat.stanford.edu/~hastie/Papers/ESLII.pdf

Adaboost

- The AdaBoost Algorithm begins by training a decision tree in which each observation is assigned an equal weight. After evaluating the first tree, we increase the weights of those observations that are difficult to classify and lower the weights for those that are easy to classify. The second tree is therefore grown on this weighted data. Here, the idea is to improve upon the predictions of the first tree.
- Our new model is therefore Tree 1 + Tree 2. We then compute the classification error from this new 2-tree ensemble model and grow a third tree to predict the revised residuals. We repeat this process for a specified number of iterations. Subsequent trees help us to classify observations that are not well classified by the previous trees. Predictions of the final ensemble model is therefore the weighted sum of the predictions made by the previous tree models.

In short Ada-boost

- retrains the algorithm iteratively by choosing the training set based on accuracy of previous training.
- The weight-age of each trained classifier at any iteration depends on the accuracy achieved.
- **How to assign weight to each classifier?**
- A classifier with 50% accuracy is given a weight of zero, and a classifier with less than 50% accuracy is given negative weight.

Adaboost

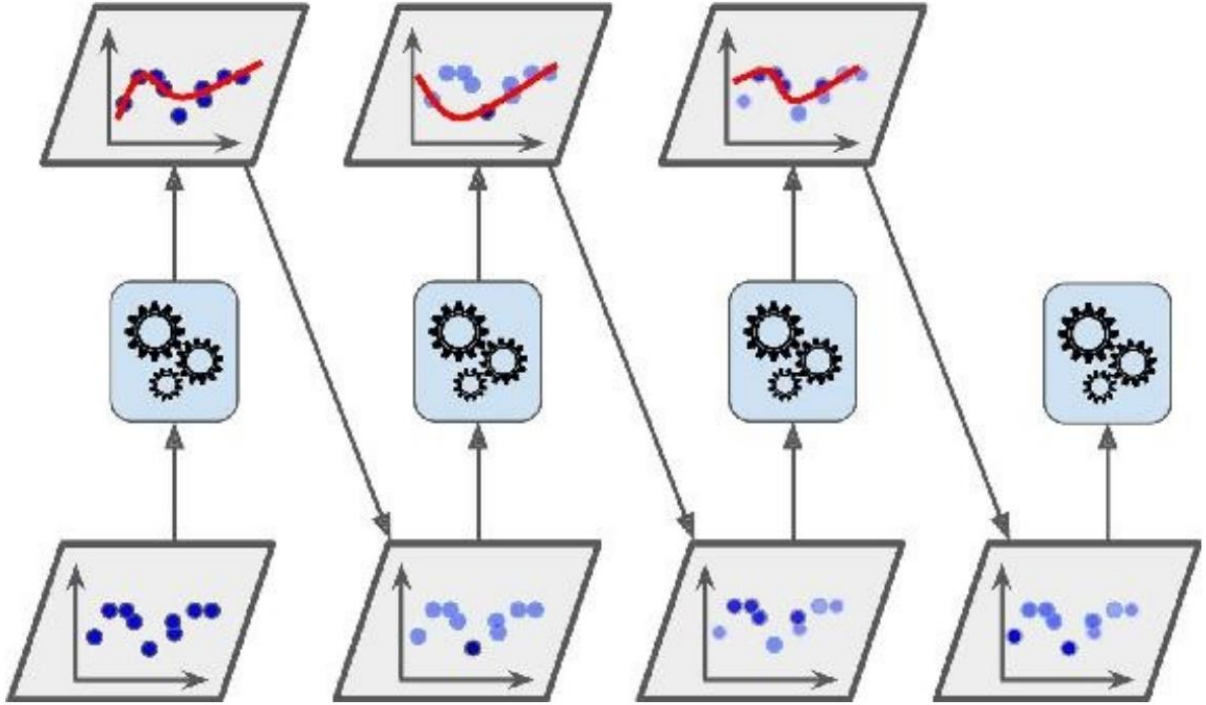
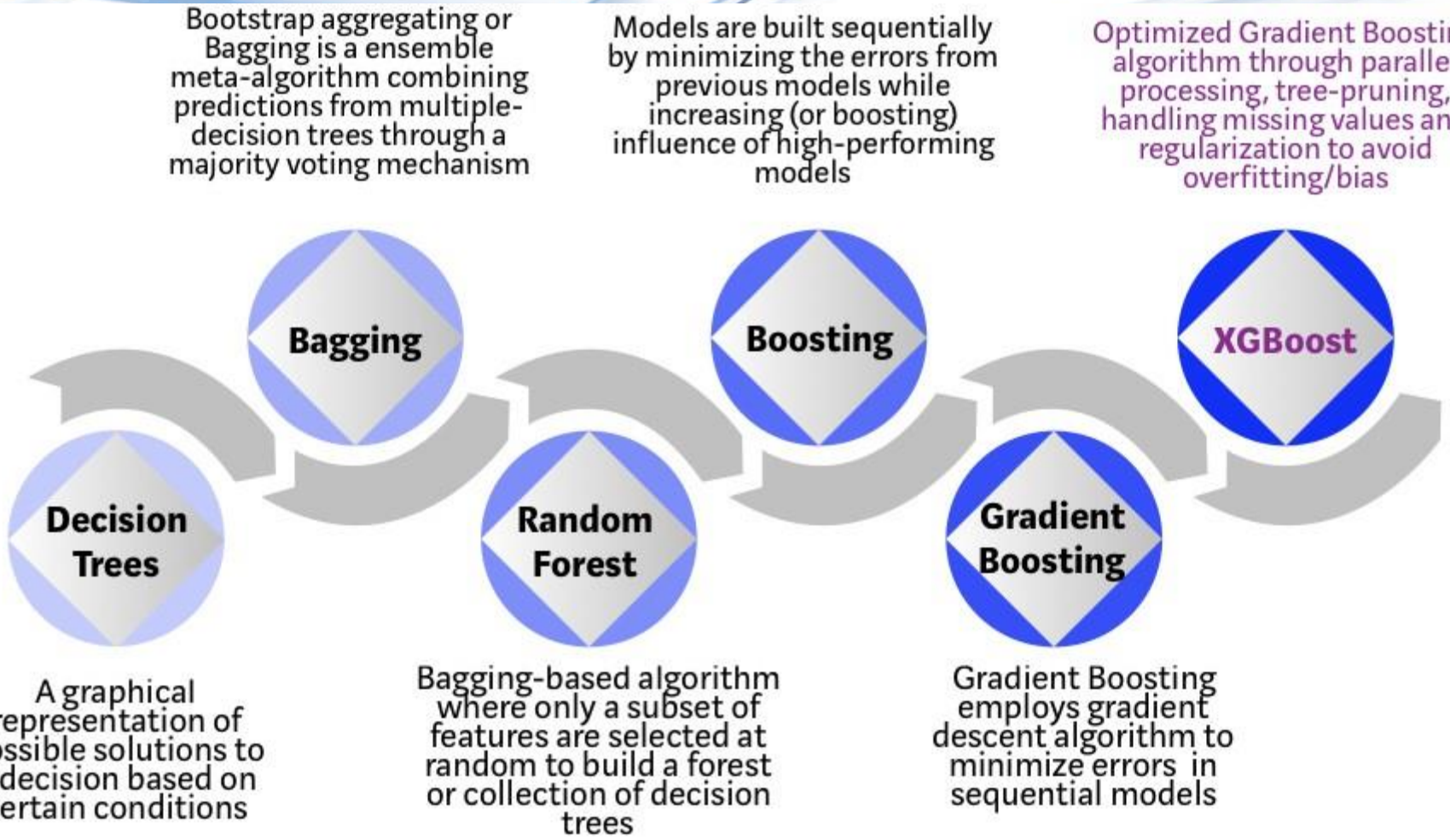
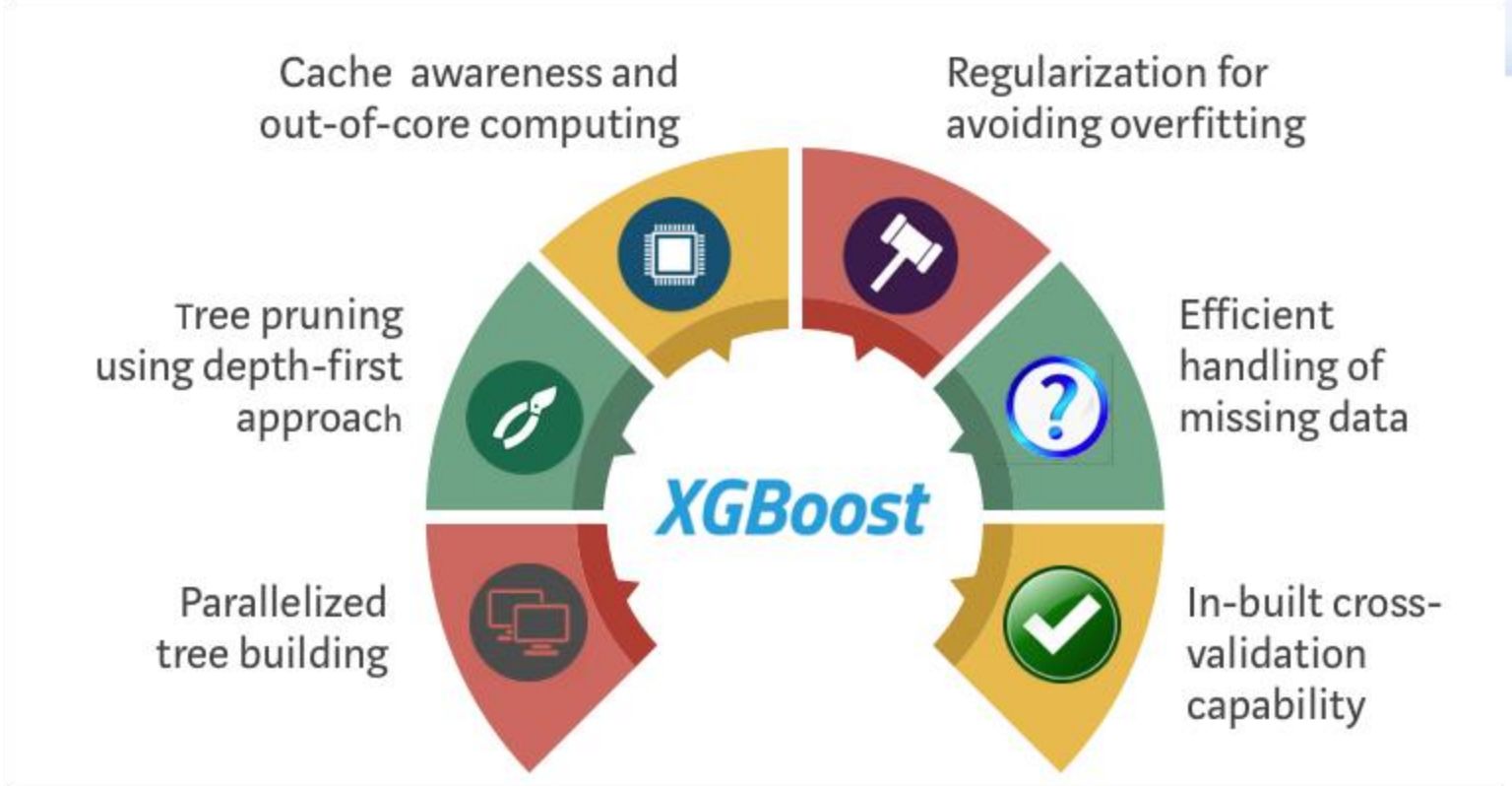


Figure 7-7. AdaBoost sequential training with instance weight updates

Gradient Boosting

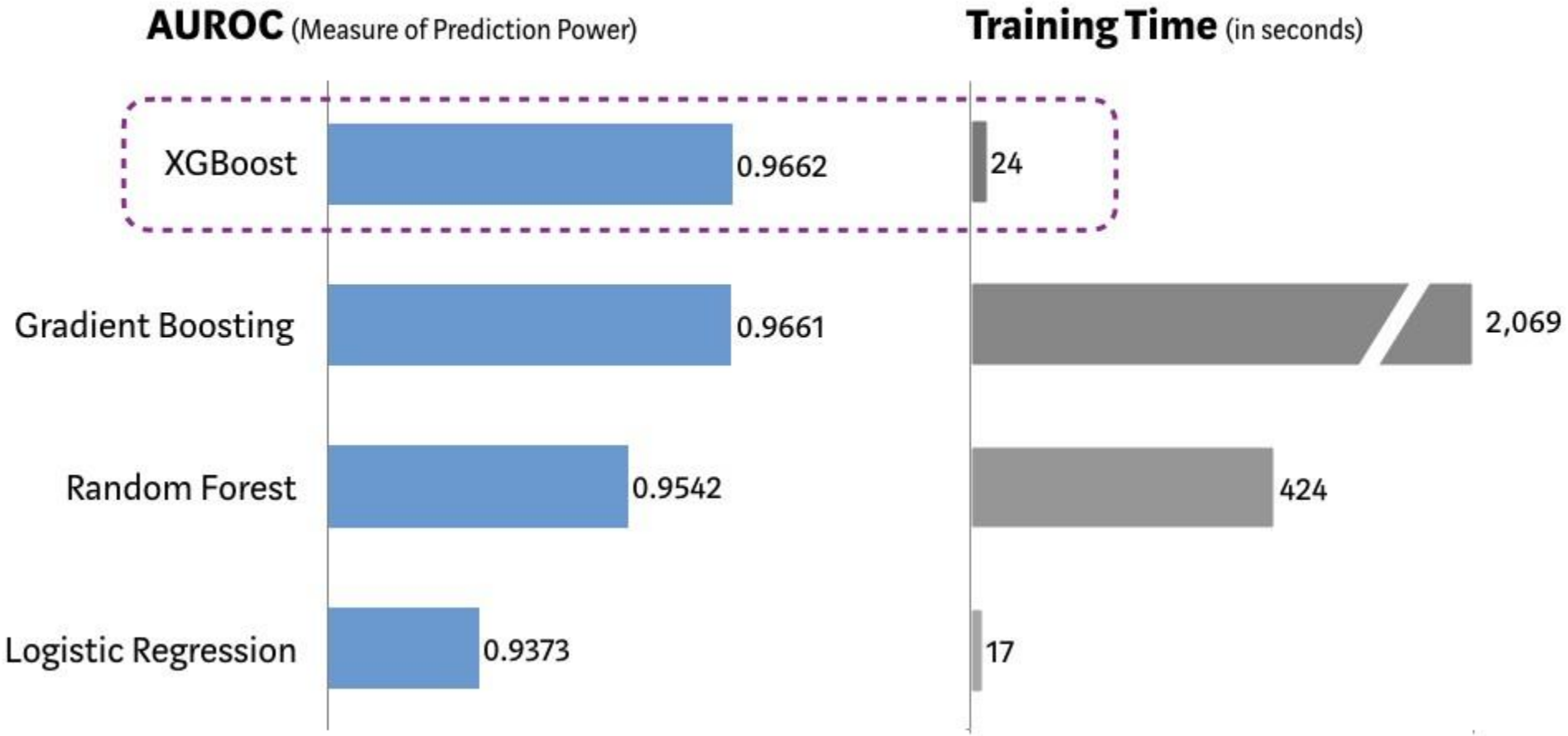
- GBM, short for “Gradient Boosting Machine”, is introduced by Friedman in 2001. It is also known as MART (Multiple Additive Regression Trees) and GBRT (Gradient Boosted Regression Trees).
- Gradient Boosting trains many models in a gradual, additive and sequential manner. The major difference between AdaBoost and Gradient Boosting Algorithm is how the two algorithms identify the shortcomings of weak learners (eg. decision trees). While the AdaBoost model identifies the shortcomings by using high weight data points, gradient boosting performs the same by using gradients in the loss function (*y=ax+b+e , e needs a special mention as it is the error term)*. The loss function is a measure indicating how good are model’s coefficients are at fitting the underlying data. A logical understanding of loss function would depend on what we are trying to optimise.


Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism
Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models
Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias
Bagging
Boosting
XGBoost
Decision Trees
Random Forest
Gradient Boosting
A graphical representation of possible solutions to a decision based on certain conditions
Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees
Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models


Cache awareness and out-of-core computing
Regularization for avoiding overfitting
Tree pruning using depth-first approach
Efficient handling of missing data
XGBoost
Parallelized tree building
In-built cross-validation capability

Performance Comparison using SKLearn's 'Make_Classification' Dataset

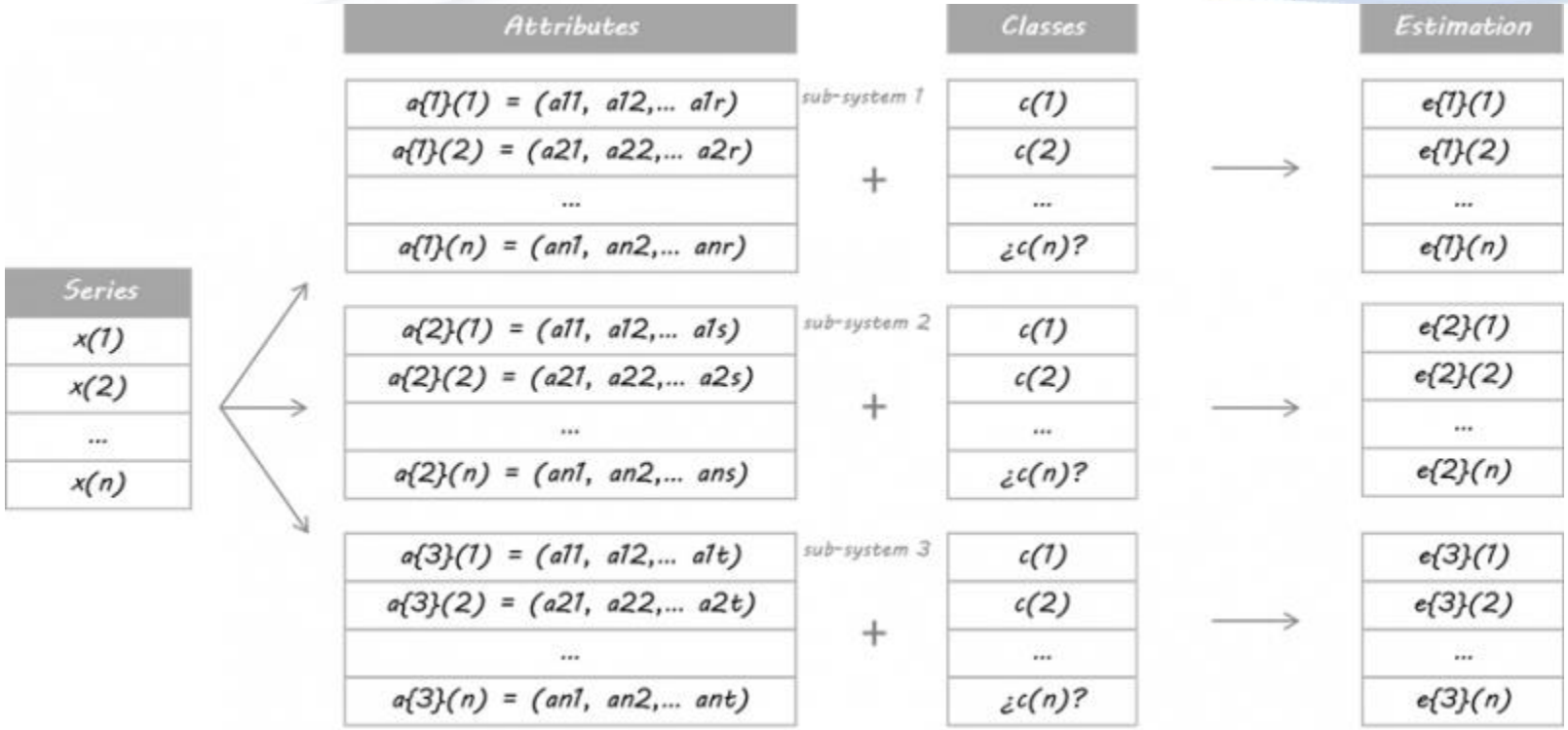
(5 Fold Cross Validation, 1MM randomly generated data sample, 20 features)



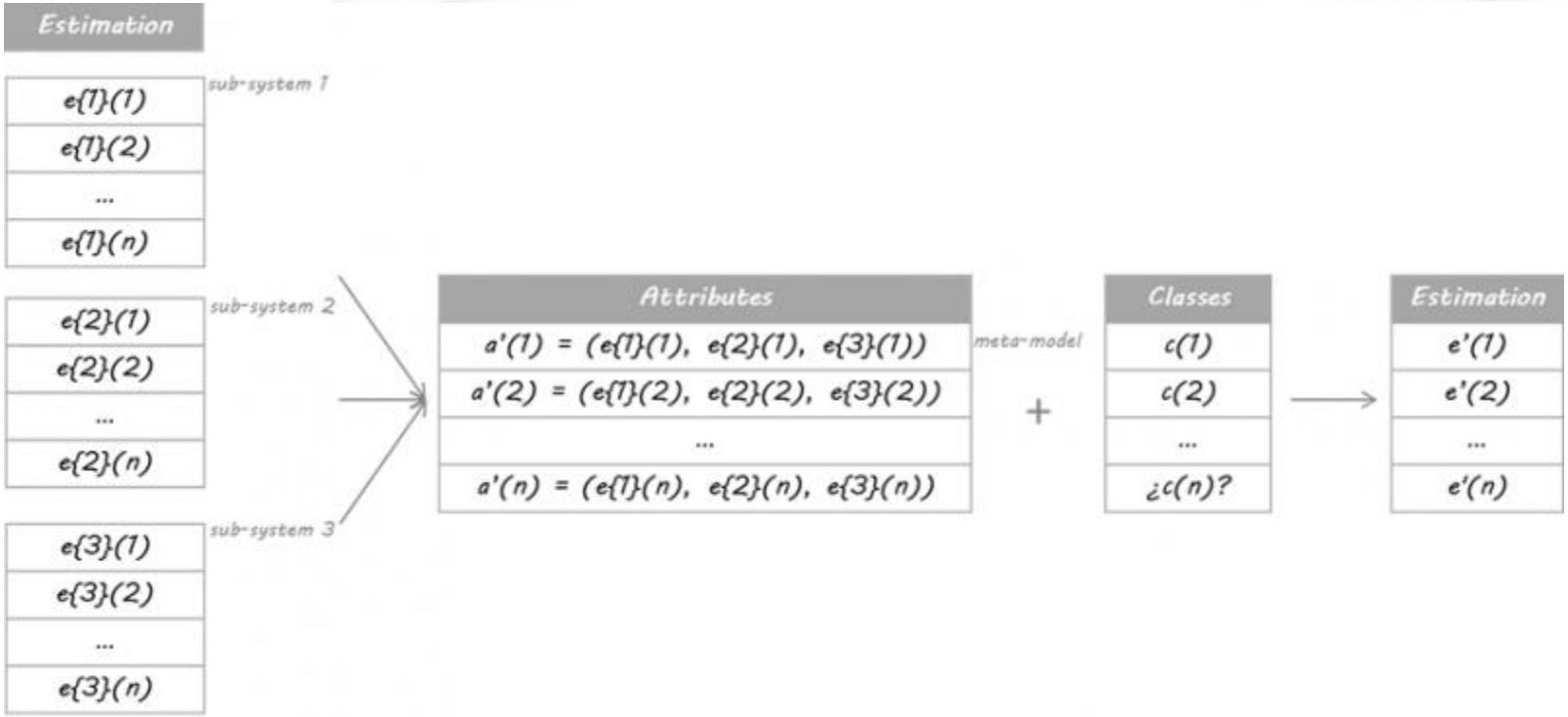
Stacking - Stacked Generalization

- Use many different learners using separate sets of attributes. It does not matter if you use the same learner algorithm or if they share some/all attributes; the key is that they must be different enough in order to guarantee diversification
- The meta-model can be a classification tree, a random forest, a support vector machine… Any classification learner is valid.

Stacking - Stacked Generalization



Stacking - Stacked Generalization



Stacking - Stacked Generalization

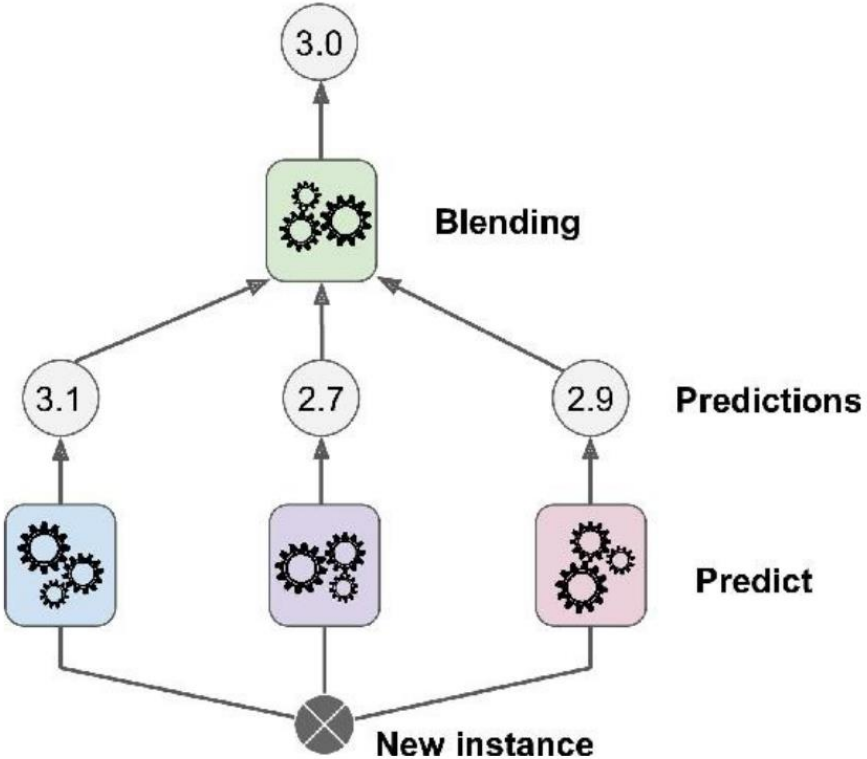


Figure 7-12. Aggregating predictions using a blending predictor