UNIVERSITY OF ENGINEERING & MANAGEMENT, KOLKATA

Course Name : AI & ML

The Ensemble methods

- The ensemble methods, also known as committee-based learning or learning multiple classifier systems train multiple hypotheses to solve the same problem. One of the most common examples of ensemble modeling is the random forest trees where a number of decision trees are used to predict outcomes.
- An ensemble contains a number of hypothesis or learners which are usually generated from training data with the help of a base learning algorithm. Most ensemble methods use a single base learning algorithm to produce homogenous base learners or homogenous ensembles and there are also some other methods which use multiple learning algorithms and thus produce heterogenous ensembles. Ensemble methods are well known for their ability to boost weak learners.

Why Use Ensemble Methods?

- The learning algorithms which output only a single hypothesis tends to suffer from basically three issues. These issues are the statistical problem, the computational problem and the representation problem which can be partly overcome by applying ensemble methods.
- The learning algorithm which suffers from the statistical problem is said to have high variance. The algorithm which exhibits the computational problem is sometimes described as having computational variance and the learning algorithm which suffers from the representational problem is said to have a high bias. These three fundamental issues can be said as the three important ways in which existing learning algorithms fail. The ensemble methods promise of reducing both the bias and the variance of these three shortcomings of the standard learning algorithm.

- Different Techniques
- Some of the commonly used Ensemble techniques are discussed below

Bagging

- Bagging or Bootstrap Aggregation is a powerful, effective and simple ensemble method. The method uses multiple versions of a training set by using the bootstrap, i.e. sampling with replacement and t it can be used with any type of model for classification or regression. Bagging is only effective when using unstable (i.e. a small change in the training set can cause a significant change in the model) non-linear models.

Boosting

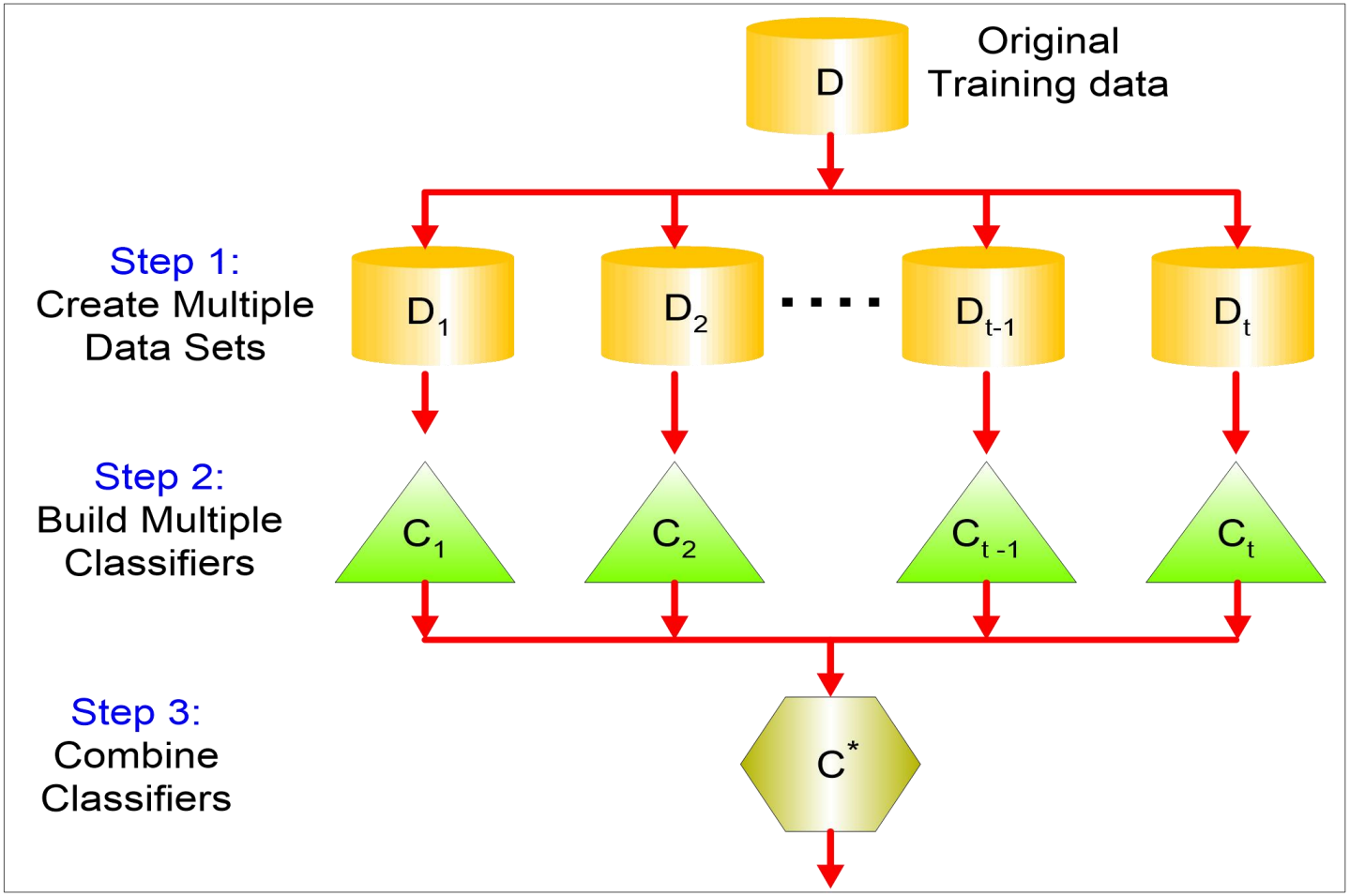
- Boosting is a meta-algorithm which can be viewed as a model averaging method. It is the most widely used ensemble method and one of the most powerful learning ideas. This method was originally designed for classification but it can also be profitably extended to regression. The original boosting algorithm combined three weak learners to generate a strong learner.

Stacking

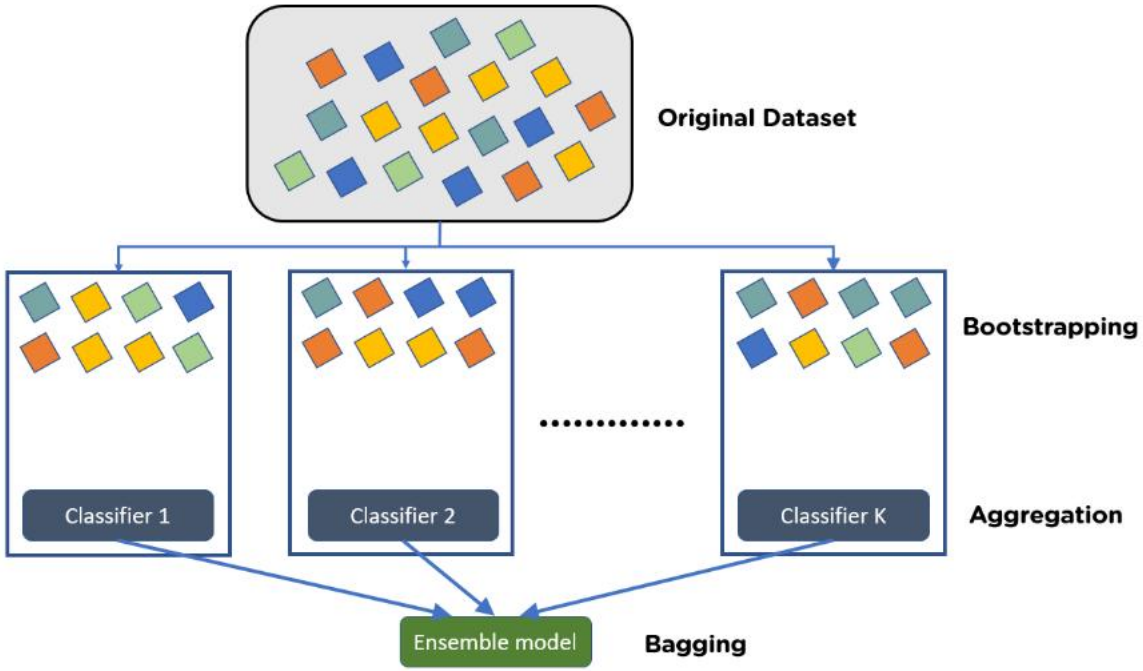
- Stacking is concerned with combining multiple classifiers generated by using different learning algorithms on a single dataset which consists of pairs of feature vectors and their classifications. This technique consists of basically two phases, in the first phase, a set of base-level classifiers is generated and in the second phase, a meta-level classifier is learned which combines the outputs of the base-level classifiers.

Applications Of Ensemble Methods

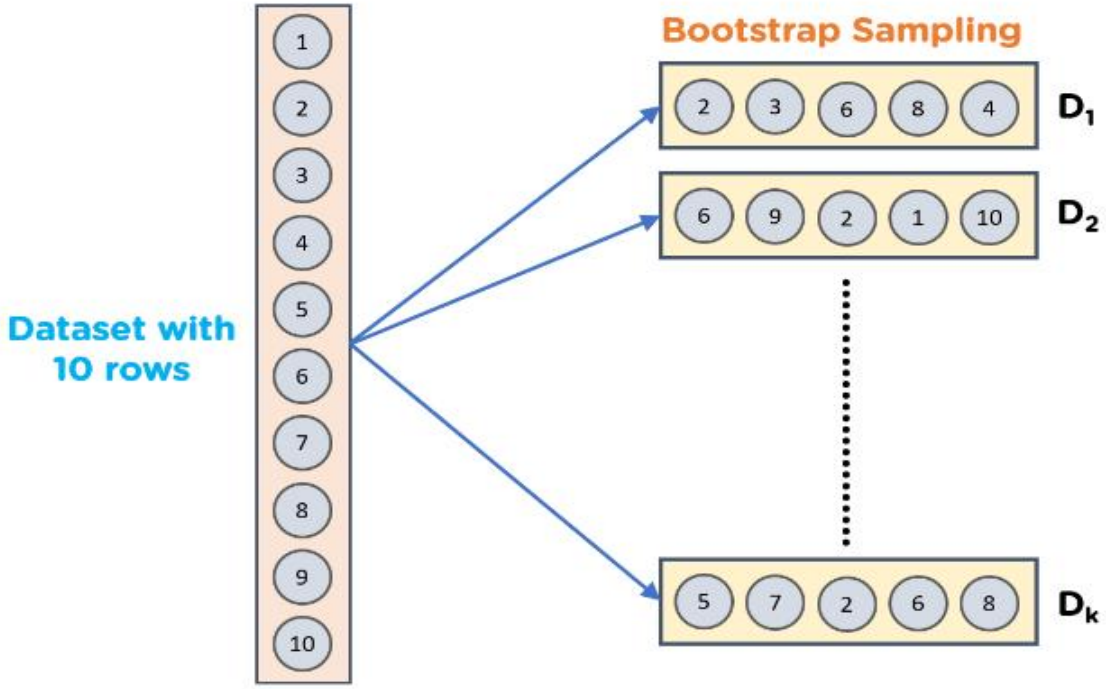
- Ensemble methods can be used as overall diagnostic procedures for a more conventional model building. The larger the difference in fit quality between one of the stronger ensemble methods and a conventional statistical model, the more information that the conventional model is probably missing.
- Ensemble methods can be used to evaluate the relationships between explanatory variables and the response in conventional statistical models. Predictors or basis functions overlooked in a conventional model may surface with an ensemble approach.
- With the help of the ensemble method, the selection process could be better captured and the probability of membership in each treatment group estimated with less bias.
- One could use ensemble methods to implement the covariance adjustments inherent in multiple regression and related procedures. One would "residualized" the response and the predictors of interest with ensemble methods.


D
Original
Training data
Step 1:
Create Multiple
Data Sets
D_1
D_2
D_{t-1}
D_t
Step 2:
Build Multiple
Classifiers
C_1
C_2
C_{t-1}
C_t
Step 3:
Combine
Classifiers
C^*

- How to generate an ensemble of classifiers?
  - Bagging
  - Boosting


Original Dataset
Bootstrapping
Classifier 1
Classifier 2
Classifier K
Aggregation
Ensemble model
Bagging

Bootstrapping is the method of randomly creating samples of data out of a population with replacement to estimate a population parameter.



Steps to Perform Bagging

- Consider there are n observations and m features in the training set. You need to select a random sample from the training dataset without replacement
- A subset of m features is chosen randomly to create a model using sample observations
- The feature offering the best split out of the lot is used to split the nodes
- The tree is grown, so you have the best root nodes
- The above steps are repeated n times. It aggregates the output of individual decision trees to give the best prediction

Advantages of Bagging in Machine Learning

- Bagging minimizes the overfitting of data
- It improves the model's accuracy
- It deals with higher dimensional data efficiently

- Bagging
- Sampling with replacement

Training Data

Data ID

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

- Build classifier on each bootstrap sample
- Each sample has probability $(1 - 1/n)^n$ of being selected as test data
- Training data = $1- (1 - 1/n)^n$ of the original data

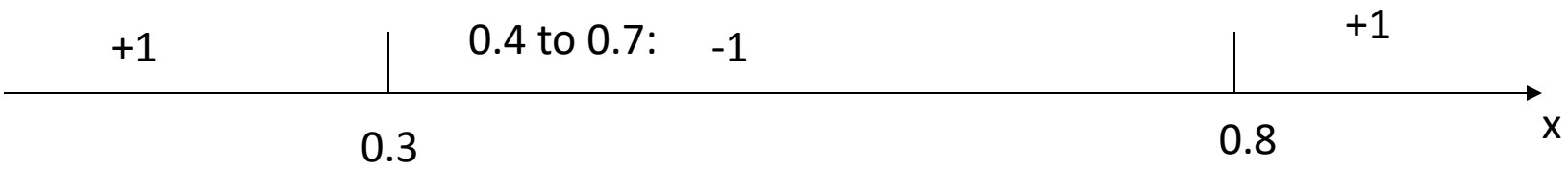
- This method is also called the *0.632 bootstrap*
  - A particular training data has a probability of 1-1/*n* of *not* being picked
  - Thus its probability of ending up in the test data (not selected) is:

$$\left(1-\frac{1}{n}\right)^{n} \approx e^{-1} = 0.368$$

  - This means the training data will contain approximately 63.2% of the instances

14

Bagging Example : Assume that the training data is:



Goal: find a collection of 10 simple thresholding classifiers that collectively can classify correctly.

-Each simple (or weak) classifier is:

(x<=K → class = +1 or -1 depending on which value yields the lowest error; where K is determined by entropy minimization)

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.8	0.9	1	1	1
y	1	1	1	-1	-1	1	1	1	1	1

x <= 0.65 ==> y = 1
x > 0.65 ==> y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ==> y = 1
x > 0.3 ==> y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ==> y = -1
x > 0.05 ==> y = 1

Figure 5.35. Example of bagging.

Bagging (applied to training data)

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1
True Class	1	1	1	-1	-1	-1	-1	1	1	1

Figure 5.36. Example of combining classifiers constructed using the bagging approach.

Accuracy of ensemble classifier: 100% ☺

Bagging Summary

- Works well if the base classifiers are unstable (complement each other)
- Increased accuracy because it ***reduces the variance*** of the individual classifier
- Does not focus on any particular instance of the training data
  - Therefore, less susceptible to model over-fitting when applied to noisy data
- What if we want to focus on a particular instances of training data?

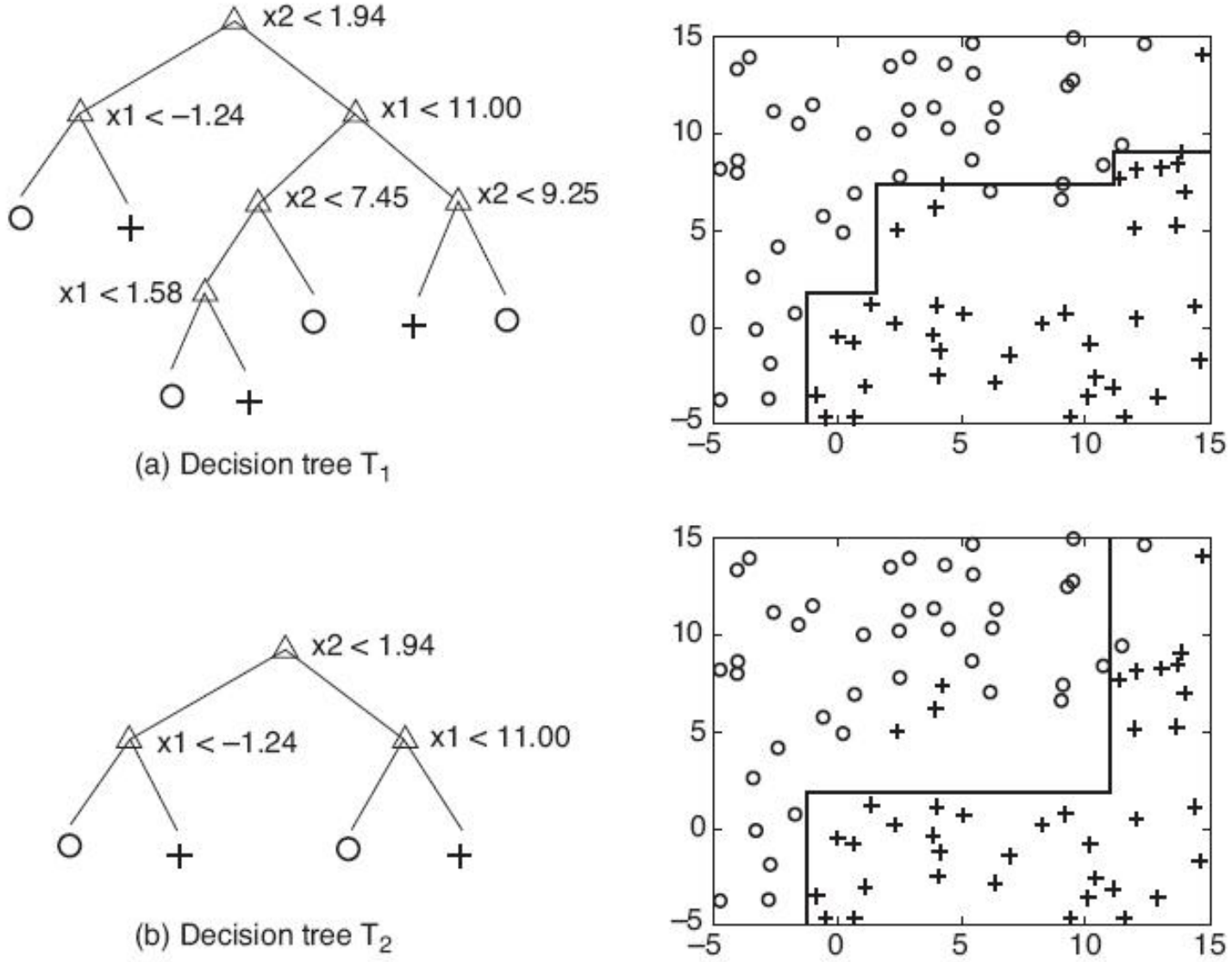
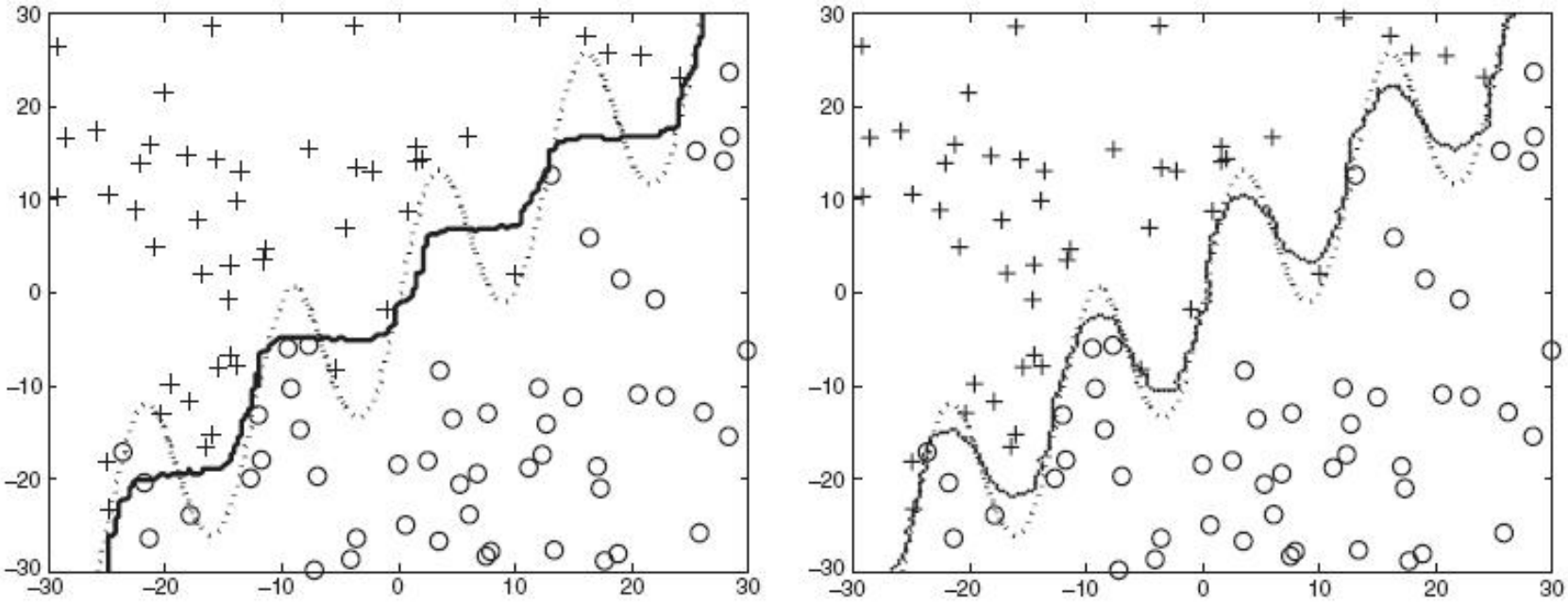


Figure 5.33. Two decision trees with different complexities induced from the same training data.

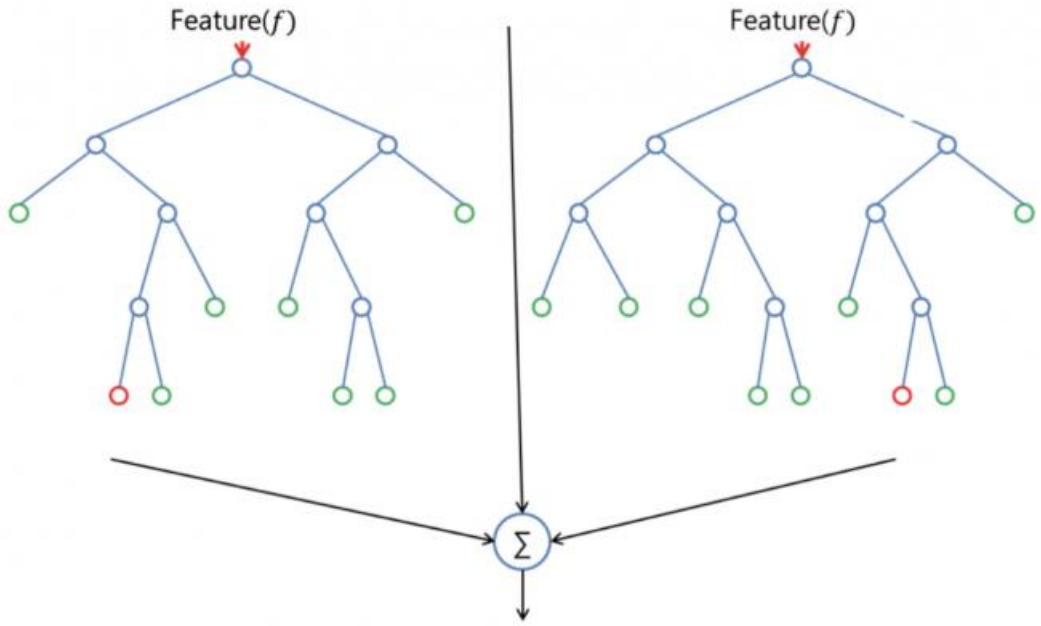


(a) Decision boundary for decision tree.

(b) Decision boundary for 1-nearest neighbor.

Figure 5.34. Bias of decision tree and 1-nearest neighbor classifiers.

- **WHAT IS RANDOM FOREST?**
- Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.
- **Put simply: random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.**


Feature(f)
Feature(f)
Σ

- Random forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.
- Therefore, in random forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. You can even make trees more random by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

DIFFERENCE BETWEEN DECISION TREES AND RANDOM FORESTS

- While random forest is a collection of decision trees, there are some differences.
- If you input a training dataset with features and labels into a decision tree, it will formulate some set of rules, which will be used to make the predictions.
- For example, to predict whether a person will click on an online advertisement, you might collect the ads the person clicked on in the past and some features that describe his/her decision. If you put the features and labels into a decision tree, it will generate some rules that help predict whether the advertisement will be clicked or not. In comparison, the random forest algorithm randomly selects observations and features to build several decision trees and then averages the results.
- Another difference is "deep" decision trees might suffer from overfitting. Most of the time, random forest prevents this by creating random subsets of the features and building smaller trees using those subsets. Afterwards, it combines the subtrees. It's important to note this doesn't work every time and it also makes the computation slower, depending on how many trees the random forest builds.

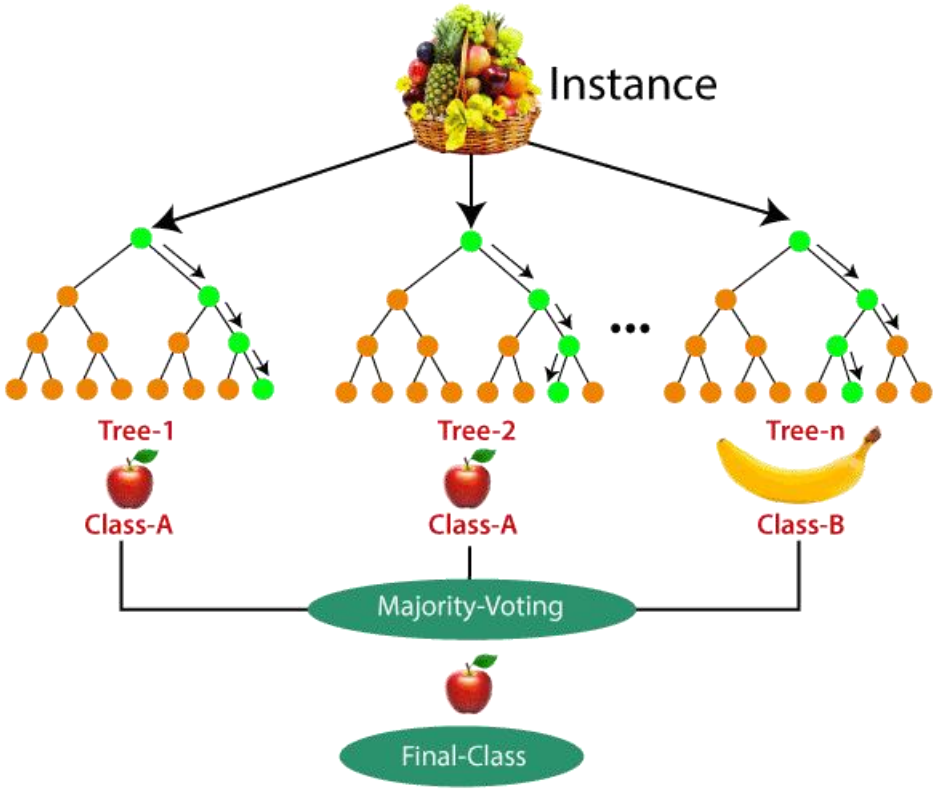
- **ADVANTAGES AND DISADVANTAGES OF THE RANDOM FOREST ALGORITHM**

- One of the biggest advantages of random forest is its versatility. It can be used for both regression and classification tasks, and it's also easy to view the relative importance it assigns to the input features.
- Random forest is also a very handy algorithm because the default hyperparameters it uses often produce a good prediction result. Understanding the hyperparameters is pretty straightforward, and there's also not that many of them.
- One of the biggest problems in machine learning is overfitting, but most of the time this won't happen thanks to the random forest classifier. If there are enough trees in the forest, the classifier won't overfit the model.
- The main limitation of random forest is that a large number of trees can make the algorithm too slow and ineffective for real-time predictions. In general, these algorithms are fast to train, but quite slow to create predictions once they are trained. A more accurate prediction requires more trees, which results in a slower model. In most real-world applications, the random forest algorithm is fast enough but there can certainly be situations where run-time performance is important and other approaches would be preferred.
- And, of course, random forest is a predictive modeling tool and not a descriptive tool, meaning if you're looking for a description of the relationships in your data, other approaches would be better.

- How does Random Forest algorithm work?
- Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

- **Step-1:** Select random K data points from the training set.
- **Step-2:** Build the decision trees associated with the selected data points (Subsets).
- **Step-3:** Choose the number N for decision trees that you want to build.
- **Step-4:** Repeat Step 1 & 2.
- **Step-5:** For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.


Instance
Tree-1
Tree-2
Tree-n
Class-A
Class-A
Class-B
Majority-Voting
Final-Class

- Applications of Random Forest
- There are mainly four sectors where Random forest mostly used:
- **Banking:** Banking sector mostly uses this algorithm for the identification of loan risk.
- **Medicine:** With the help of this algorithm, disease trends and risks of the disease can be identified.
- **Land Use:** We can identify the areas of similar land use by this algorithm.
- **Marketing:** Marketing trends can be identified using this algorithm.

- **Lazy learners**
- Lazy learners simply store the training data and wait until a testing data appear. When it does, classification is conducted based on the most related data in the stored training data. Compared to eager learners, lazy learners have less training time but more time in predicting.
- *Ex. k-nearest neighbor, Case-based reasoning*
- **Eager learners**
- Eager learners construct a classification model based on the given training data before receiving data for classification. It must be able to commit to a single hypothesis that covers the entire instance space. Due to the model construction, eager learners take a long time for train and less time to predict.
- *Ex. Decision Tree, Naive Bayes, Artificial Neural Networks*

- **k-Nearest-Neighbor Method**
  - first described in the early 1950s
  - It has since been widely used in the area of pattern recognition.
  - The training instances are described by ***n* attributes**.
  - Each instance represents a point in an ***n-dimensional* space**.
  - A **k-nearest-neighbor classifier** searches the pattern space for the **k training instances** that are closest to the unknown instance.

- The nearest neighbor can be defined in terms of Euclidean distance, dist($\mathbf{X_1}$, $\mathbf{X_2}$)
- The Euclidean distance between two points or instances, say, X1 = (x_{11}, x_{12}, … , x_{1n}) and X_2 = (x_{21}, x_{22}, … , x_{2n}), is:

$$dist(\boldsymbol{X_1}, \boldsymbol{X_2}) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}$$

  - Nominal attributes: distance either 0 or 1
  - Refer to cluster analysis for more distance metrics

KNN Example

Sepal Length	Sepal Width	Species
5.3	3.7	Setosa
5.1	3.8	Setosa
7.2	3.0	Virginica
5.4	3.4	Setosa
5.1	3.3	Setosa
5.4	3.9	Setosa
7.4	2.8	Virginica
6.1	2.8	Verscicolor
7.3	2.9	Virginica
6.0	2.7	Verscicolor
5.8	2.8	Virginica
6.3	2.3	Verscicolor
5.1	2.5	Verscicolor
6.3	2.5	Verscicolor
5.5	2.4	Verscicolor

Sepal Length	Sepal Width	Species
5.2	3.1	?

Continued…

Sepal Length	Sepal Width	Species	Distance
5.3	3.7	Setosa	0.608
5.1	3.8	Setosa	0.707
7.2	3.0	Virginica	2.002
5.4	3.4	Setosa	0.36
5.1	3.3	Setosa	0.22
5.4	3.9	Setosa	0.82
7.4	2.8	Virginica	2.22
6.1	2.8	Verscicolor	0.94
7.3	2.9	Virginica	2.1
6.0	2.7	Verscicolor	0.89
5.8	2.8	Virginica	0.67
6.3	2.3	Verscicolor	1.36
5.1	2.5	Verscicolor	0.60
6.3	2.5	Verscicolor	1.25
5.5	2.4	Verscicolor	0.75

- For k-nearest-neighbor classification, the unknown instance is assigned the most common class among its *k* nearest neighbors.
- When *k = 1,* the unknown instance is assigned the class of the training instance that is closest to it in pattern space.
- Nearest-neighbor classifiers can also be used for prediction, that is, to return a real-valued prediction for a given unknown instance.
  - In this case, the classifier returns the average value of the real-valued labels associated with the *k nearest neighbors of the unknown* instance.

- **Distances for categorical attributes**:
  - A simple method is to compare the corresponding value of the attribute in instance X1 with that in instance X2.
  - If the two are identical (e.g., instances X1 and X2 both have the color *blue*), then the difference between the two is taken as 0, otherwise 1.
  - Other methods may incorporate more sophisticated schemes for differential grading (e.g., where a difference score is assigned, say, for blue and white than for blue and black).

- **Handling missing values:**
  - In general, if the value of a given attribute A is missing in instance X1 and/or in instance X2, we assume the maximum possible difference.
  - For categorical attributes, we take the difference value to be 1 if either one or both of the corresponding values of A are missing.
  - If A is numeric and missing from both instances X1 and X2, then the difference is also taken to be 1.
    - If only one value is missing and the other (which we'll call v') is present and normalized, then we can take the difference to be either |1 - v'| or |0 – v'| , whichever is greater.

- Typically, we normalize the values of each attribute in advanced.
- This helps prevent attributes with initially large ranges (such as *income*) from outweighing attributes with initially smaller ranges (such as binary attributes).
- Min-max normalization:

$$v' = \frac{v - min_A}{max_A - min_A}$$

  - all attribute values lie between 0 and 1
  - For more information on normalization methods refer to data preprocessing section

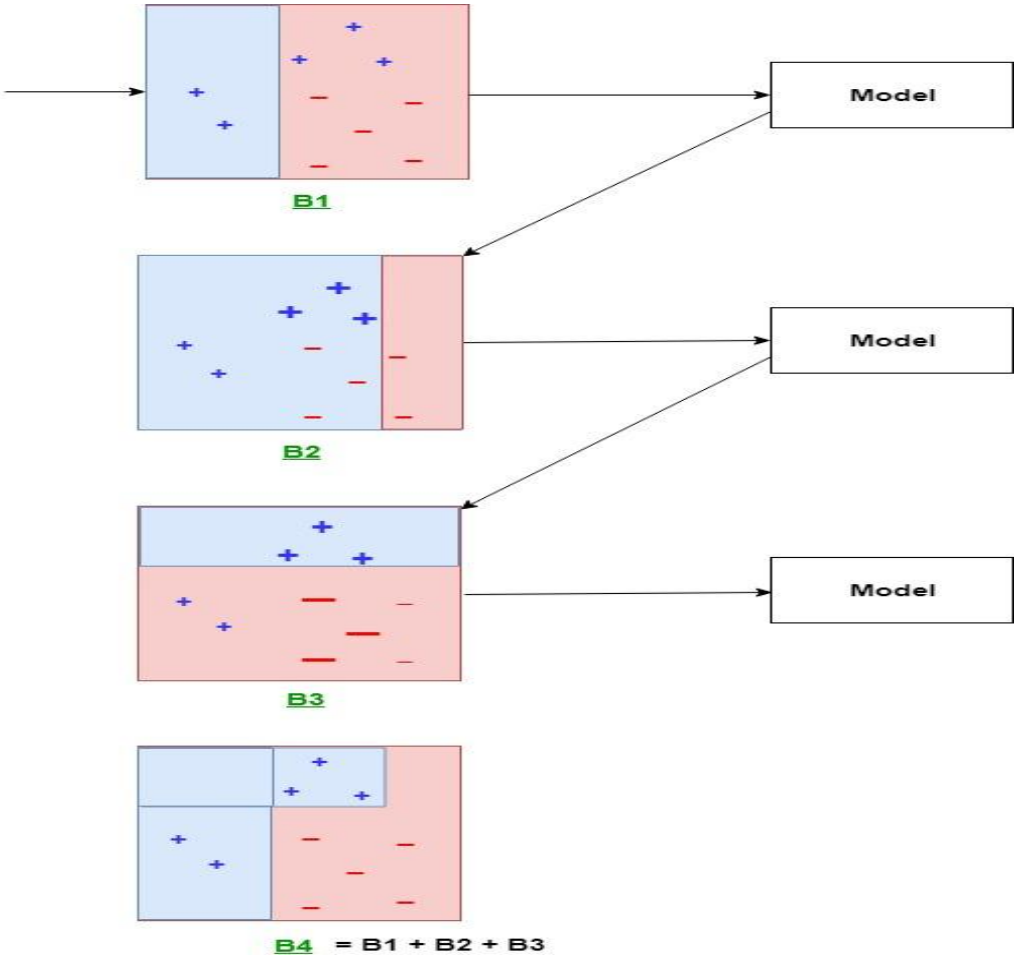
- **Determining a good value for k:**
  - k can be determined experimentally.
  - Starting with *k = 1,* we use a test set to estimate the error rate of the classifier.
  - This process can be repeated each time by incrementing *k* to allow for one more neighbor.
  - The *k* value that gives the minimum error rate may be selected.
  - In general, the larger the number of training instances is, the larger the value of *k* will be

- **Boosting**
- An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records
  - Initially, all N records are assigned equal weights
  - Unlike bagging, weights may change at the end of a boosting round

Steps for Boosting

- *Initialise the dataset and assign equal weight to each of the data point.*
- *Provide this as input to the model and identify the wrongly classified data points.*
- *Increase the weight of the wrongly classified data points.*
- *if (got required results)*
  Goto step 5
  else
  Goto step 2

- *End*


Model
B1
Model
B2
Model
B3
B4 = B1 + B2 + B3

- **Explanation:**
  The above diagram explains the AdaBoost algorithm in a very simple way. Let's try to understand it in a stepwise process:
- **B1** consists of 10 data points which consist of two types namely plus(+) and minus(-) and 5 of which are plus(+) and the other 5 are minus(-) and each one has been assigned equal weight initially. The first model tries to classify the data points and generates a vertical separator line but it wrongly classifies 3 plus(+) as minus(-).
- **B2** consists of the 10 data points from the previous model in which the 3 wrongly classified plus(+) are weighted more so that the current model tries more to classify these pluses(+) correctly. This model generates a vertical separator line that correctly classifies the previously wrongly classified pluses(+) but in this attempt, it wrongly classifies three minuses(-).
- **B3** consists of the 10 data points from the previous model in which the 3 wrongly classified minus(-) are weighted more so that the current model tries more to classify these minuses(-) correctly. This model generates a horizontal separator line that correctly classifies the previously wrongly classified minuses(-).
- **B4** combines together B1, B2, and B3 in order to build a strong prediction model which is much better than any individual model used.
-

- Records that are wrongly classified will have their weights increased
- Records that are classified correctly will have their weights decreased

Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

- Example 4 is hard to classify
- Its weight is increased, therefore it is more likely to be chosen again in subsequent rounds

- Equal weights are assigned to each training instance (1/N for round 1) at first round
- After a classifier C_i is learned, the weights are adjusted to allow the subsequent classifier

C_{i+1} to "pay more attention" to data that were misclassified by C_i.

- Final boosted classifier C* combines the votes of each individual classifier
  - Weight of each classifier's vote is a function of its accuracy
- Adaboost – popular boosting algorithm

Adaboost (Adaptive Boost)

- Input:
  - Training set D containing ***N*** instances
  - *T* rounds
  - A classification learning scheme
- Output:
  - A composite model

- **Adaboost: Training Phase**
- Training data D contain N labeled data (X_1,y_1), (X_2,y_2), (X_3,y_3),....(X_N,y_N)
- Initially assign equal weight 1/d to each data
- To generate *T* base classifiers, we need *T* rounds or iterations
- Round i, data from D are sampled with replacement , to form Di (size *N*)
- Each data's chance of being selected in the next rounds depends on its weight
  - Each time the new sample is generated directly from the training data D with different sampling probability according to the weights; these weights are not zero

- Base classifier C_i, is derived from training data of Di
- Error of C_i is tested using Di
- Weights of training data are adjusted depending on how they were classified
  - Correctly classified: Decrease weight
  - Incorrectly classified: Increase weight
- Weight of a data indicates how hard it is to classify it (directly proportional)

Adaboost: Testing Phase

- The lower a classifier error rate, the more accurate it is, and therefore, the higher its weight for voting should be
- Weight of a classifier C_i's vote is $\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$
- Testing:
  - For each class c, sum the weights of each classifier that assigned class c to X (unseen data)
  - The class with the highest sum is the WINNER!

$$C^*(x_{test}) = \arg\max_y \sum_{i=1}^{T} \alpha_i \delta\left(C_i(x_{test}) = y\right)$$

Example: Error and Classifier Weight in AdaBoost

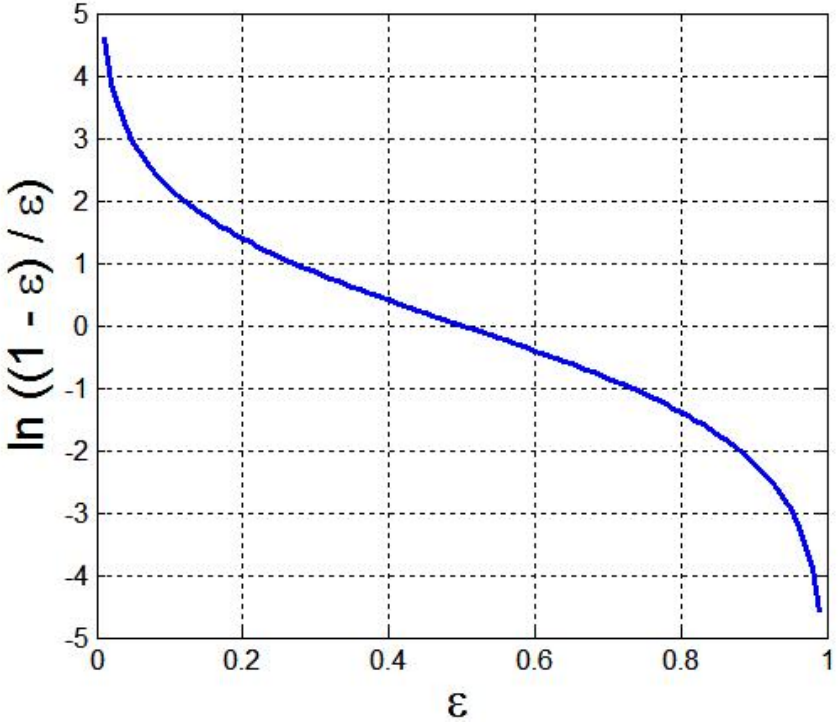
- Base classifiers: C_1, C_2, …, C_T

- Error rate: (*i* = index of classifier, *j*=index of instance)

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

- Importance of a classifier:

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$



Example: Data Instance Weight in AdaBoost

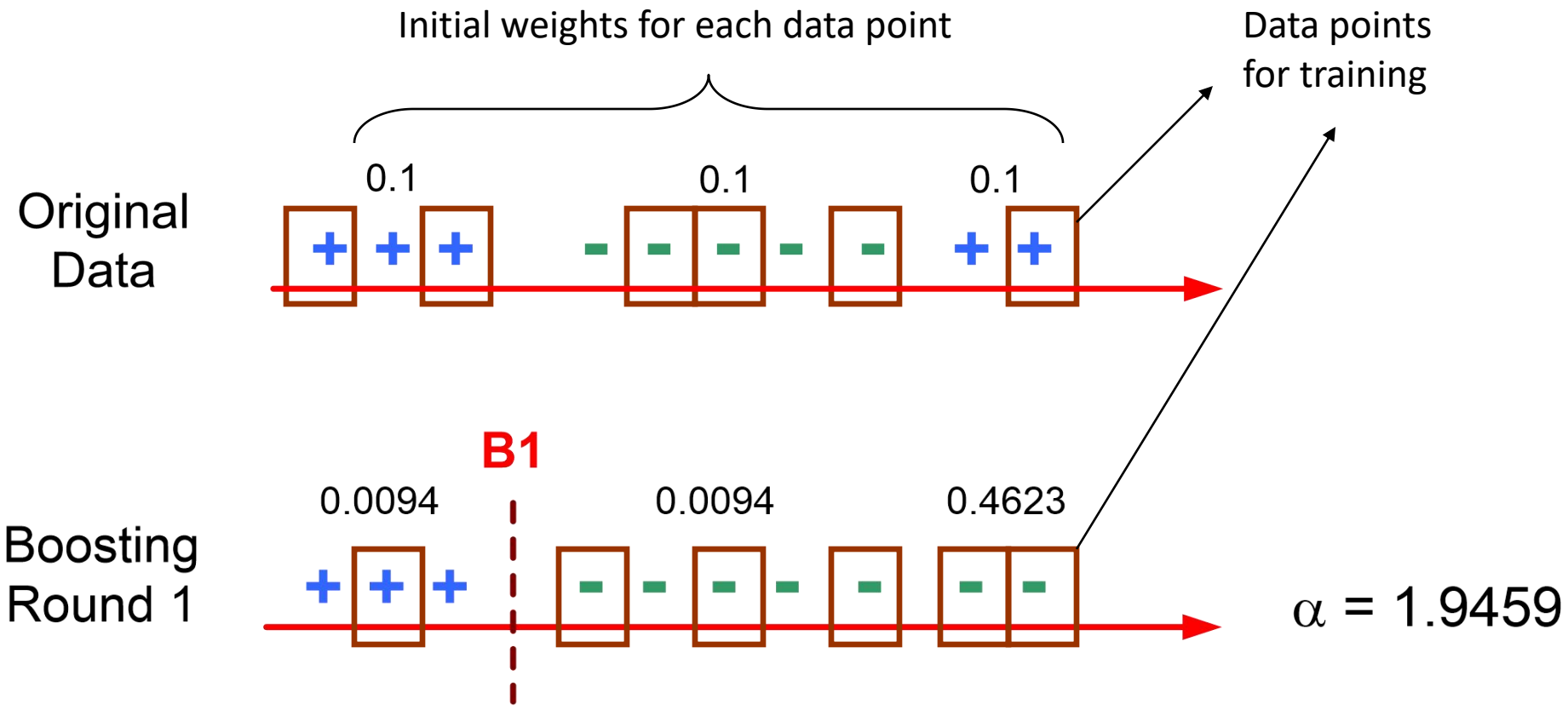
- Assume: *N* training data in D, *T* rounds, (x_j,y_j) are the training data, C_i, a_i are the classifier and weight of the i^{th} round, respectively.
- Weight update on all training data in *D*:

$$w_j^{(i+1)} = \frac{w_j^{(i)}}{Z_i} \begin{cases} \exp^{-\alpha_i} & \text{if } C_i(x_j) = y_j \\ \exp^{\alpha_i} & \text{if } C_i(x_j) \neq y_j \end{cases}$$

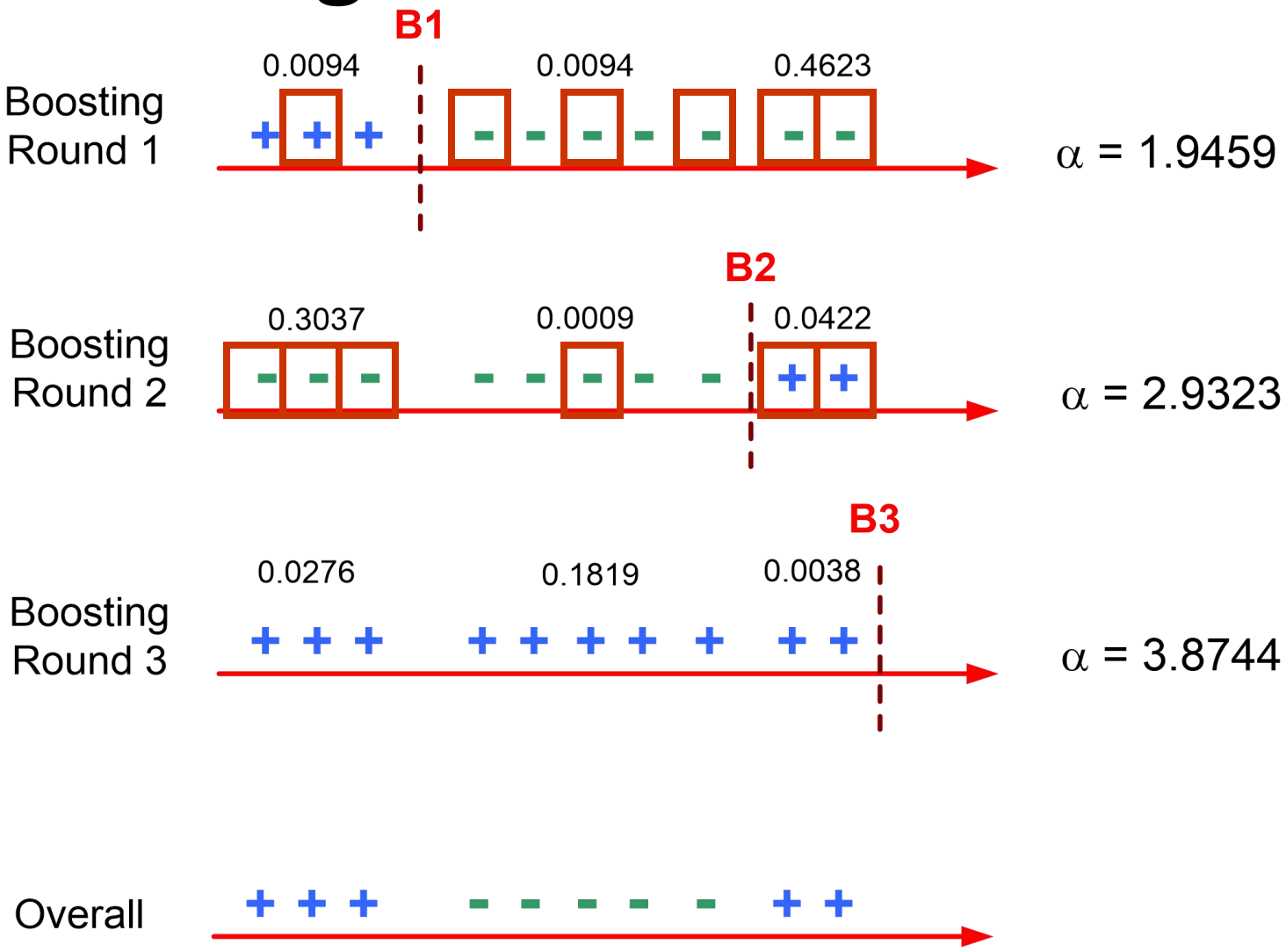
where Z_i is the normalization factor

$$C^*(x_{test}) = \arg\max_y \sum_{i=1}^{T} \alpha_i \delta\left(C_i(x_{test}) = y\right)$$

Illustrating AdaBoost



Illustrating AdaBoost



Thank You