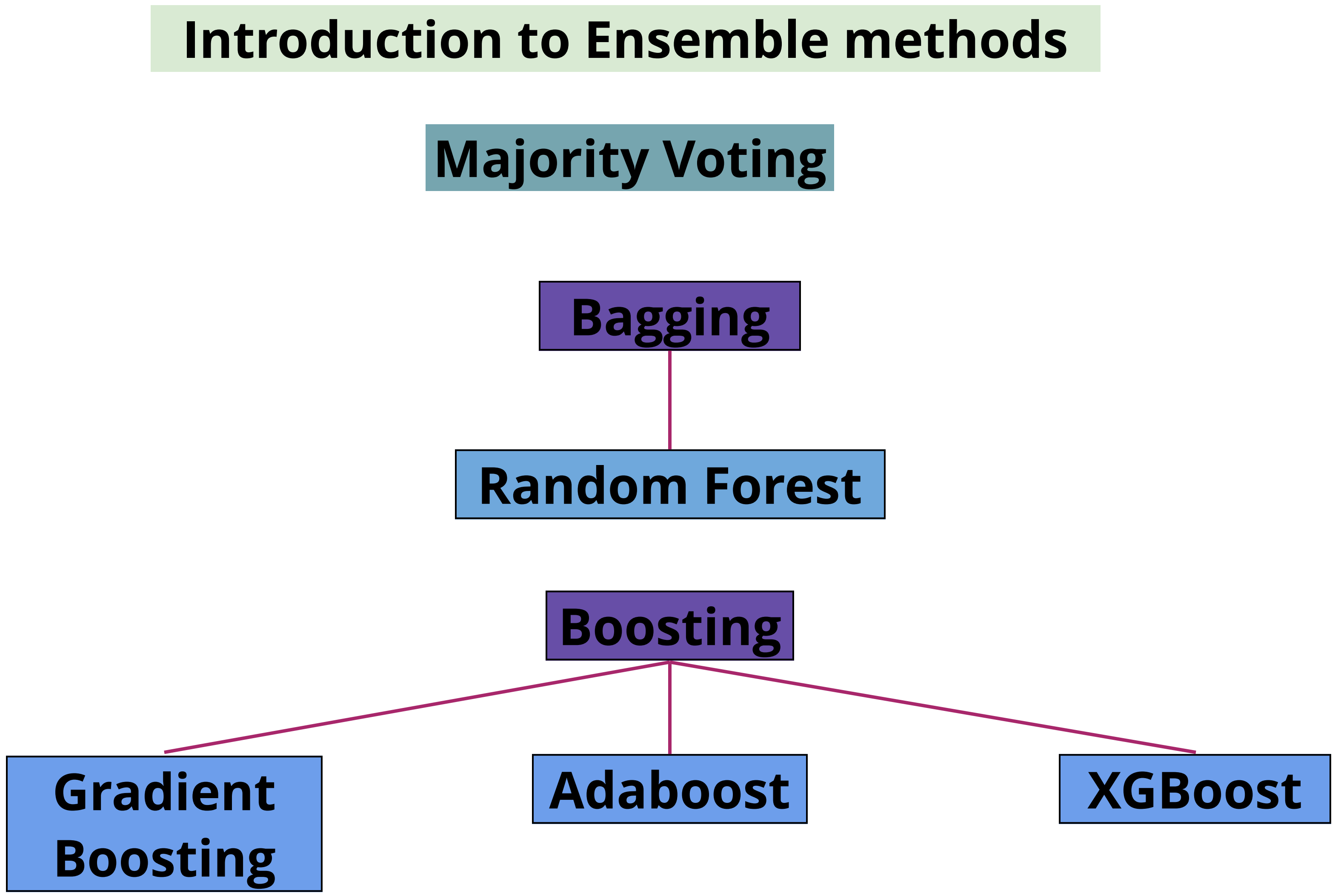
Ensemble Learning

Machine Learning Techniques

Dr. Ashish Tendulkar
IIT Madras

What will be covered in this module?

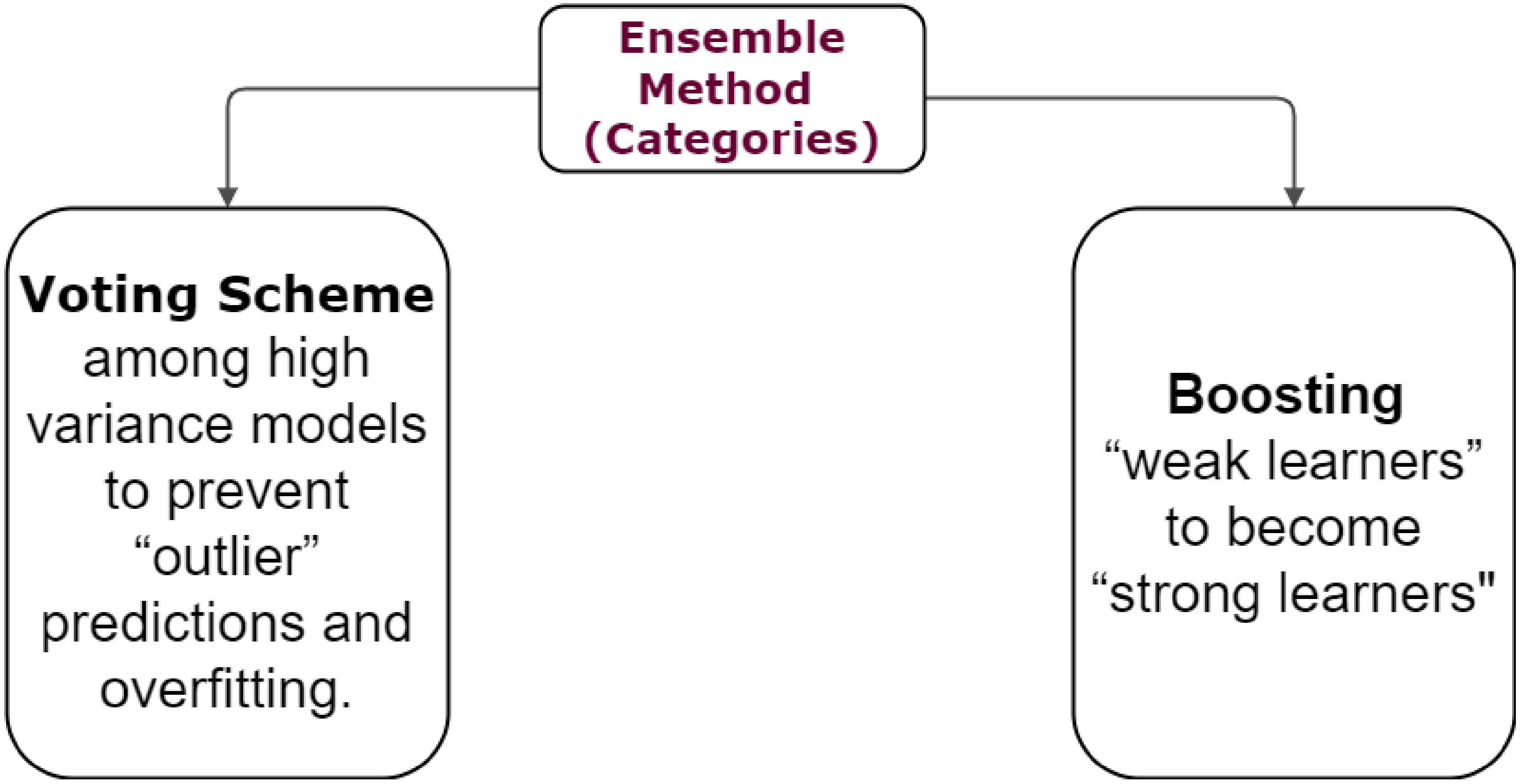


Introduction to Ensemble Methods

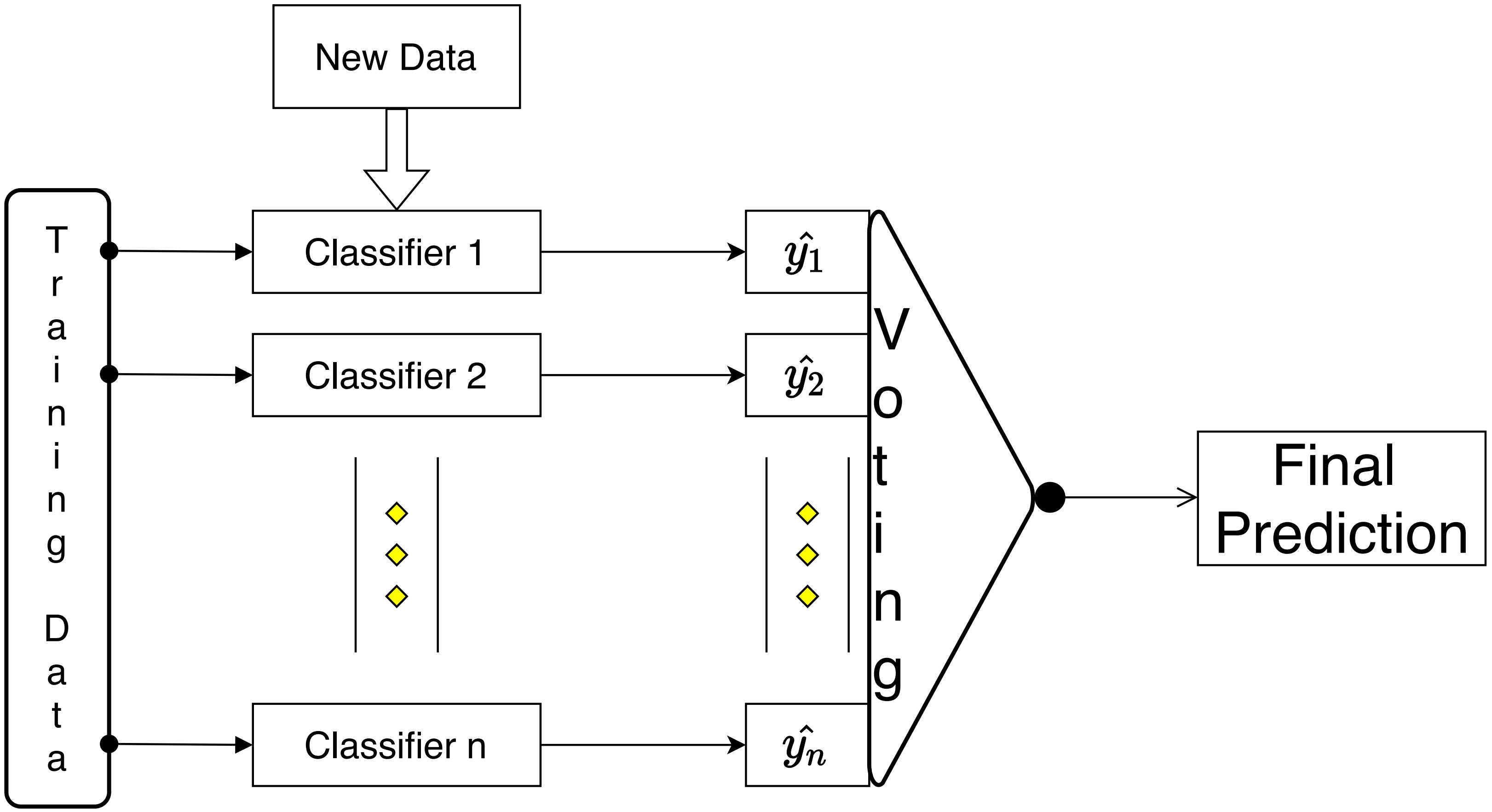
- Ensemble Learning is a machine Learning approach in which we combine predictions of different models to get better performance.

- In broad terms, using ensemble methods is about combining models to an ensemble such that the ensemble has a better performance than an individual model on average.

There are two main categories of Ensemble Learning:


Ensemble Method (Categories)
Voting Scheme among high variance models to prevent “outlier” predictions and overfitting.
Boosting “weak learners” to become “strong learners"

Majority Vote



Why Majority Voting can be useful?

- Let there be q classifiers with error rates ϵ_i, where $1 \leq i \leq q$.
- Assuming the error rate is better than random guessing (i.e., lower than 0.5 for binary classification):

$$\forall \epsilon_i \in \{\epsilon_1, \epsilon_2, ..., \epsilon_q\}, \epsilon_i < 0.5$$

- The ensemble makes a wrong prediction if more than 50% of the q classifiers make a wrong prediction.
- The probability that we make a wrong prediction via the ensemble if r classifiers predict the same class label (where $r > \frac{q}{2}$) is then $\binom{q}{r}\epsilon^r(1-\epsilon)^{q-r}$.

Why Majority Voting can be useful?

- Consider an example with $q = 11$ with error rate $\epsilon = 0.3$.

  The error of ensemble can be calculated as:

$$\binom{11}{6}0.3^6(1-0.3)^5 = 6.696 \times 10^{-6}$$

Hard Voting

1. Get class label prediction from each classifier. Let's consider this prediction as a vote of the classifier.
2. The class label that receives the highest number of votes is chosen as final prediction.

Soft Voting

1. Take average of probability vectors produced by different classifiers.
2. The class with the highest probability is assigned to the example.

In case of regression, we take average of output predicted by different regression models in the ensemble.

Bagging

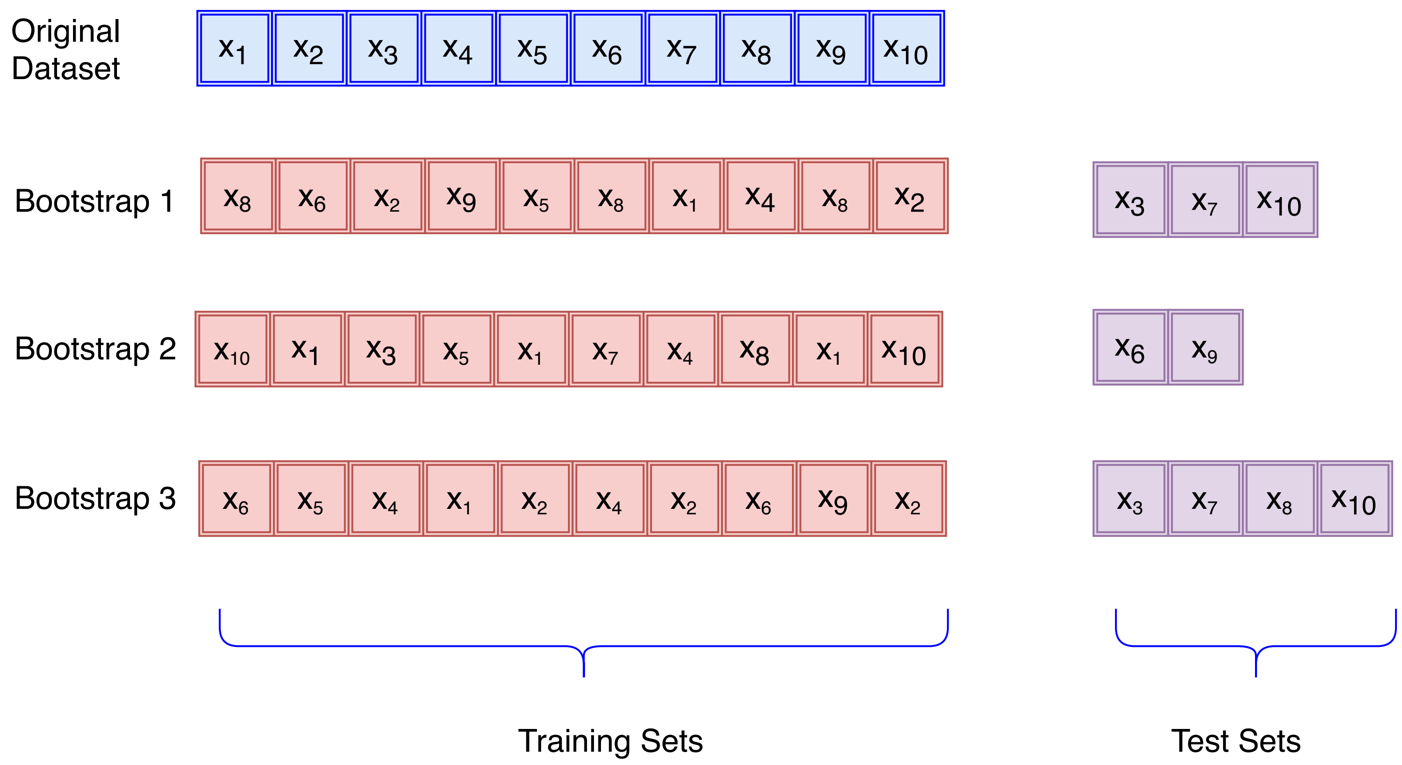
Like voting with same classifier trained on different datasets

Where is it used?

- The major idea of the bootstrap aggregation or bagging method is to combine prediction functions learned from multiple data sets, with a view to improving overall prediction accuracy.

- Bagging is especially beneficial when dealing with predictors that tend to overfit the data, such as in decision trees, where the (unpruned) tree structure is very sensitive to small changes in the training set.

Bootstrapping



Original Dataset
x1 x2 x3 x4 x5 x6 x7 x8 x9 x10
Bootstrap 1
x8 x6 x2 x9 x5 x8 x1 x4 x8 x2
x3 x7 x10
Bootstrap 2
x10 x1 x3 x5 x1 x7 x4 x8 x1 x10
x6 x9
Bootstrap 3
x6 x5 x4 x1 x2 x4 x2 x6 x9 x2
x3 x7 x8 x10
Training Sets
Test Sets

- In each of the bootstraps some original data points are removed and replaced by existing data points.
- Updated dataset is rearranged.

Bagging Algorithm

- Let n be the number of bootstrap samples

- for $i = 1$ to q do
  - Draw boostrap sample of size n: $D^{(i)}$.
  - Train a base classifier h_i on $D^{(i)}$.

- For a new data point $\mathbf{x}$, the prediction: $\hat{y} = \frac{1}{q}\sum_{j=1}^{q} h_j(\mathbf{x})$

Random Forest

Overview

One of the most widely used ML algorithm due to

- Ease of use
- not much tuning required to get good results

The random forest algorithm can be thought of as bagging with decision trees.

Random Forest Algorithm

- Sample q data sets $D_1, D_2, \dots, D_q$ from D with replacement.
- For each D_i, train a full decision tree with one small modification - randomly sample u of m features (without replacement) and only consider these features for split.
- The final classifier is: $h(\mathbf{x}) = \frac{1}{q}\sum_{j=1}^{q} h_j(\mathbf{x})$

Two hyperparameters:

- # classifiers: q
- # features: u

Random forest is insensitive to its hyper-parameters.

- Set the number of classifiers q as large as we can afford.
- Set the number of features u to either $\sqrt{m}$ or $\log m + 1$, where m is the total number of features.

Boosting

Train different classifiers on the same training data.

Overview

The principle behind boosting algorithms is that first we build a model on the training dataset, then a second model is built to rectify the errors present in the first model.

This procedure is continued until and unless the errors are minimized, and the dataset is predicted correctly.

In particular, we start with a weak model and subsequently, each new model is fit on a modified version of the original data set.

Note

Weak Learner:

A weak learner is a model that performs at least slightly better than a random model.

- Boosting is an iterative process, where the training set is reweighted, at each iteration, based on mistakes of a weak leaner (i.e., misclassifications).
- The two approaches, adaptive and gradient boosting, differ mainly regarding how the weights are updated and how the classifiers are combined.

Gradient Boosting

- Make a first guess for y_{train} and y_{test} using the average of y_{train}.

$$y_{\text{train}_{q0}} = \frac{1}{n}\sum_{i=1}^{n} y_{\text{train}_i}$$

$$y_{\text{test}_{q0}} = y_{\text{train}_{q0}}$$

- Calculate the residuals from training data sets using the following:

$$r_0 = y_{\text{train}} - y_{\text{train}_{q0}}$$

Gradient Boosting

- Fit a weak learner to residuals minimizing the loss function. Let's call it f_0:

$$r_0 = f_0(X_{\text{train}_0})$$

- Increment the predicted y's

$$\begin{aligned} y_{\text{train}_{p1}} &= y_{\text{train}_{p0}} + \alpha f_0(X_{\text{train}}) \\ y_{\text{test}_{p1}} &= y_{\text{test}_{p0}} + \alpha f_0(X_{\text{test}}) \end{aligned}$$

- Repeat steps 2 to 4 until you reach the boosting rounds

Adaptive Boosting (AdaBoost)

- Ensemble techniques attempts to create strong classifiers from a number of weak classifiers.

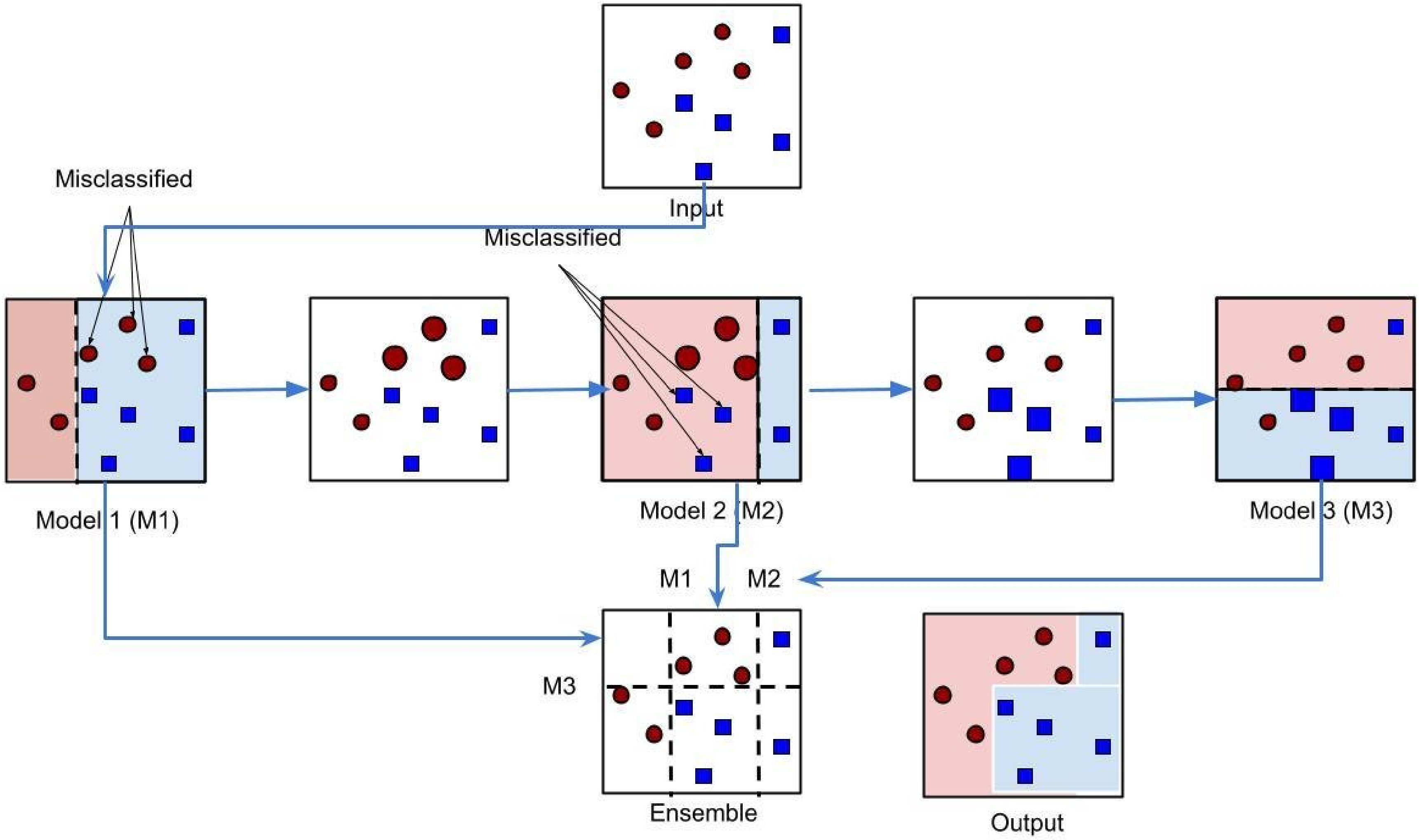
- AdaBoost uses a decision tree with one depth i.e. with only one split.   Such trees are called decision stumps.
    - Stumps have one node and two leaves and are not expected to produce a good accuracy hence are weak classifiers.

- Multiple decision stumps are combined to make a strong classifier.

- It builds an initial model while giving equal weights to all the data points.
- It then assigns higher weights to points that were mis-classified.
- Now all the points which have higher weights are given more importance in the next model.
- In other words, each model compensates the weaknesses of its predecessors.
- In this way, it will keep training models.
- The final model uses the weighted average of individual models.

- It works by putting more weight on difficult to classify instances and less on those already handled well.
- AdaBoost algorithm can solve both classification and regression problems.

AdaBoost schematic



- In the image in previous slide, we see that the data points misclassified by Model 1 (M1) are given more weight before the data is fed to model 2 (M2).
- Again, the data points misclassified by Model 2 are given more weight before they are fed to model 3 (M3).
- Giving higher weights to misclassified points means that the subsequent model is going to focus more on these data points while deciding the decision boundary.
- We can see that all these individual models, M1, M2 and M3 are not strong enough to correctly classify the data points.

Stump Performance

The performance α is calculated as follows:

$$\alpha = \frac{1}{2} \ln \frac{1 - \text{Total Error}}{\text{Total Error}}$$

- Total error is sum of weights of misclassified samples. It is always between 0 and 1.

- Higher the value of α, better is the performance of stump.

- α is used to updates weights for the next model.

Weight Updating

- Weights are updated based on the performance of the stump.

- For misclassified samples

$$\text{weight}^{(\text{new})} = \text{weight}^{(\text{old})} \times e^{\alpha}$$

- For correctly classified samples

$$\text{weight}^{(\text{new})} = \text{weight}^{(\text{old})} \times e^{-\alpha}$$