DATA 572: Supervised Learning

2023W2

Shan Du

Tree-Based Methods

- Tree-based methods involve *stratifying* or *segmenting* the predictor space into a number of simple regions.
- In order to make a prediction for a given observation, we typically use the mean or the mode response value for the training observations in the region to which it belongs.
- Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as *decision tree* methods.

Tree-Based Methods

- Tree-based methods are simple and useful for interpretation.
- There are also some multiple-tree approaches that produce multiple trees which are then combined to yield a single consensus prediction: *bagging*, *random forests*, *boosting*, and *Bayesian additive regression trees.*

Decision Tree

- *Decision tree* is a learning algorithm that breaks the input space into regions and has separate parameters for each region.
- Decision trees can be applied to both regression and classification problems.
  - Regression Trees
  - Classification Trees

Regression Trees

- In order to motivate *regression trees*, we begin with a simple example.
- We use the *Hitters* data set to predict a baseball player's *Salary* based on *Years* (the number of years that he has played in the major leagues) and *Hits* (the number of hits that he made in the previous year).
- We first remove observations that are missing *Salary* values, and log-transform *Salary* so that its distribution has more of a typical bell-shape. (Recall that *Salary* is measured in thousands of dollars.)

Regression Trees

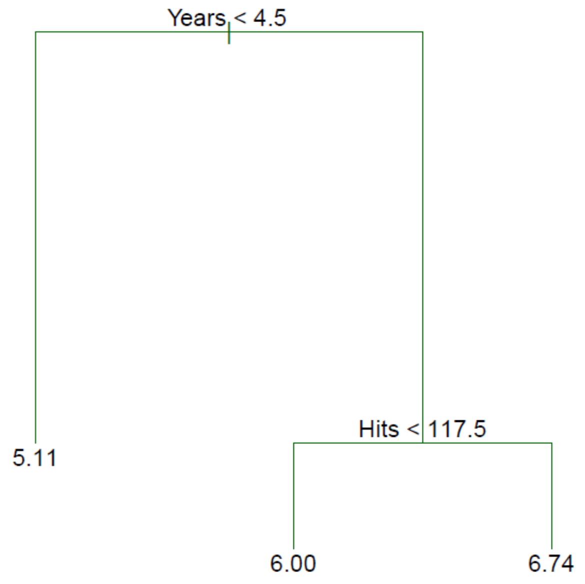


FIGURE 8.1. *For the* Hitters *data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year. At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$. For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to* Years<4.5*, and the right-hand branch corresponds to* Years>=4.5*. The tree has two internal nodes and three terminal nodes, or leaves. The number in each leaf is the mean of the response for the observations that fall there.*

Regression Trees

- Overall, the tree stratifies or segments the players into three regions of predictor space: players who have played for 4.5 or fewer years, players who have played for 4.5 or more years and who made fewer than 117.5 hits last year, and players who have played for 4.5 or more years and who made at least 118 hits last year.
- These three regions can be written as $R_1 = \{X \mid Years < 4.5\}, R_2 = \{X|Years >= 4.5, Hits < 117.5\}$, and $R_3 = \{X|Years >= 4.5, Hits >= 117.5\}$.

Regression Trees

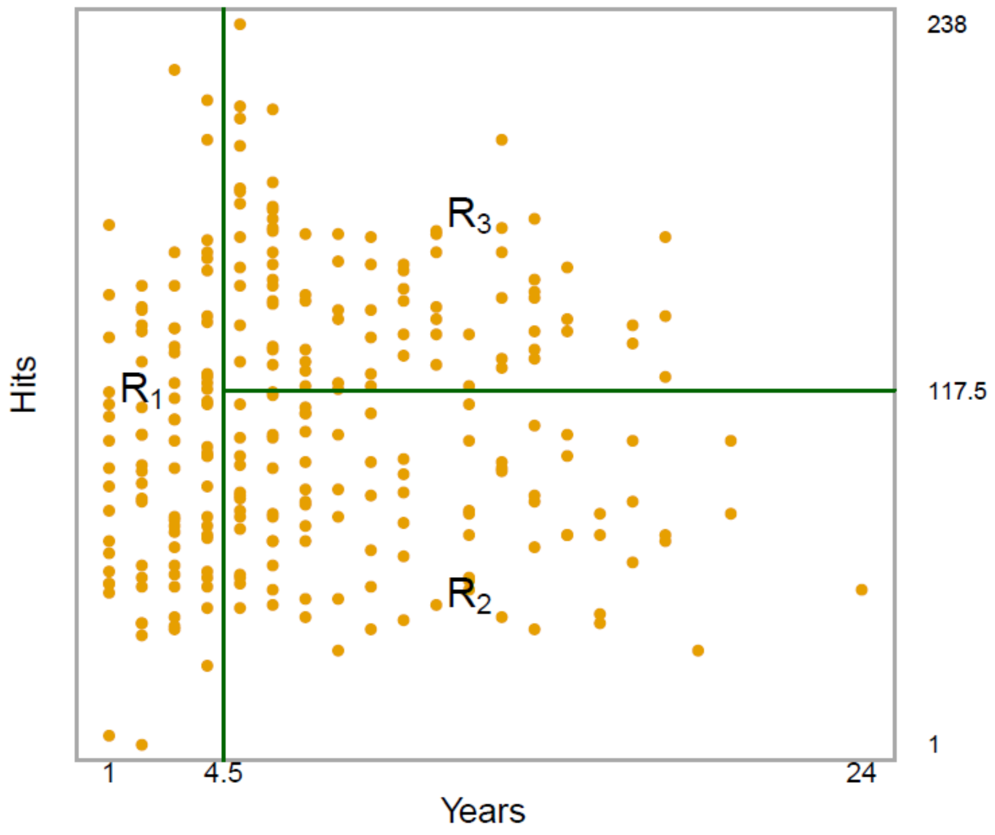


FIGURE 8.2. *The three-region partition for the* Hitters *data set from the regression tree illustrated in Figure 8.1.*

Regression Trees

- In keeping with the tree analogy, the regions R_1, R_2, and R_3 are known as *terminal nodes* or *leaves* of the tree.
- Decision trees are typically drawn *upside down*, in the sense that the leaves are at the bottom of the tree. The points along the tree where the predictor space is split are referred to as *internal nodes*.
- We refer to the node segments of the trees that connect the nodes as *branches*.

Regression Trees

- We might interpret the regression tree displayed in Figure 8.1 as follows:
  - *Years* is the most important factor in determining *Salary*, and players with less experience earn lower salaries than more experienced players.
  - Given that a player is less experienced, the number of *hits* that he made in the previous year seems to play little role in his salary.
  - But among players who have been in the major leagues for five or more years, the number of hits made in the previous year does affect salary, and players who made more hits last year tend to have higher salaries.

Prediction via Stratification of the Feature Space

- Building a regression tree has two steps:
  1. We divide the predictor space — that is, the set of possible values for $X_1, X_2, \ldots, X_p$— into J distinct and non-overlapping regions,$R_1, R_2, \ldots, R_J$.
  2. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

Prediction via Stratification of the Feature Space

- For instance, suppose that in Step 1 we obtain two regions, R_1 and R_2, and that the response mean of the training observations in the first region is 10, while the response mean of the training observations in the second region is 20.
- Then for a given observation $X = x$, if $x \in R_1$, we will predict a value of 10, and if if $x \in R_2$, we will predict a value of 20.

Prediction via Stratification of the Feature Space

- How do we construct the regions $R_1, R_2, \ldots, R_J$?
  - In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model.
  - The goal is to find boxes $R_1, R_2, \ldots, R_J$ that minimize the RSS, given by $\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$ where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

Prediction via Stratification of the Feature Space

- Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a *top-down, greedy* approach that is known as *recursive binary splitting*.
- The approach is *top-down* because it begins at the top of the tree (at which point all observations belong to a single region) and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is *greedy* because at each step of the tree-building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

Prediction via Stratification of the Feature Space

- In order to perform recursive binary splitting, we first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS. (The notation $\{X|X_j < s\}$ means *the region of predictor space in which X_j takes on a value less than s.*)
- That is, we consider all predictors $X_1, \ldots, X_p$, and all possible values of the cutpoint s for each of the predictors, and then choose the predictor and cutpoint such that the resulting tree has the lowest RSS.

Prediction via Stratification of the Feature Space

- Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions. However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions.
- We now have three regions. Again, we look to split one of these three regions further, so as to minimize the RSS. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

Prediction via Stratification of the Feature Space

- Once the regions $R_1, R_2, \ldots, R_J$ have been created, we predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.

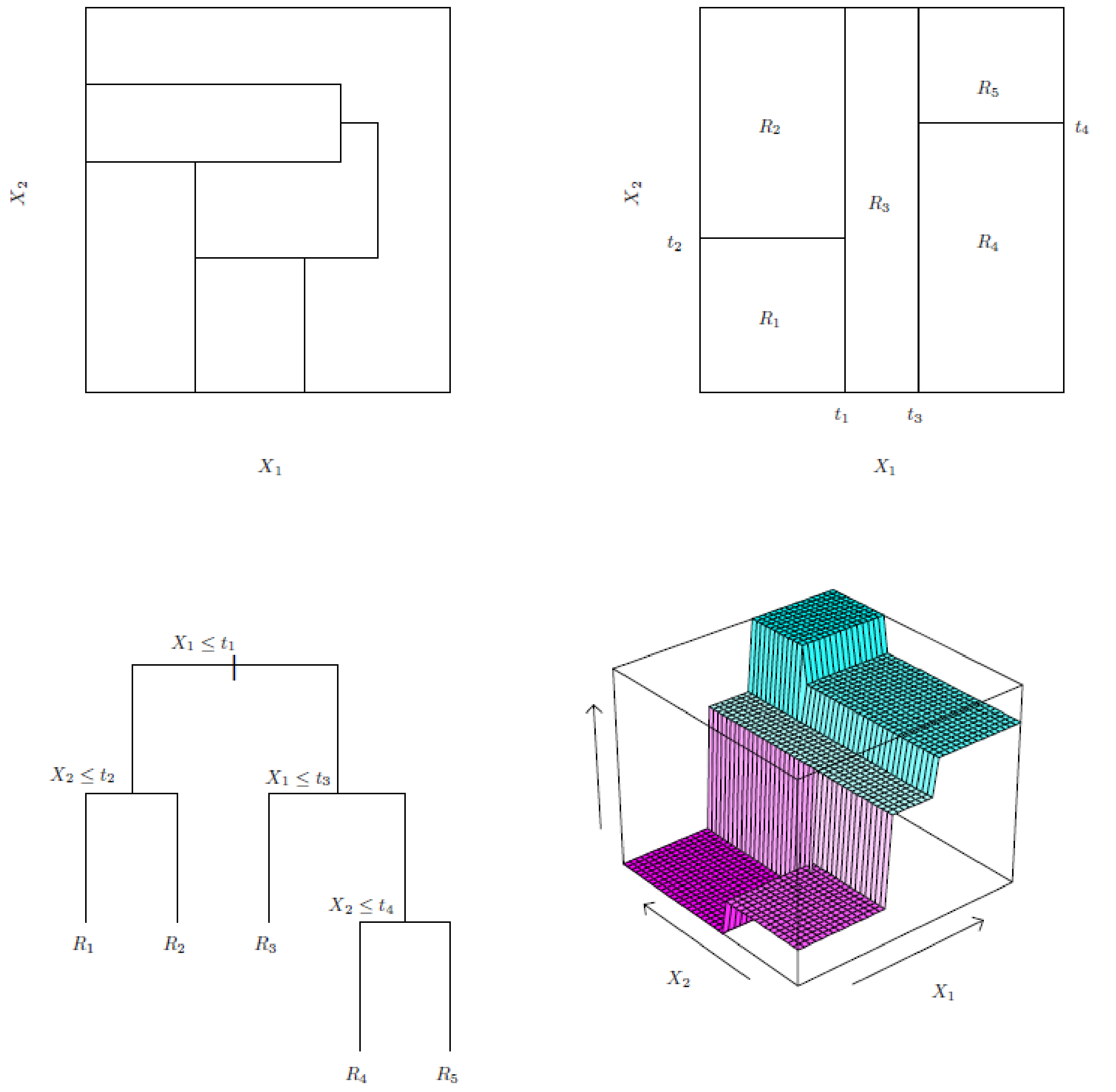


FIGURE 8.3. Top Left: *A partition of two-dimensional feature space that could not result from recursive binary splitting.* Top Right: *The output of recursive binary splitting on a two-dimensional example.* Bottom Left: *A tree corresponding to the partition in the top right panel.* Bottom Right: *A perspective plot of the prediction surface corresponding to that tree.*

Classification Trees

- A *classification tree* is very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one.
- For a classification tree, we predict that each observation belongs to the *most commonly occurring class* of training observations in the region to which it belongs.

Classification Trees

- The task of growing a classification tree is quite similar to the task of growing a regression tree. Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- However, in the classification setting, RSS cannot be used as a criterion for making the binary splits. A natural alternative to RSS is the *classification error rate*.

Classification Trees

- The classification error rate is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \max_{k}(\hat{p}_{mk})$$

- Here $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class. However, it turns out that classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.

Classification Trees

- The *Gini index* is defined by

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

a measure of total variance across the K classes.

- It is not hard to see that the Gini index takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one. For this reason the Gini index is referred to as a measure of node purity—a small value indicates that a node contains predominantly observations from a single class.

Classification Trees

- An alternative to the Gini index is *entropy*, given by

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} log \hat{p}_{mk}$$

- Since $0 \leq \hat{p}_{mk} \leq 1$, it follows $0 \leq -\hat{p}_{mk} log \hat{p}_{mk}$.
- The entropy will take on a value near zero if the $\hat{p}_{mk}$'s are all near zero or near one. Therefore, like the Gini index, the entropy will take on a small value if the mth node is pure. In fact, it turns out that the Gini index and the entropy are quite similar numerically.

Trees Versus Linear Models

- If the relationship between the features and the response is well approximated by a linear model, then an approach such as linear regression will likely work well, and will outperform a method such as a regression tree that does not exploit this linear structure.
- If instead there is a highly nonlinear and complex relationship between the features and the response, then decision trees may outperform classical approaches.

Trees Versus Linear Models

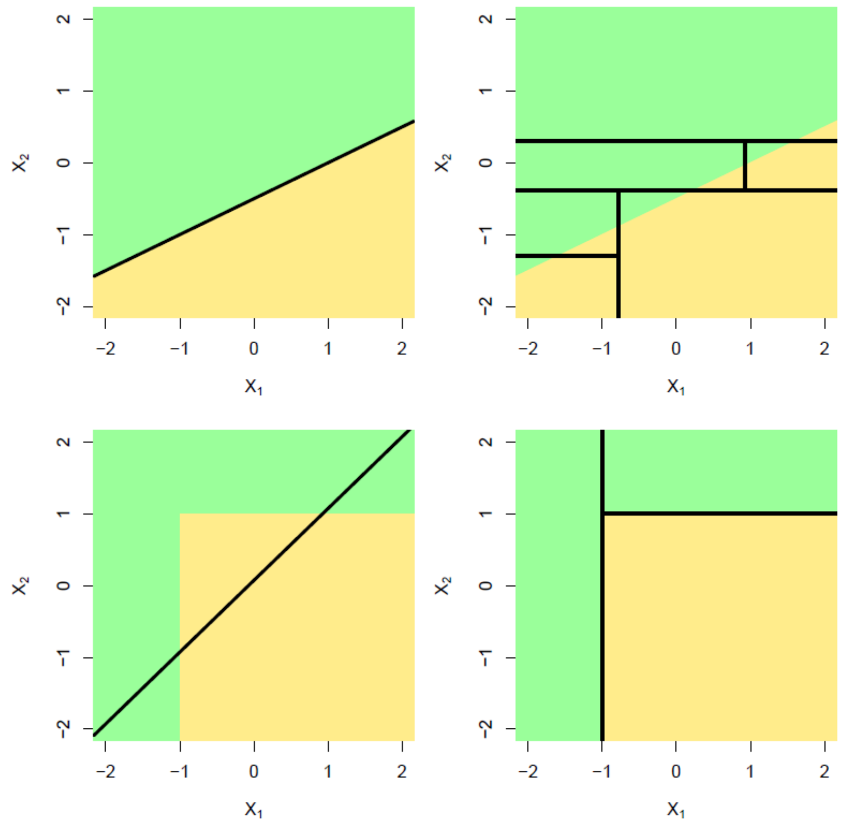


FIGURE 8.7. Top Row: *A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a decision tree that performs splits parallel to the axes (right).* Bottom Row: *Here the true decision boundary is non-linear. Here a linear model is unable to capture the true decision boundary (left), whereas a decision tree is successful (right).*

Advantages and Disadvantages of Trees

- Decision trees for regression and classification have a number of advantages:
  - Trees are very easy to explain to people.
  - Some people believe that decision trees more closely mirror human decision-making.
  - Trees can be displayed graphically, and are easily interpreted even by a non-expert.
  - Trees can easily handle qualitative predictors without the need to create dummy variables.

Advantages and Disadvantages of Trees

- Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches.
- Additionally, trees can be very non-robust. In other words, a small change in the data can cause a large change in the final estimated tree.
- However, by aggregating many decision trees, using methods like bagging, random forests, and boosting, the predictive performance of trees can be substantially improved.

Ensemble Method

- An *ensemble* method is an approach that combines many simple "building block" models in order to obtain a single and potentially very powerful ensemble model.
- These simple building block models are sometimes known as *weak learners*, since they may lead to mediocre predictions on their own.
- Bagging, random forests, boosting, and Bayesian additive regression trees are ensemble methods for which the simple building block is a regression or a classification tree.

Bagging

- The decision 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 that we get could be quite different.
- *Bootstrap aggregation*, or *bagging*, is a general-purpose procedure for reducing the variance of a statistical learning method.

Bagging

- To apply bagging to regression trees, we simply construct B regression trees using B bootstrapped training sets, and average the resulting predictions. These trees are grown deep, and are not pruned.
- Hence each individual tree has high variance, but low bias. Averaging 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.

Bagging

- How can bagging be extended to a classification problem where Y is qualitative?
  - For a given test observation, we can record the class predicted by each of the B trees, and take a majority vote: the overall prediction is the most commonly occurring class among the B predictions.

Random Forests

- *Random forests* provide an improvement over bagged trees by way of a small tweak that *decorrelates* the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors.

Random Forests

- A fresh sample of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$.
- In other words, in building a random forest, at each split in the tree, the algorithm is not even allowed to consider a majority of the available predictors.
- This may sound crazy, but it has a clever rationale. Suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors. Then in the collection of bagged trees, most or all of the trees will use this strong predictor in the top split. Hence the predictions from the bagged trees will be highly correlated.

Random Forests

- Unfortunately, averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities.
- In particular, this means that bagging will not lead to a substantial reduction in variance over a single tree in this setting.
- Random forests overcome this problem by forcing each split to consider only a subset of the predictors. Therefore, on average $(p - m)/p$ of the splits will not even consider the strong predictor, and so other predictors will have more of a chance. We can think of this process as *decorrelating* the trees, thereby making the average of the resulting trees less variable and hence more reliable.

Random Forests

- The main difference between bagging and random forests is the choice of predictor subset size m. For instance, if a random forest is built using $m = p$, then this amounts simply to bagging.

Boosting

- Like bagging, *boosting* is a general approach that can be applied to many statistical learning methods for regression or classification. Here we restrict our discussion of boosting to the context of decision trees.
- Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model.

Boosting

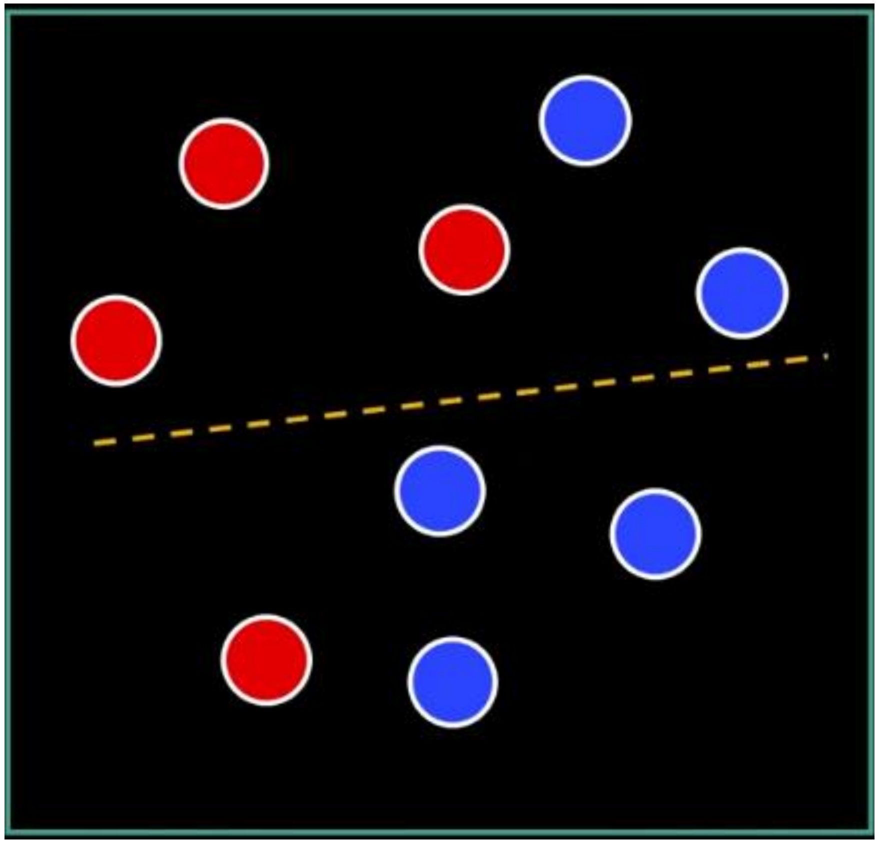
- Notably, each tree is built on a bootstrap data set, independent of the other trees.
- Boosting works in a similar way, except that the trees are grown sequentially: each tree is grown using information from previously grown trees. Boosting does not involve bootstrap sampling; instead each tree is fit on a modified version of the original data set.

Boosting

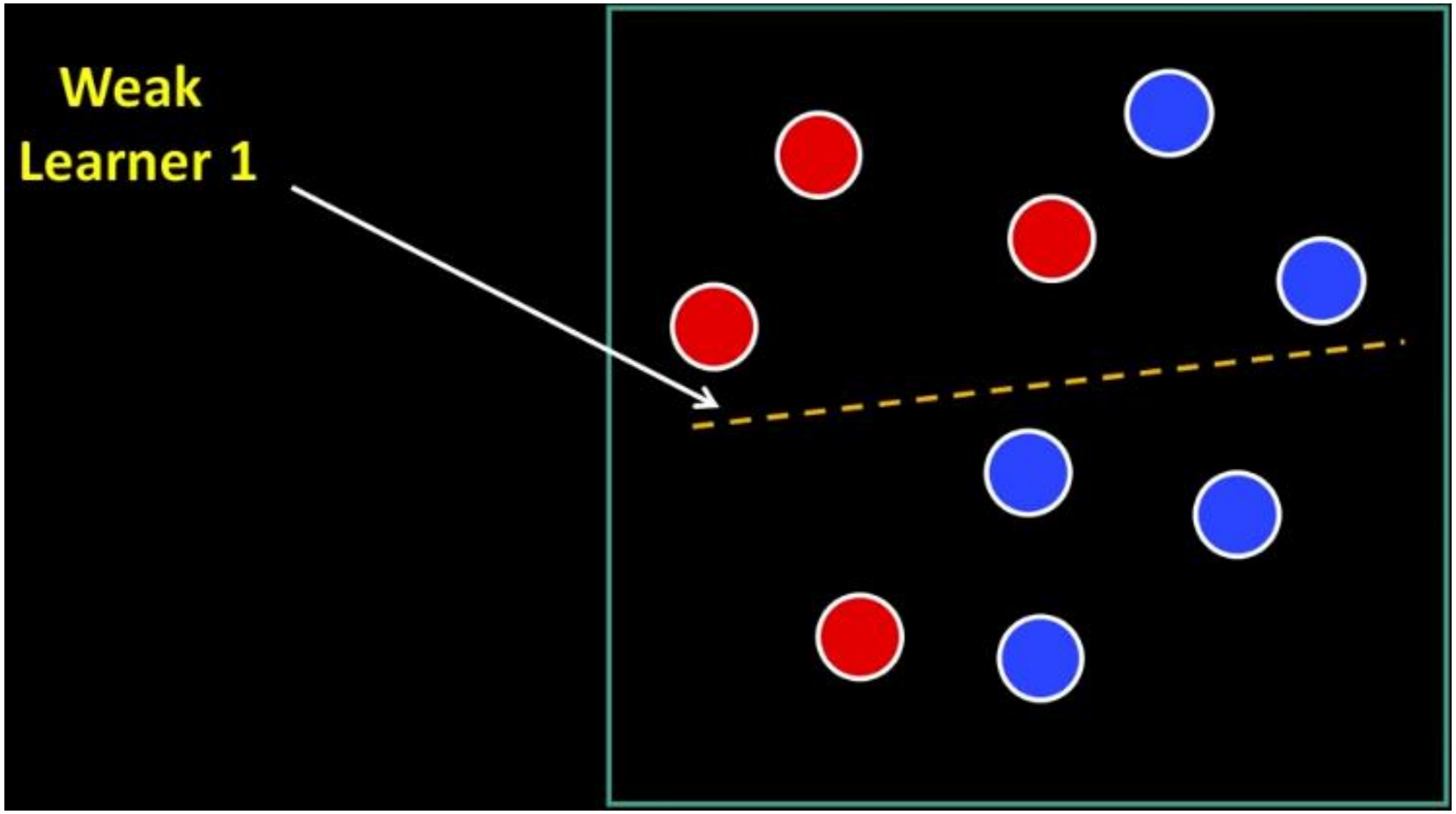
Training method:

- Initially, weight each training example equally
- In each boosting round:
  - Find the weak learner that achieves the lowest weighted training error
  - Raise weights of training examples misclassified by current weak learner

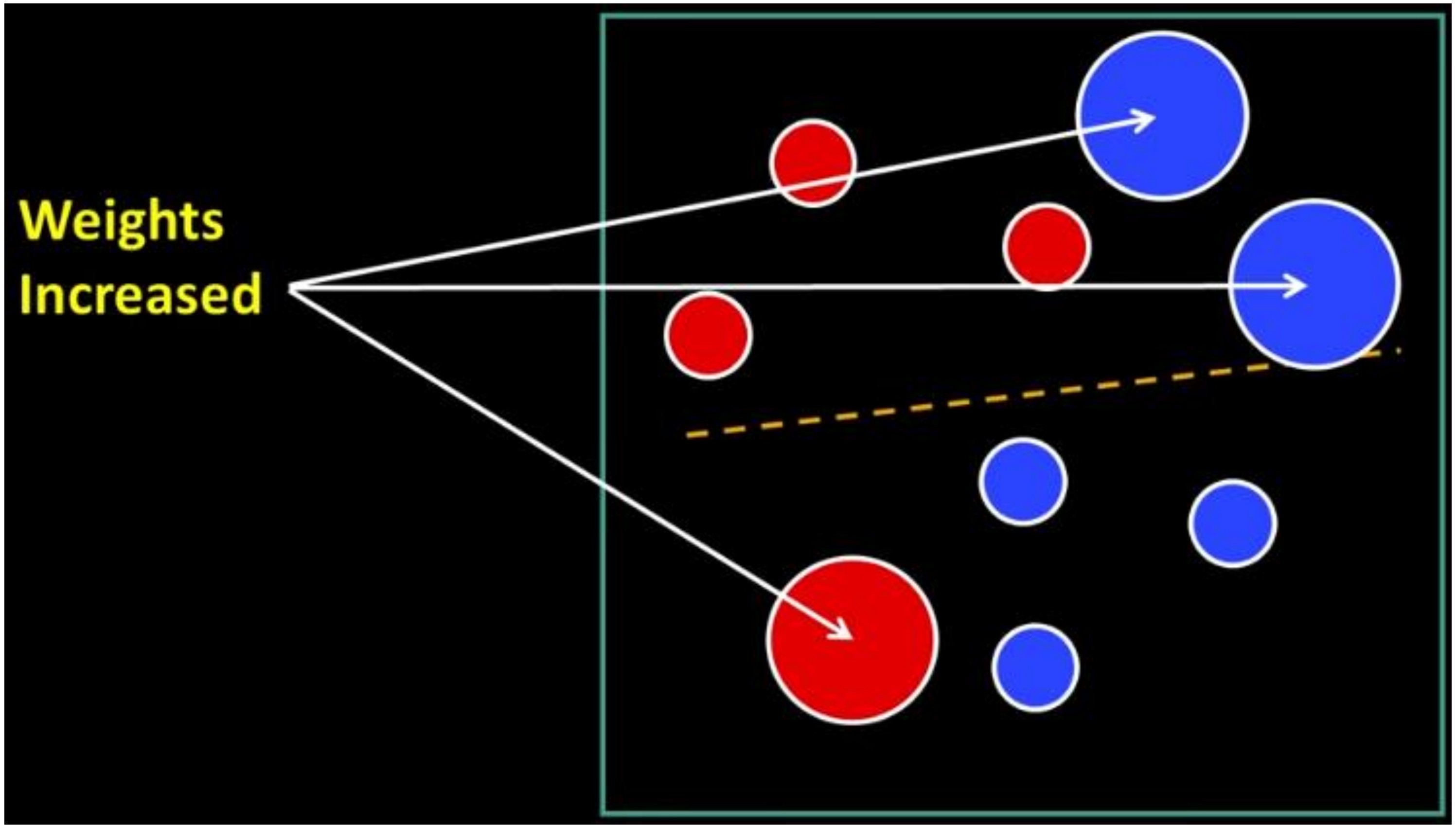
Boosting



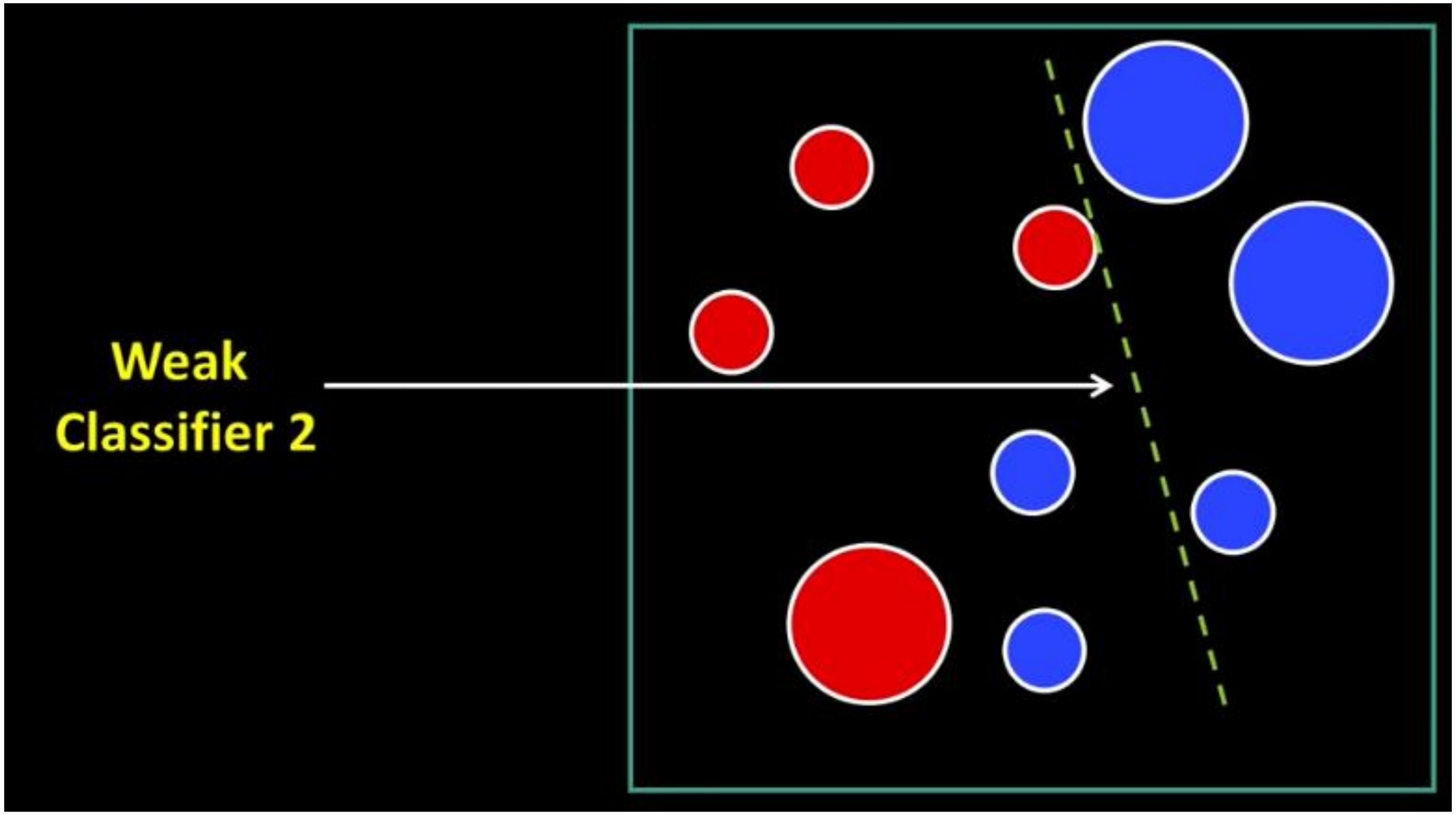
Boosting



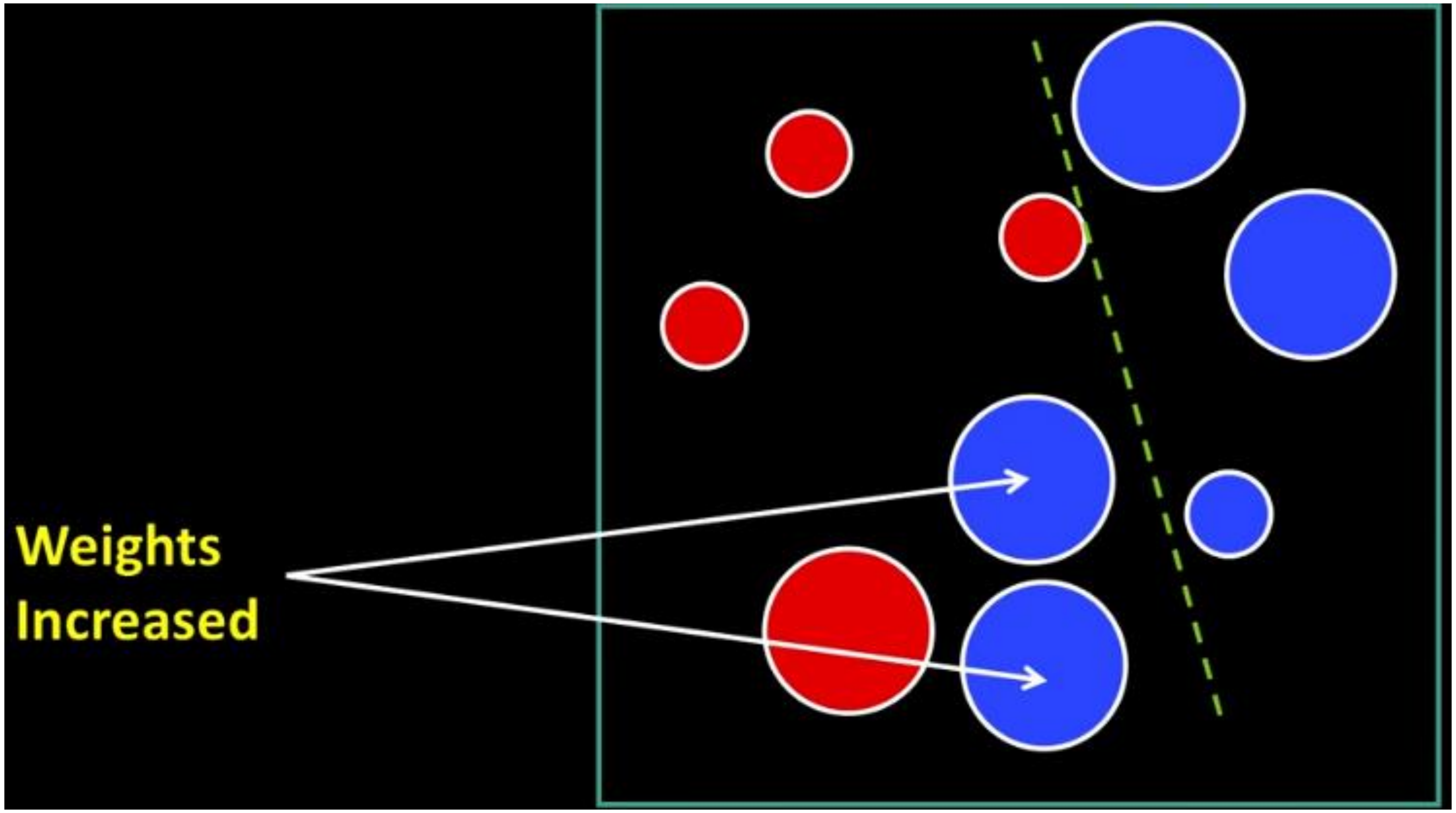
Boosting



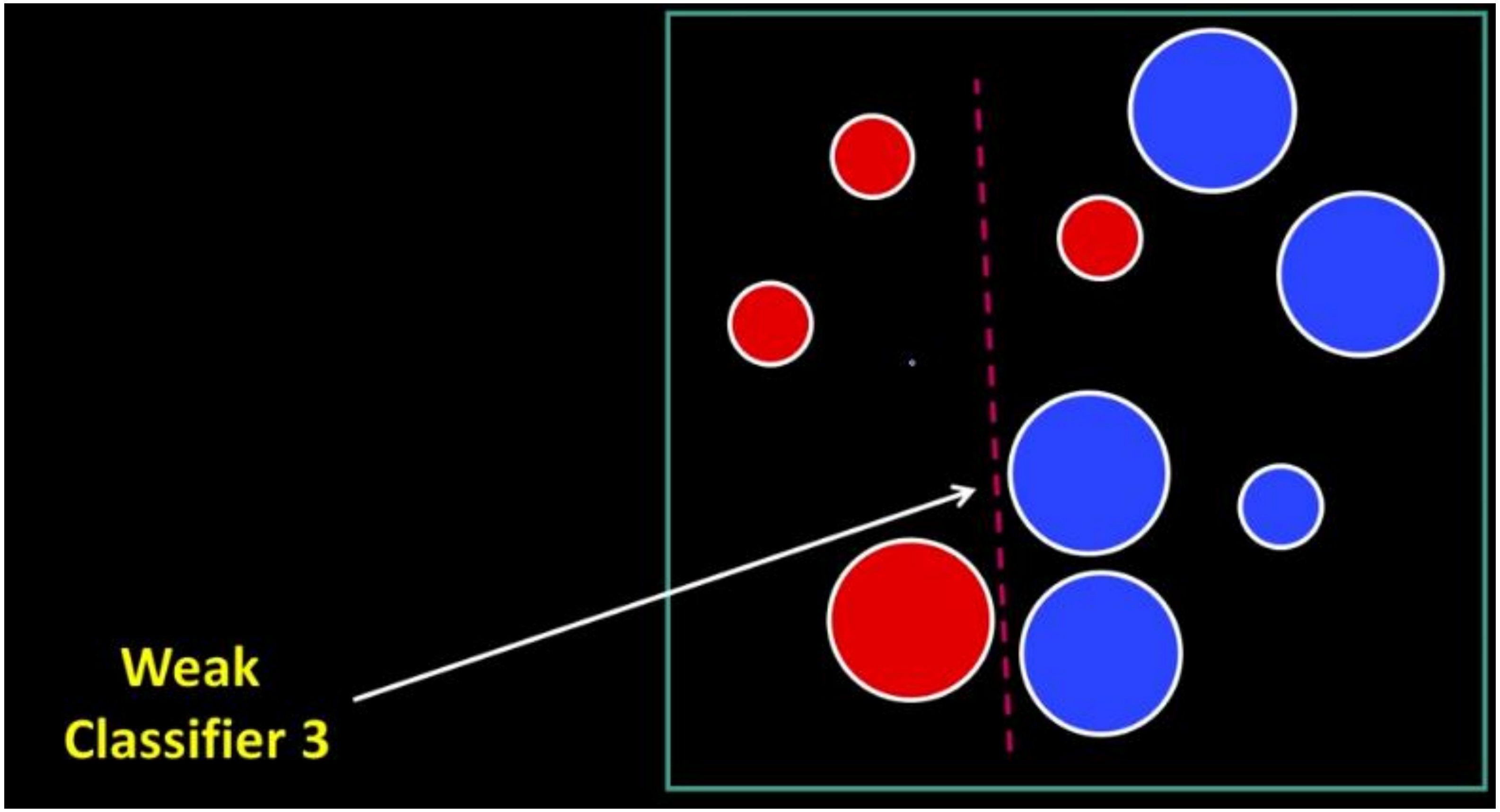
Boosting



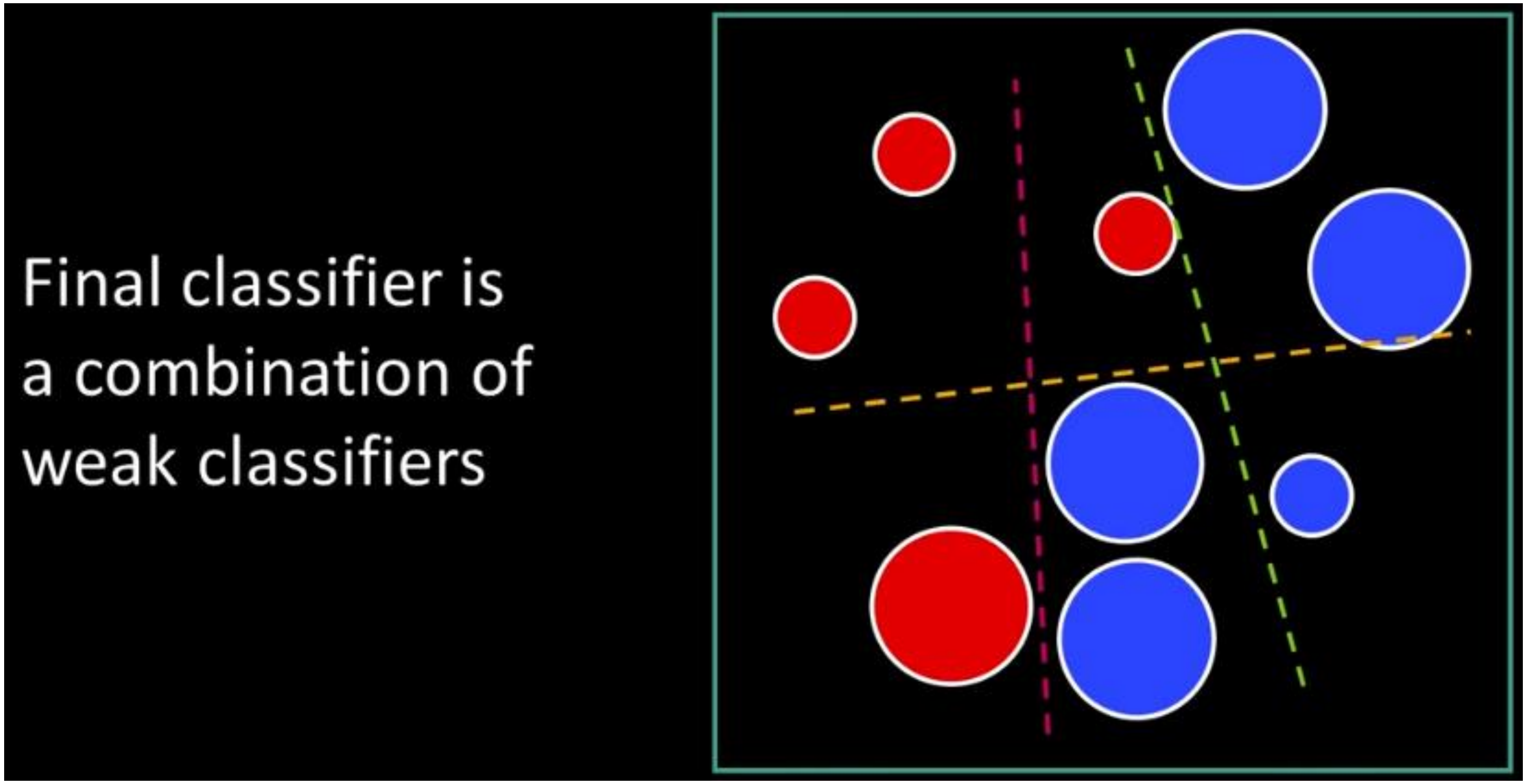
Boosting



Boosting



Boosting



Final classifier is a combination of weak classifiers

Boosting

- General: compute final classifier as linear combination of all weak learners (weight of each learner is directly proportional to its accuracy)
- Exact formulas for re-weighting and combining weak learners depend on the particular boosting scheme (e.g., Adaboost)