Ensemble Techniques

- Bagging
- Boosting
- Stacking

Ensemble Methods

Bagging : that often considers homogeneous weak learners, learns them independently from each other in parallel and combines them following some kind of deterministic averaging process

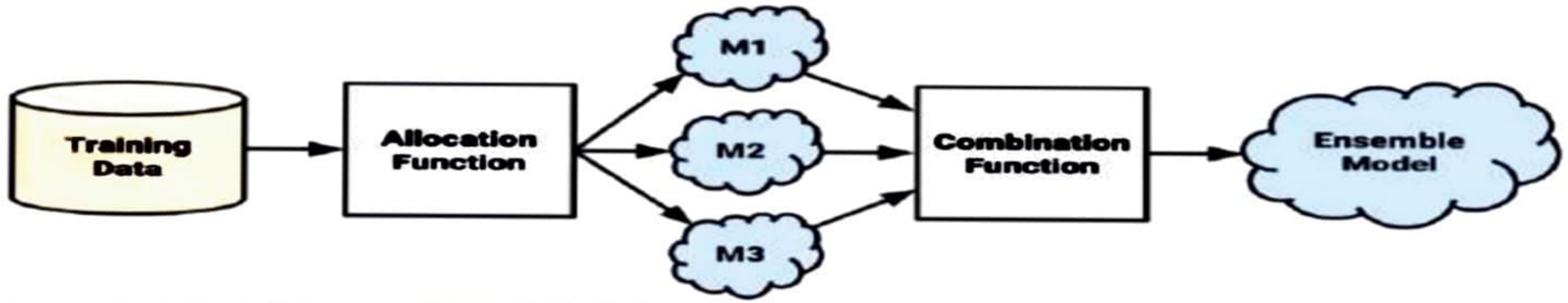
Boosting : that often considers homogeneous weak learners, learns them sequentially in a very adaptative way (a base model depends on the previous ones) and combines them following a deterministic strategy

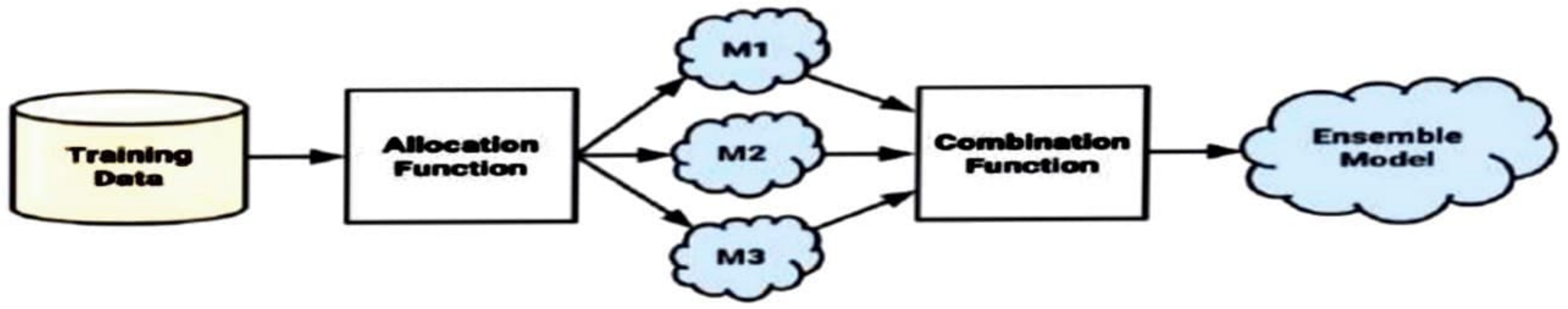
Stacking : that often considers heterogeneous weak learners, learns them in parallel and combines them by training a meta-model to output a prediction based on the different weak models predictions

Understanding ensembles

Suppose you were a contestant on a television trivia show that allowed you to choose a panel of five friends to assist you with answering the final question for the million-dollar prize. Most people would try to stack the panel with a diverse set of subject matter experts. A panel containing professors of literature, science, history, and art, along with a current pop-culture expert would be a safely well-rounded group. Given their breadth of knowledge, it would be unlikely to find a question that stumps the group.

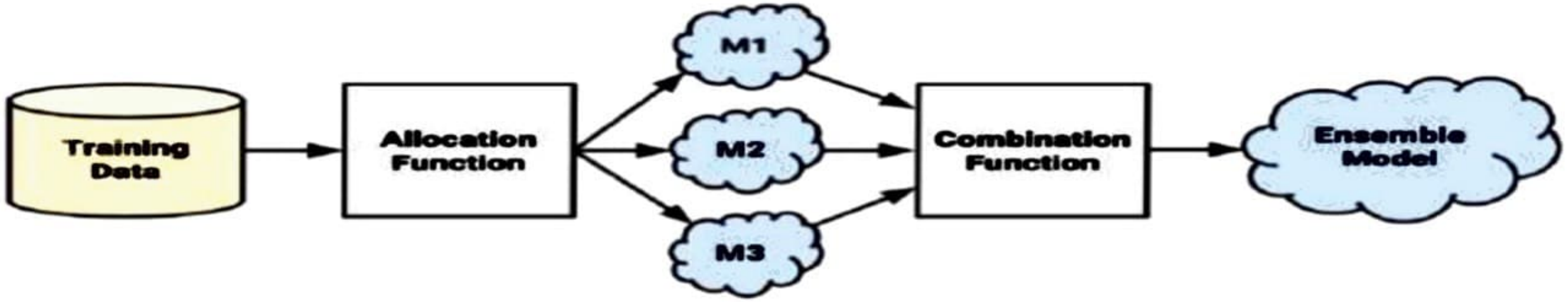
The meta-learning approach that utilizes a similar principle of creating a varied team of experts is known as an **ensemble**. All the ensemble methods are based on the idea that by combining multiple weaker learners, a stronger learner is created.





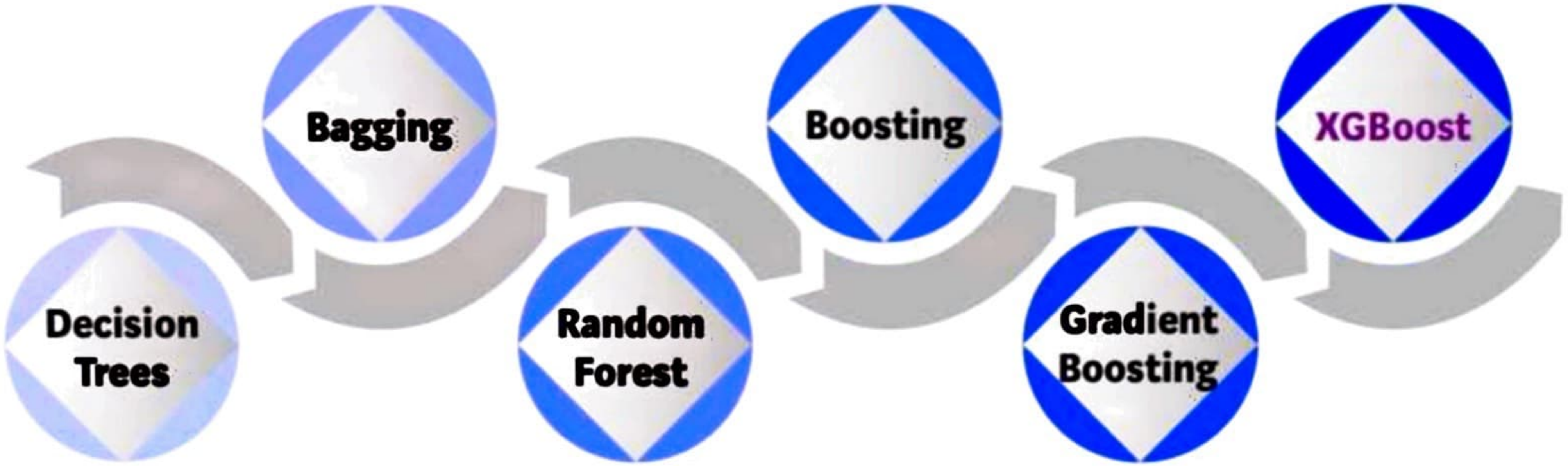
- First, input training data is used to build a number of models.
- The allocation function dictates how much of the training data each model receives. Do they each receive the full training dataset or merely a sample? Do they each receive every feature or a subset?
- After the models are constructed, they can be used to generate a set of predictions, which must be managed in some way.
- The combination function governs how disagreements among the predictions are reconciled

  For example, the ensemble might use a majority vote to determine the final prediction, or it could use a more complex strategy such as weighting each model's votes based on its prior performance



- First, input training data is used to build a number of models.
- The allocation function dictates how much of the training data each model receives. Do they each receive the full training dataset or merely a sample? Do they each receive every feature or a subset?
- After the models are constructed, they can be used to generate a set of predictions, which must be managed in some way.
- The combination function governs how disagreements among the predictions are reconciled

  For example, the ensemble might use a majority vote to determine the final prediction, or it could use a more complex strategy such as weighting each model's votes based on its prior performance



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 Boosti algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias

A graphical epresentation of ssible solutions to decision based on ertain 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

Bagging

One of the first ensemble methods to gain widespread acceptance used a technique called **bootstrap aggregating** or **bagging** for short.

Bagging generates a number of training datasets by bootstrap sampling the original training data. These datasets are then used to generate a set of models using **a single learning algorithm**. The models' predictions are combined using voting (for classification) or averaging (for numeric prediction).

Although bagging is a relatively simple ensemble, it can perform quite well as long as it is used with relatively **unstable** learners, that is, those generating models that tend to change substantially when the input data changes only slightly. Unstable models are essential in order to ensure the ensemble's diversity in spite of only minor variations between the bootstrap training datasets. For this reason, bagging is often used with decision trees, which have the tendency to vary dramatically given minor changes in the input data.

Bootstrap Method

The bootstrap is a powerful statistical method for estimating a quantity from a data sample. This is easiest to understand if the quantity is a descriptive statistic such as a mean or a standard deviation

Let's assume we have a sample of 100 values (x) and we'd like to get an estimate of the mean of the sample. We can calculate the mean directly from the sample as:

mean(x) = 1/100 * sum(x)

We know that our sample is small and that our mean has error in it. We can improve the estimate of our mean using the bootstrap procedure:

- ✓ Create many (e.g. 1000) random sub-samples of our dataset with replacement (meaning we can select the same value multiple times).
- ✓ Calculate the mean of each sub-sample.
- ✓ Calculate the average of all of our collected means and use that as our estimated mean for the data.

Let's assume we have a sample dataset of 1000 instances (x) and we are using the C5.0 algorithm. Bagging of the C5.0 algorithm would work as follows.

1.Create many (e.g. 100) random sub-samples of our dataset with replacement.
2.Train a C5.0 model on each sample.
3.Given a new dataset, calculate the average prediction from each model.

For example,

if we had 5 bagged decision trees that made the following class predictions for a in input sample: blue, blue, red, blue and red, we would take the most frequent class and predict blue

When bagging with decision trees, **we are less concerned about individual trees overfitting the training data**. For this reason and for efficiency, the individual decision trees are grown deep (e.g. few training samples at each leaf-node of the tree) and the trees are not pruned. These trees will have both high variance and low bias. These are important characterize of sub-models when combining predictions using bagging.

The only **parameters when bagging decision trees is the number of samples and hence the number of trees to include**. This can be chosen by increasing the number of trees on run after run until the accuracy begins to stop showing improvement (e.g. on a cross validation test harness). Very large numbers of models may take a long time to prepare, but will not over fit the training data

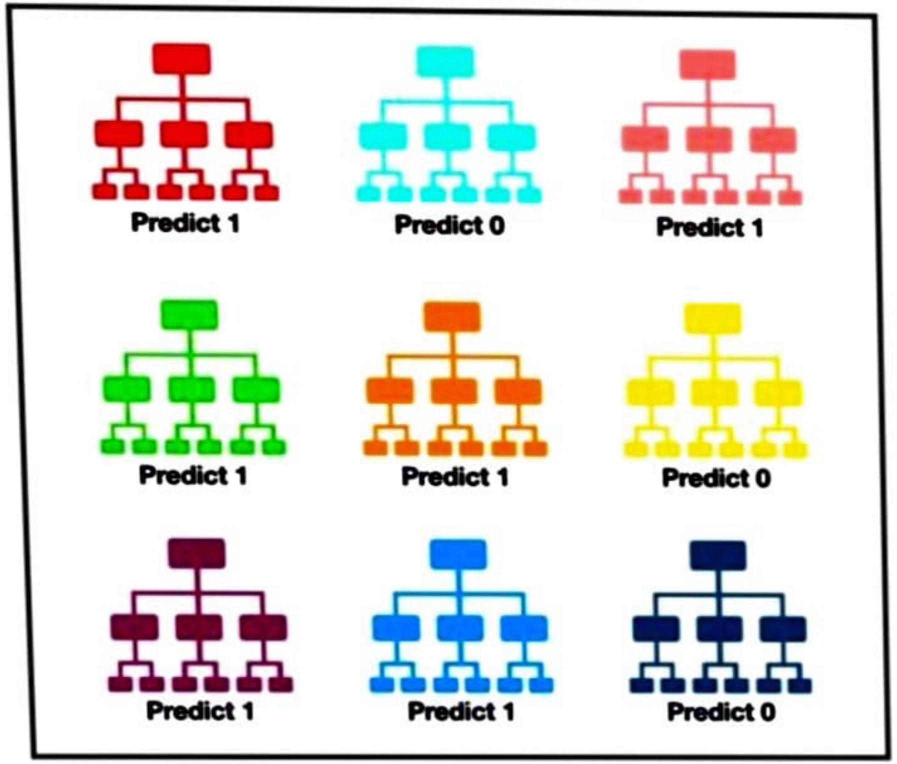
Random Forest

The random forest is a model made up of many decision trees. Rather than just simply averaging the prediction of trees

This model uses two key concepts that gives it the name random:
1. Random sampling of training data points when building trees
2. Random subsets of features considered when splitting nodes

When training, each tree in a random forest learns from a random sample of the data points. The samples are drawn with replacement, known as bootstrapping, which means that some samples will be used multiple times in a single tree

The other main concept in the random forest is that only a subset of all the features are considered for splitting each node in each decision tree. Generally this is set to sqrt(n_features) for classification meaning that if there are 16 features, at each node in each tree, only 4 random features will be considered for splitting the node.



Tally: Six 1s and Three 0s
Prediction: 1

The fundamental concept behind random forest is a simple but powerful one—the wisdom of crowds.

A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models

Gini Impurity

The Gini Impurity of a node is the probability that a randomly chosen sample in a node would be incorrectly labelled if it was labelled by the distribution of samples in the node. For example, in the top (root) node, there is a 44.4% chance of incorrectly classifying a data point chosen at random based on the sample labels in the node. We arrive at this value using the following equation

$$I_G(n) = 1 - \sum_{i=1}^{J} (p_i)^2$$

The Gini Impurity of a node n is 1 minus the sum over all the classes J (for a binary classification task this is 2) of the fraction of examples in each class p_i squared

$$I_{root} = 1 - ((\tfrac{2}{6})^2 + (\tfrac{4}{6})^2) = 1 - \tfrac{5}{9} = 0.444$$

At each node, the decision tree searches through the features for the value to split on that results in the *greatest reduction* in Gini Impurity.

Random Forest pseudocode:

1. Randomly select **“k”** features from total **“m”** features.
    1. Where **k << m**
2. Among the **“k”** features, calculate the node **“d”** using the best split point.
3. Split the node into **daughter nodes** using the **best split**.
4. Repeat **1 to 3** steps until “l” number of nodes has been reached.
5. Build forest by repeating steps **1 to 4** for “n” number times to create **“n” number of trees**.

Random Forest pseudocode:

```
1.Randomly select "k" features from total "m" features.
    1. Where k << m
2.Among the "k" features, calculate the node "d" using the best split point.
3.Split the node into daughter nodes using the best split.
4.Repeat 1 to 3 steps until "l" number of nodes has been reached.
5.Build forest by repeating steps 1 to 4 for "n" number times to create "n"
number of trees.
```

Advantages of random forest algorithm

- The same **random forest algorithm** or the random forest classifier can use for both classification and the regression task.
- Random forest classifier will **handle the missing** values
- The overfitting problem will never come when we use the random forest algorithm in any classification problem.
- The same random forest algorithm can be used for both classification and regression task.
- The random forest algorithm can be used for feature engineering.
    - Which means identifying the most important features out of the available features from the training dataset.

Boosting

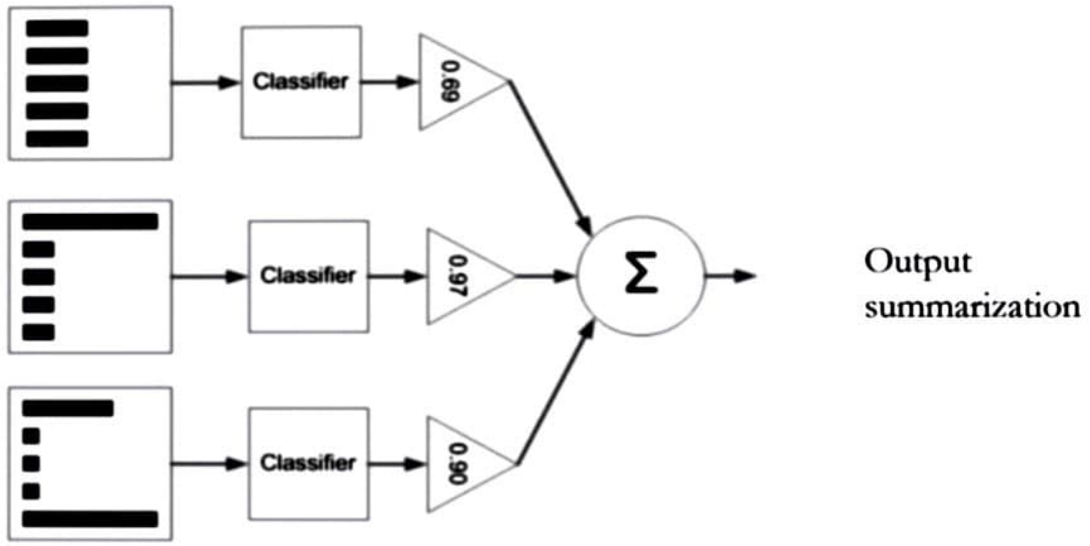
Boosting is an ensemble technique that attempts to create a strong classifier from a number of weak classifiers

This is done by building a model from the training data, then creating a second model that attempts to correct the errors from the first model. Models are added until the training set is predicted perfectly, or a maximum number of models are added.

AdaBoost was the first really successful boosting algorithm developed for binary classification and later on extended to multiclass problem. It is the best starting point for understanding boosting.

AdaBoost can be used to boost the performance of any machine learning algorithm. It is best used with weak learners. These are models that achieve accuracy just above random chance on a classification problem

Learning An AdaBoost Model From Data



1) Each sample have the same starting weight (1/*n*) initially
2) Fit a weak classifier using the weighted samples and compute the kth model's misclassification error (errk)
3) Compute the kth stage value as ln ((1 − errk) /errk)
4) Update the sample weights giving more weight to incorrectly predicted samples and less weight to correctly predicted samples

Where error is the mis-classification rate, correct are the number of training instance predicted correctly by the model and N is the total number of training instances. For example, if the model predicted 78 of 100 training instances correctly the error or mis-classification rate would be 78-100 /100 or 0.22.

$$error = \frac{correct - N}{N}$$

The above formula is modified to use the weightage of the training instances:

$$error = \frac{\sum_{i=1}^{n}(w_i \times perror_i)}{\sum_{i=1}^{n} w}$$

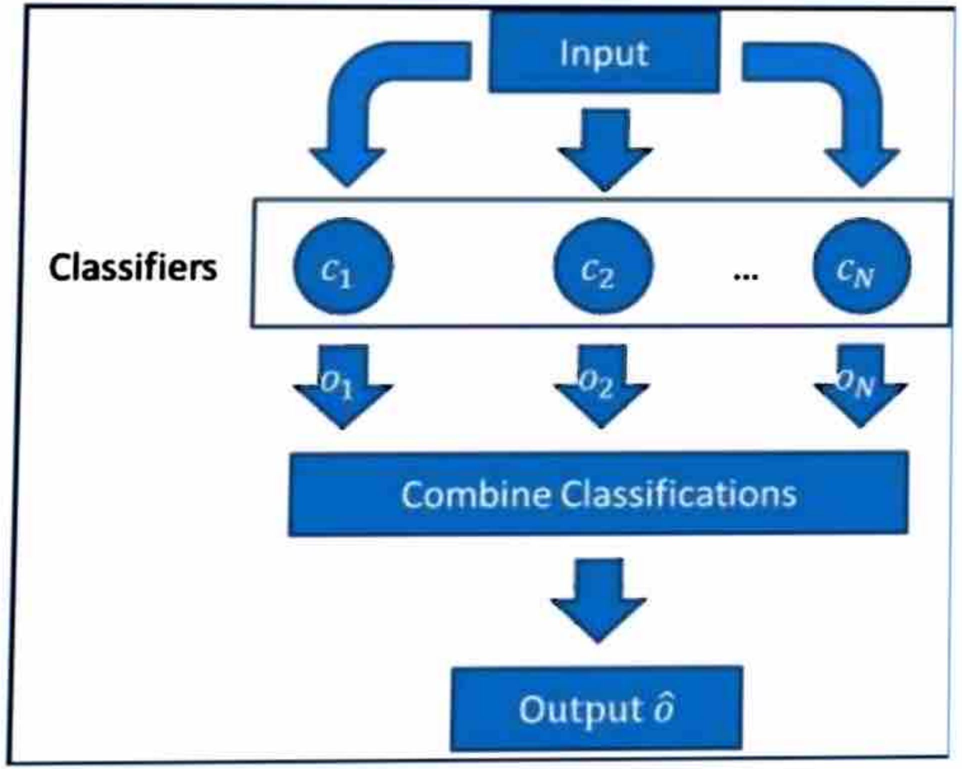
A stage value is calculated for the trained model which provides a weighting for any predictions that the model makes. The stage value for a trained model is calculated as follows:

$$stage = ln(\frac{1 - error}{error})$$

$$w = w \times e^{stage \times perror}$$

Stacking

- **Stacking**. Building multiple models (typically of differing types) and supervisor model that learns how to best combine the predictions of the primary models.


Input
Classifiers
c_1
c_2
...
c_N
o_1
o_2
o_N
Combine Classifications
Output $\hat{o}$