Ensemble Learning

Biswajit Biswas
Ph.D.(C.S.E)

Department of Computer Science
VIVEKANANDA CENTENARY COLLEGE
RAHARA, KOLKATA-700 118

March 18, 2025

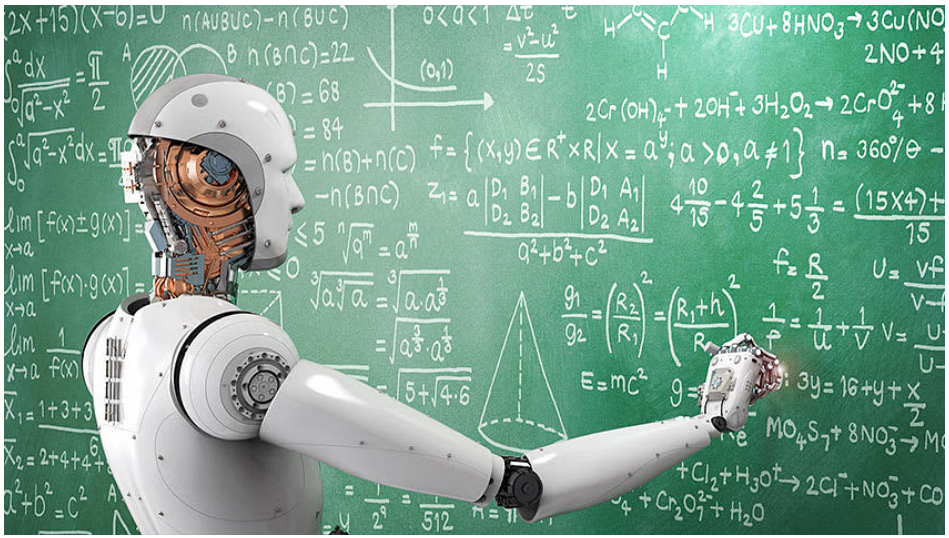
Ensemble Learning

Ensemble learning combines the predictions of multiple models (called "weak learners" or "base models") to make a stronger, more reliable prediction. The goal is to reduce errors and improve performance.

- Ensemble learning is a learning technique in which multiple individual models combine to create a master model;
- A machine learning technique that combines multiple models to improve prediction accuracy;
- A machine learning technique that aggregates two or more learners (e.g. regression models, neural networks) in order to produce better predictions;
- Ensemble techniques are effective because they reduce overfitting and improve generalization, leading to more robust models;

Ensemble Learning

- Most common ensemble technique is bagging, which uses bootstrap sampling to create multiple datasets from the original data and trains a model on each dataset.
- Boosting, which trains models sequentially, each focusing on the previous models' mistakes.
- Random forests are a popular ensemble method that uses decision trees as base learners and combines their predictions to make a final prediction.



Ensemble Learning

Types of Ensemble Learning in Machine Learning

There are two main types of ensemble methods:

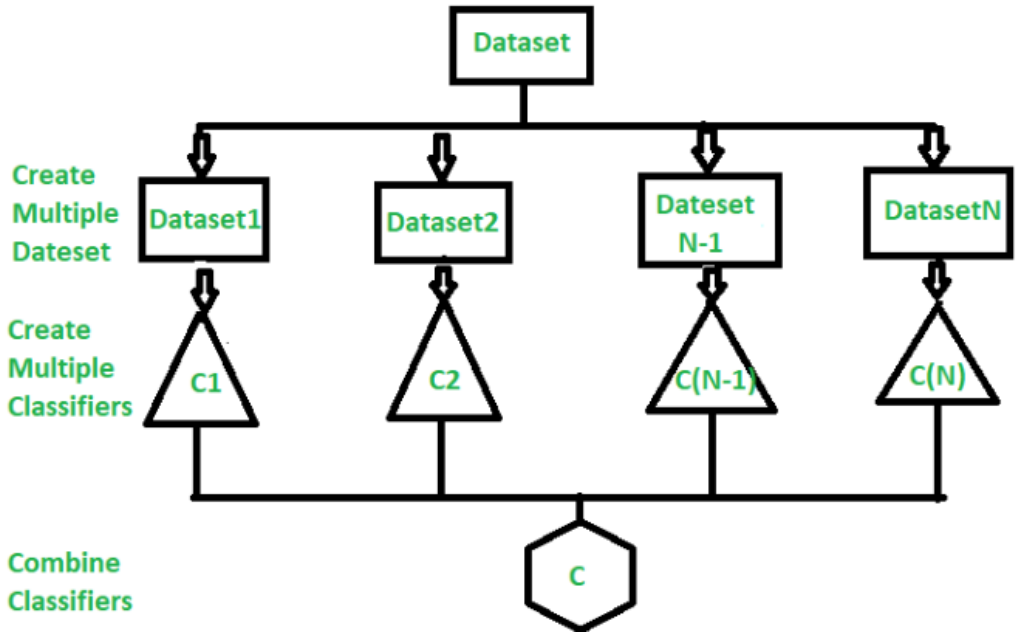
- **Bagging (Bootstrap Aggregating)**: Models are trained independently on different subsets of the data, and their results are averaged or voted on.
- **Boosting:** Models are trained sequentially, with each one learning from the mistakes of the previous model.

Think of it like asking multiple doctors for a diagnosis (bagging) or consulting doctors who specialize in correcting previous misdiagnoses (boosting).

Ensemble Classifier

Ensemble learning helps improve machine learning results by combining several models. This approach allows the production of better predictive performance compared to a single model. Basic idea is to learn a set of classifiers (experts) and to allow them to vote.

- **Advantage:** Improvement in predictive accuracy.
- **Disadvantage:** It is difficult to understand an ensemble of classifiers.



Why do ensembles work?

Mostly ensembles overcome three problems:

- **Statistical Problem:** The Statistical Problem arises when the hypothesis space is too large for the amount of available data. Hence, there are many hypotheses with the same accuracy on the data and the learning algorithm chooses only one of them! There is a risk that the accuracy of the chosen hypothesis is low on unseen data;
- **Computational Problem:** The Computational Problem arises when the learning algorithm cannot guarantees finding the best hypothesis;
- **Representational Problem:** The Representational Problem arises when the hypothesis space does not contain any good approximation of the target class(es);

Main Challenge for Developing Ensemble Models?

The main challenge is not to obtain highly accurate base models, but rather to obtain base models which make different kinds of errors. For example, if ensembles are used for classification, high accuracies can be accomplished if different base models misclassified different training examples, even if the base classifier accuracy is low.

Methods for Independently Constructing Ensembles:

- Majority Vote
- Bagging and Random Forest
- Randomness Injection
- Feature-Selection Ensembles
- Error-Correcting Output Coding

Main Challenge for Developing Ensemble Models?

Methods for Coordinated Construction of Ensembles:

- Boosting
- Stacking

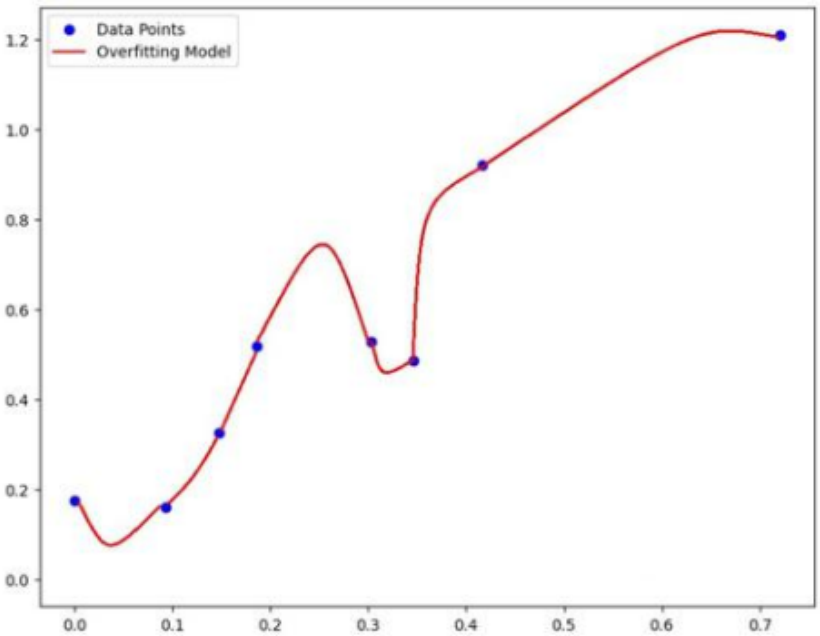
Reliable Classification: Meta-Classifier Approach:

- Co-Training and Self-Training

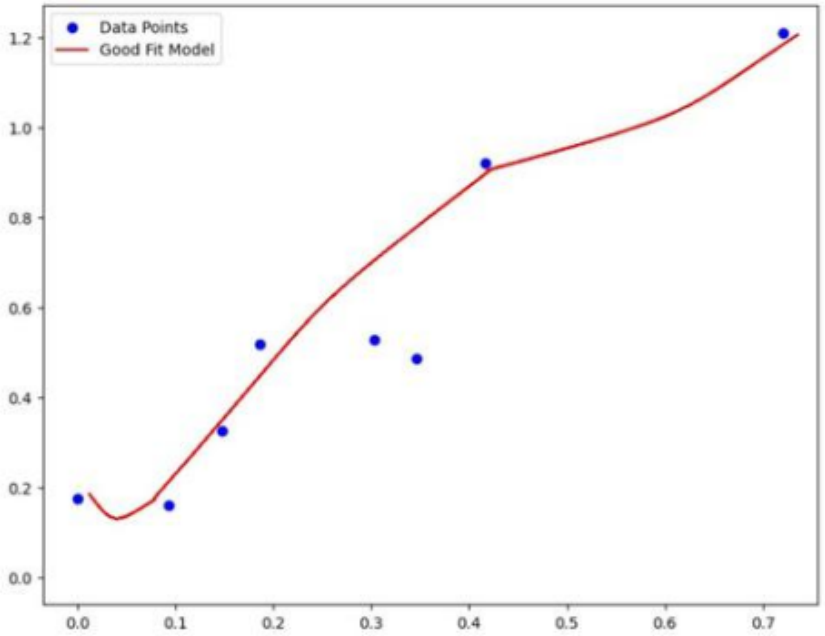
How does Ensemble Modeling Avoid Overfitting?

- **Bias:** Measures the average difference between predicted and actual values.
- **Variance:** Measures how predictions on the same data point vary.
- **Irreducible Error:** Represents data noise that cannot be minimized.

Bagging Algorithm



High Variance
(Overfitting)

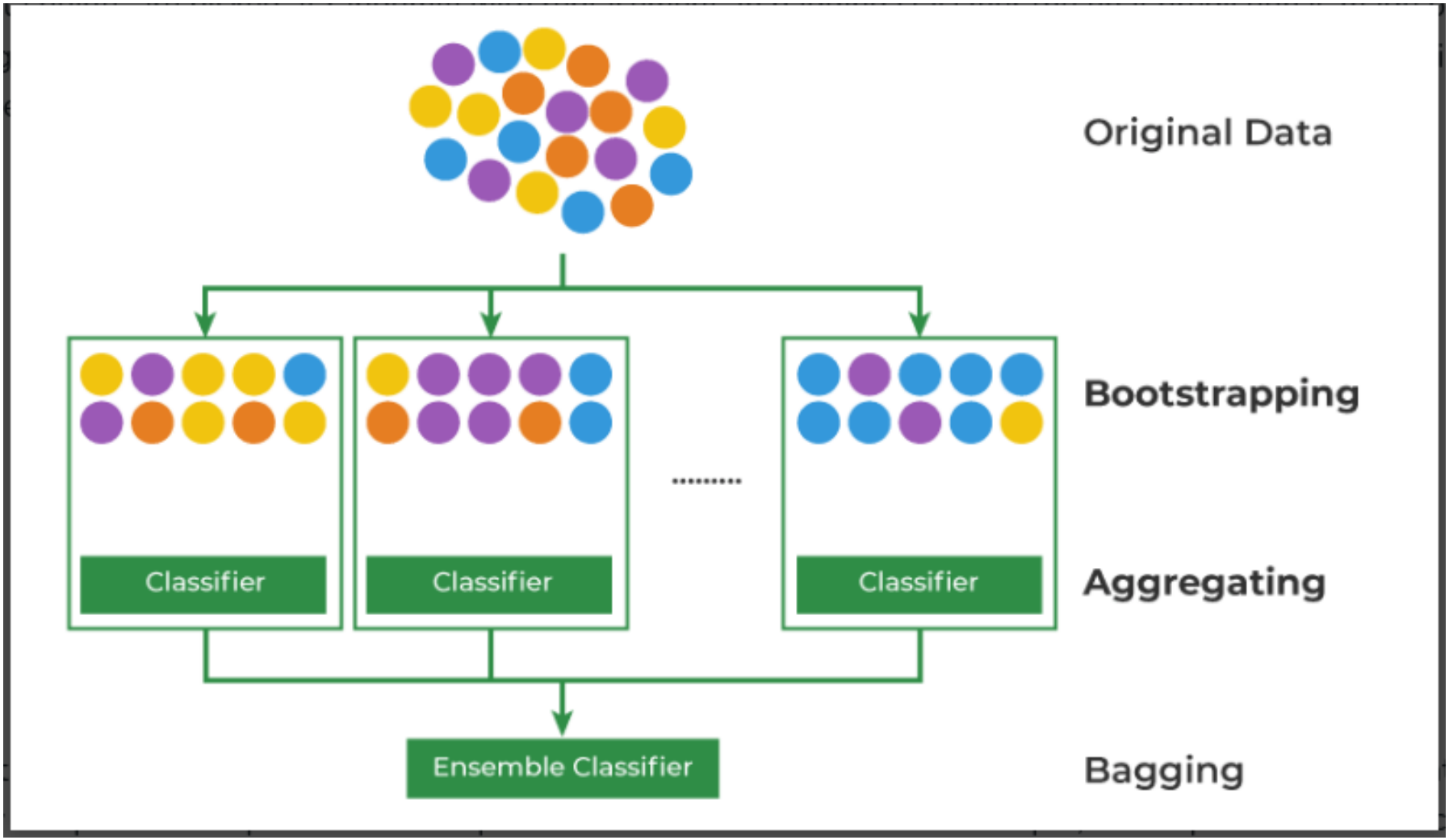


Low Variance, Low Bias
(Good Fit)

Ensemble Modeling

- Bagging or Bootstrap aggregating is a type of ensemble learning in which multiple base models are trained independently and parallelly on different subsets of training data.
- Each subset is generated using bootstrap sampling in which data points are picked at randomly with replacement.
- Bagging classifier the final prediction is made by aggregating the predictions of all base model using majority voting.
- This models of regression the final prediction is made by averaging the predictions of the all base model and that is known as bagging regression.

Bagging Algorithm



Model Implementation

The following Python code:

```
#Import Required Libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

#Generating and splitting the dataset
# Create synthetic data
X, y = make_regression(n_samples=30, n_features=1, noise=30, random_state=42)
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

#Model Implementation
# Create synthetic data
X, y = make_regression(n_samples=30, n_features=1, noise=30, random_state=42)
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Random Forest
rf = RandomForestRegressor(n_estimators=100, max_depth=5, random_state=123)
rf.fit(X_train, y_train)
```

Model Implementation

The following Python code:

```
# Gradient Boosting
gb = GradientBoostingRegressor(n_estimators=100, max_depth=5, random_state=123)
gb.fit(X_train, y_train)

# Decision Tree
dt = DecisionTreeRegressor(max_depth=3, random_state=123)
dt.fit(X_train, y_train)

#Calculating Accuracy
# Calculate accuracies
dt_accuracy_train = dt.score(X_train, y_train)
dt_accuracy_test = dt.score(X_test, y_test)
rf_accuracy_train = rf.score(X_train, y_train)
rf_accuracy_test = rf.score(X_test, y_test)
gb_accuracy_train = gb.score(X_train, y_train)
gb_accuracy_test = gb.score(X_test, y_test)

# Display accuracies
print(&quot;Decision Tree - Training Accuracy:&quot;, f&quot;{dt_accuracy_train:.2f}&quot;)
print(&quot;Decision Tree - Test Accuracy:&quot;, f&quot;{dt_accuracy_test:.2f}&quot;)
print(&quot;Random Forest - Training Accuracy:&quot;, f&quot;{rf_accuracy_train:.2f}&quot;)
print(&quot;Random Forest - Test Accuracy:&quot;, f&quot;{rf_accuracy_test:.2f}&quot;)
print(&quot;Gradient Boosting - Training Accuracy:&quot;, f&quot;{gb_accuracy_train:.2f}&quo
print(&quot;Gradient Boosting - Test Accuracy:&quot;, f&quot;{gb_accuracy_test:.2f}&quot;)
```

Bagging Algorithm

Starting with an original dataset containing multiple data points (represented by colored circles). The original dataset is randomly sampled with replacement multiple times. This means that in each sample, a data point can be selected more than once or not at all. These samples create multiple subsets of the original data.

- For each of the bootstrapped subsets, a separate classifier (e.g., decision tree, logistic regression) is trained.
- The predictions from all the individual classifiers are combined to form a final prediction. This is often done through a majority vote (for classification) or averaging (for regression).

How does Bagging Classifier Work?

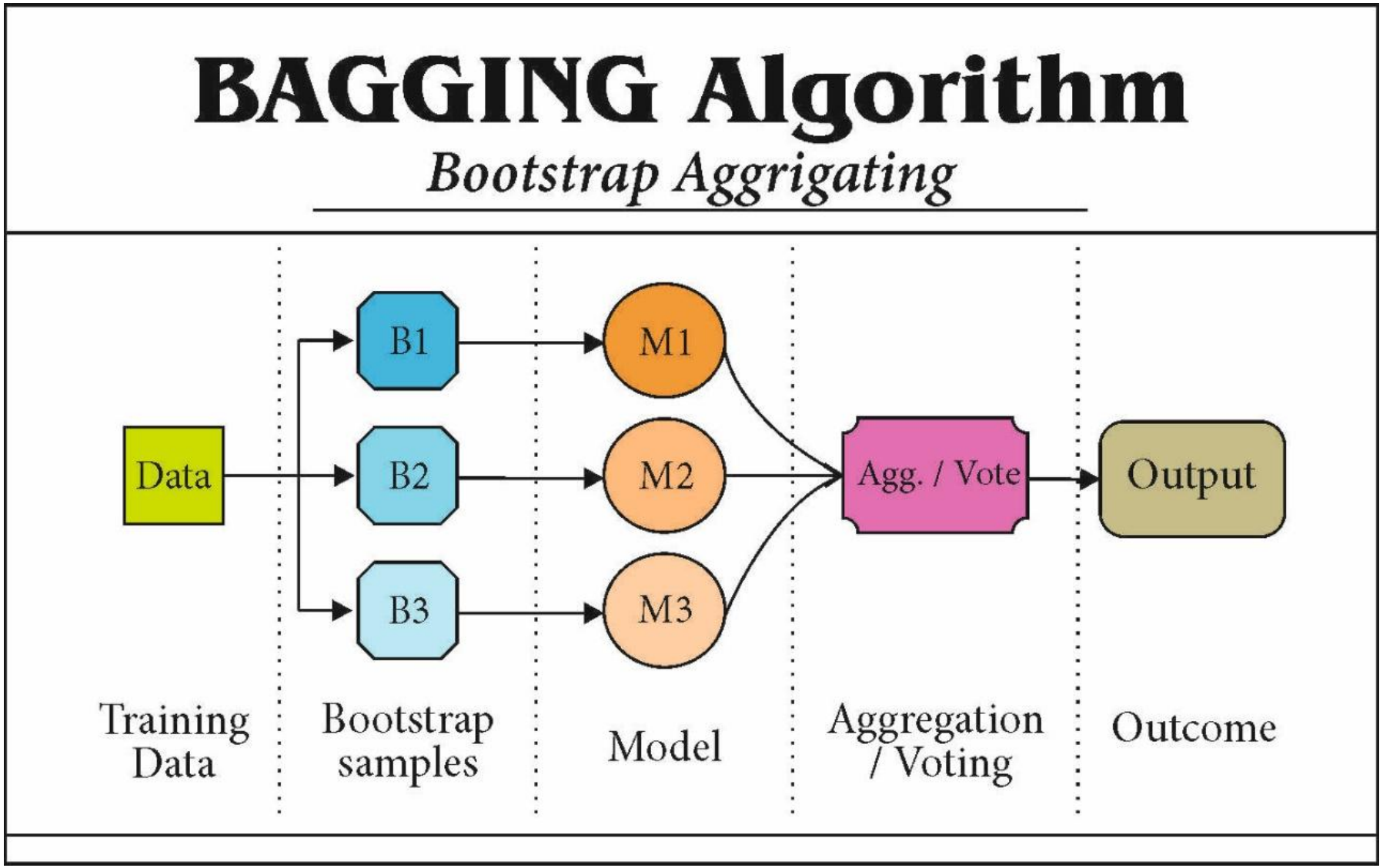
An overview of Bagging classifier algorithm:

- **Bootstrap Sampling:** Divides the original training data into 'N' subsets and randomly selects a subset with replacement in some rows from other subsets. This step ensures that the base models are trained on diverse subsets of the data and there is no class imbalance.
- **Base Model Training:** For each bootstrapped sample we train a base model independently on that subset of data. These weak models are trained in parallel to increase computational efficiency and reduce time consumption. We can use different base learners i.e different ML models as base learners to bring variety and robustness.

Bagging Algorithm

- **Prediction Aggregation:** To make a prediction on testing data combine the predictions of all base models. For classification tasks it can include majority voting or weighted majority while for regression it involves averaging the predictions.
- **Out-of-Bag (OOB) Evaluation:** Some samples are excluded from the training subset of particular base models during the bootstrapping method. These "out-of-bag" samples can be used to estimate the model's performance without the need for cross-validation.
- **Final Prediction:** After aggregating the predictions from all the base models, Bagging produces a final prediction for each instance.

Bagging Algorithm

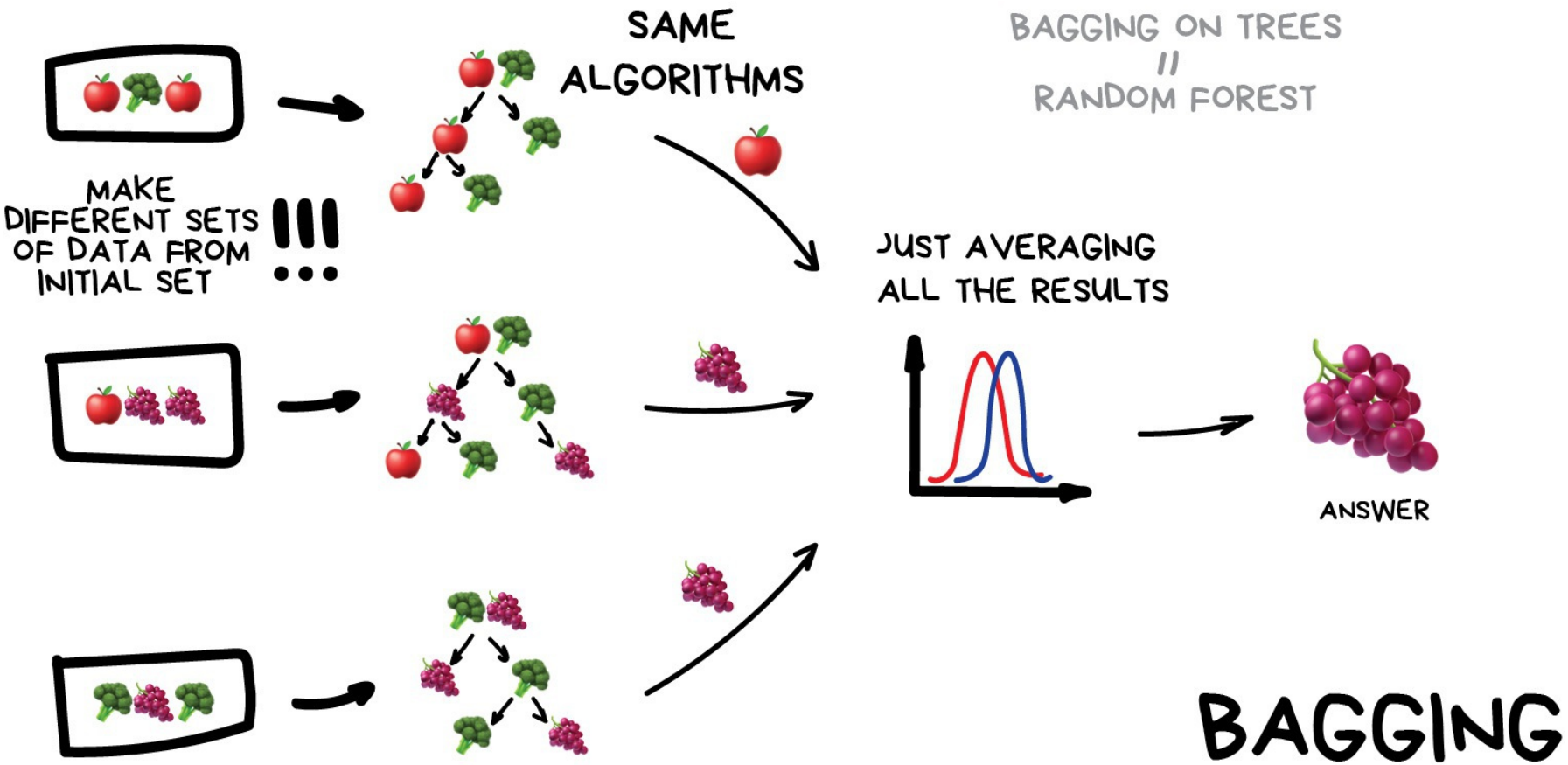


Bagging Algorithm

Advantages of Bagging Classifier:

- **Improved Predictive Performance:** It outperforms single classifiers by reducing overfitting and increasing predictive accuracy by combining multiple base models.
- **Robustness:** Reduces the impact of outliers and noise in data by aggregating predictions from multiple models. This enhances the overall stability and robustness of the model.
- **Reduced Variance:** Since each base model is trained on different subsets of the data the aggregated model's variance is significantly reduced compared to individual model.
- **Parallel Working:** Bagging allows parallel processing as each base model can be trained independently in parallel. This makes it computationally efficient for large datasets.
- **Flexibility:** It can be applied to wide range of machine learning algorithms such as decision trees, random forests and support vector machines.

Bagging Algorithm



Bagging Algorithm

Disadvantages of Bagging:

- **Loss of Interpretability:** It involves aggregating predictions from multiple models making it harder to interpret individual base model.
- **Computationally Expensive:** As the number of iterations of bootstrap samples increases, computational cost of bagging also increase.

Bagging Algorithm

The following Python code:

```
#Importing Libraries and Loading Data

from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


#Loading and Splitting the Iris Dataset
data = load_iris()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#Creating a Base Classifier
base_classifier = DecisionTreeClassifier()

#Creating and Training the Bagging Classifier
bagging_classifier = BaggingClassifier(base_classifier, n_estimators=10, random_state=42)
bagging_classifier.fit(X_train, y_train)

#Making Predictions and Evaluating Accuracy
y_pred = bagging_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Applications of Bagging Classifier

- **Fraud Detection:** It can be used to detect fraudulent transactions by aggregating predictions from multiple fraud detection models.
- **Spam filtering:** It can be used to filter spam emails by aggregating predictions from multiple spam filters trained on different subsets of the spam emails.
- **Credit scoring:** Can be used to improve the accuracy of credit scoring models by combining multiple models.
- **Image Classification:** It can be used to improve the accuracy of image classification task.
- **Natural language processing:** In NLP tasks it can combine predictions from multiple language models to achieve better text classification results.

Popular Ensemble Methods Apart from Bagging and Boosting

Elbow Method:

- **Description:** The Elbow Method involves plotting the sum of squared distances from each point to its assigned cluster center (inertia) against the number of clusters. As the number of clusters increases, inertia decreases. The goal is to identify the point where the inertia no longer decreases significantly with each additional cluster. This point resembles an "elbow" in the plot.
- **Application:** This method is straightforward and visually intuitive, commonly used with K-means clustering. The "elbow" point suggests a diminishing return on the addition of more clusters and is considered the optimal number.

Popular Ensemble Methods

Silhouette Score Method:

- **Description:** The Silhouette Score measures how similar an object is to its own cluster compared to other clusters. The score ranges from -1 to 1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters.
- **Application:** By calculating the average Silhouette Score for different numbers of clusters, you can determine the optimal number. The configuration with the highest average Silhouette Score typically provides the best separation and cohesion of clusters.

Popular Ensemble Methods

Gap Statistic Method:

- **Description:** The Gap Statistic compares the total intra-cluster variation for different numbers of clusters with their expected values under a null reference distribution of the data, i.e., a distribution with no obvious clustering. The optimal clusters will be the value at which the gap statistic is maximized, indicating that the clustering structure is significantly better than random.
- **Application:** This method involves more computation but provides a more robust statistical basis for determining the number of clusters. It's especially useful when the data structure is ambiguous or when validating the presence of clusters.

Popular Ensemble Methods

Davies-Bouldin Index Method:

- **Description:** This index evaluates the clustering quality by measuring the average 'similarity' between each cluster and its most similar one, where similarity is a combination of cluster compactness and separation. The optimal number of clusters is indicated by the lowest Davies-Bouldin Index.
- **Application:** The index is straightforward to compute and does not require knowledge of labels, making it effective for datasets where cluster structures are not clearly defined.

The AdaBoost Algorithm

Notations:

- Let $\mathcal{X}$ denote the instance space, or feature space and $\mathcal{Y}$ denote the set of labels that express the underlying concepts which are to be learned;
- For example, let $\mathcal{Y} = \{-1, +1\}$ for binary classification;
- A training set $\mathcal{D}$ consists of m instances whose associated labels are observed, i.e., $\mathcal{D} = \{(x_i, y_i)\}(i \in \{1, \cdots, m)\}$, while the label of a test instance is unknown and thus to be predicted;
- After training on a training data set $\mathcal{D}$, a learning algorithm $\mathcal{L}$ will output a hypothesis h, which is a mapping from $\mathcal{X}$ to $\mathcal{Y}$, or called as a classifier, i.e.

$$\mathcal{L} : \mathcal{X} \to \mathcal{Y}$$

- The learning process mainly picking the best hypothesis from a hypothesis space, where the word "best" refers to a loss function;

The AdaBoost Algorithm

- For classification, the loss function can naturally be 0/1-loss,i.e.

$$\mathsf{L}_{0/1}(h|x) = \mathsf{I}[h(\mathsf{x}) \neq y]$$

where $\mathsf{I}[\cdot]$ is the indication function which outputs 1 if the inner expression is true and 0 otherwise, which means that one error is counted if an instance is wrongly classified.

Input: Instance distribution $\mathcal{D}$;
Base learning algorithm L;
Number of learning rounds T.

Process:

1. $\mathcal{D}_1 = \mathcal{D}$. % Initialize distribution
2. **for** $t = 1, \cdots, T$:
3. $h_t = L(\mathcal{D}_t)$; % Train a weak learner from distribution $\mathcal{D}_t$
4. $\epsilon_t = \Pr_{\boldsymbol{x} \sim \mathcal{D}_t, y} \boldsymbol{I}[h_t(\boldsymbol{x}) \neq y]$; % Measure the error of h_t
5. $\mathcal{D}_{t+1} = AdjustDistribution\ (\mathcal{D}_t, \epsilon_t)$
6. **end**

Output: $H(\boldsymbol{x}) = CombineOutputs(\{h_t(\boldsymbol{x})\})$

The AdaBoost Algorithm

What is AdaBoost?

- AdaBoost is an algorithm for constructing a ”strong” classifier as linear combination

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

of “simple” “weak” classifiers $h_t(x)$.

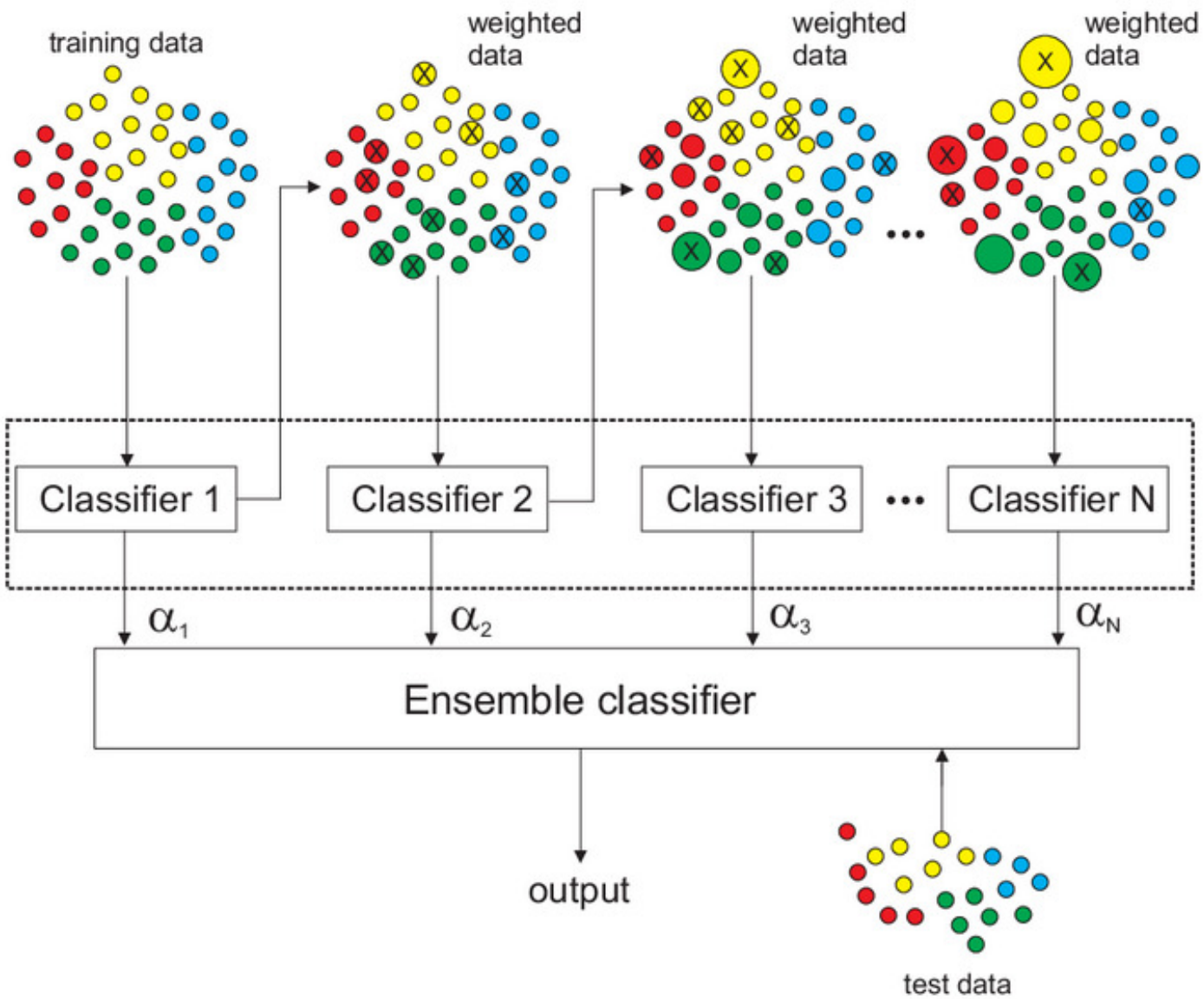
Terminology

- $h_t(x)$ ... “weak” or basis classifier, hypothesis, ”feature”
- $H(x) = \textit{sign}(f(x))$ ... ‘’strong” or final classifier/hypothesis

Comments

- The $h_t(x)$’s can be thought of as features.
- Often (typically) the set $\mathcal{H} = \{h(x)\}$ is infinite.

The AdaBoost Algorithm



The AdaBoost Algorithm

Given: $(x_1, y_1), ..., (x_m, y_m)$; $x_i \in \mathcal{X}, y_i \in \{-1, 1\}$

Initialize weights $D_1(i) = 1/m$

For $t = 1, ..., T$:

1. (Call *WeakLearn*), which returns the weak classifier $h_t : \mathcal{X} \rightarrow \{-1, 1\}$ with minimum error w.r.t. distribution D_t;
2. Choose $\alpha_t \in R$,
3. Update

$$D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor chosen so that D_{t+1} is a distribution

Output the strong classifier:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

The AdaBoost Algorithm

Given: $(x_1, y_1), ..., (x_m, y_m)$; $x_i \in \mathcal{X}, y_i \in \{-1, 1\}$

Initialize weights $D_1(i) = 1/m$

For $t = 1, ..., T$:

1. (Call *WeakLearn*), which returns the weak classifier $h_t : \mathcal{X} \to \{-1, 1\}$ with minimum error w.r.t. distribution D_t;
2. Choose $\alpha_t \in R$,
3. Update

$$D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor chosen so that D_{t+1} is a distribution

Output the strong classifier:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Comments

- The computational complexity of selecting h_t is independent of t.
- All information about previously selected "features" is captured in D_t!

The AdaBoost Algorithm

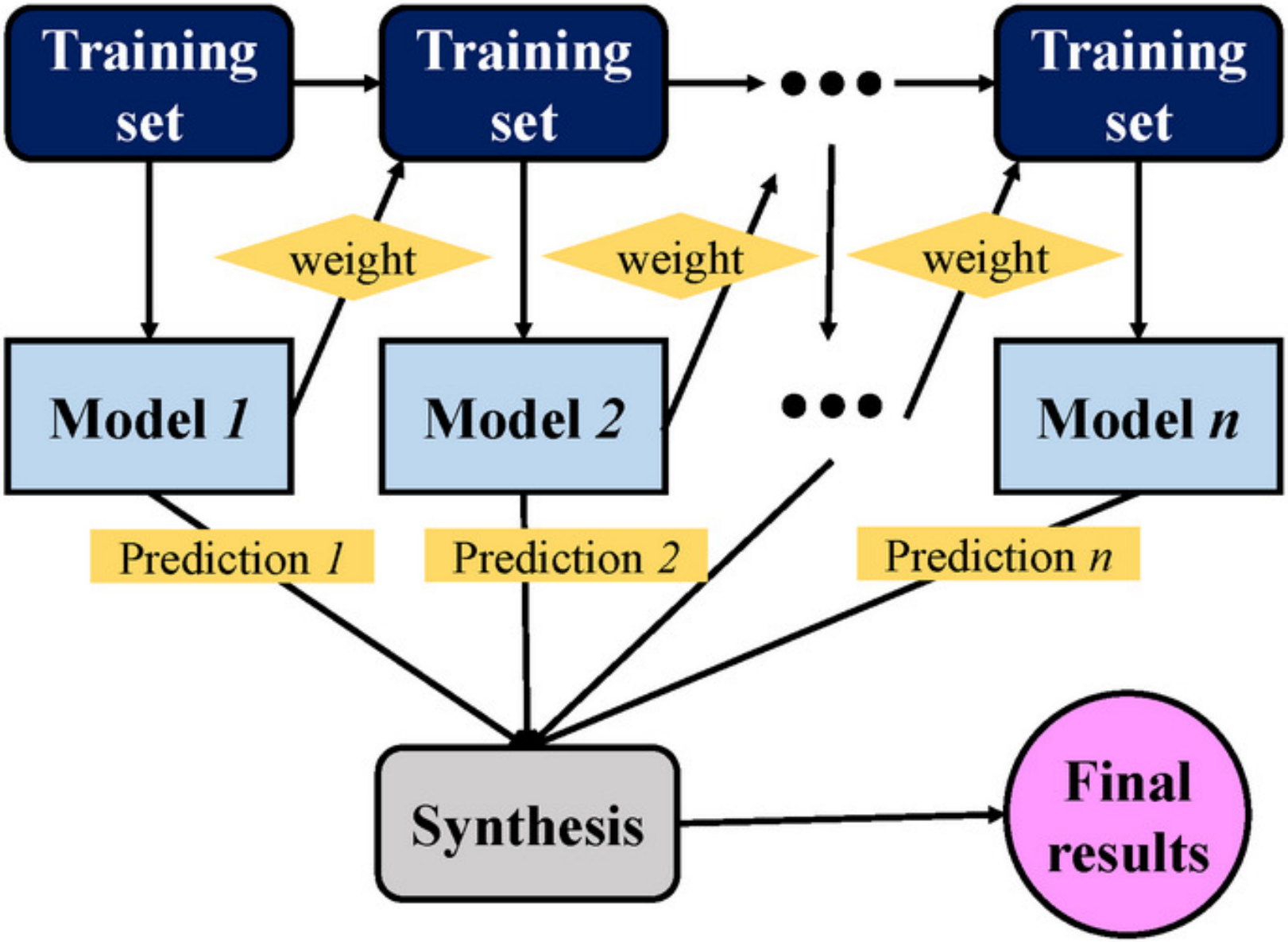
Input: Data set $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
Base learning algorithm L;
Number of learning rounds T.

Process:

1. $\mathcal{D}_1(i) = 1/m$. % Initialize the weight distribution
2. **for** $t = 1, \cdots, T$:
3. $h_t = L(D, \mathcal{D}_t)$; % Train a learner h_t from D using distribution $\mathcal{D}_t$
4. $\epsilon_t = \Pr_{\boldsymbol{x} \sim \mathcal{D}_t, y} \boldsymbol{I}[h_t(\boldsymbol{x}) \neq y]$; % Measure the error of h_t
5. **if** $\epsilon_t > 0.5$ **then break**
6. $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$; % Determine the weight of h_t
7. $\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } h_t(\boldsymbol{x}_i) = y_i \\ \exp(\alpha_t) & \text{if } h_t(\boldsymbol{x}_i) \neq y_i \end{cases}$
$= \frac{\mathcal{D}_t(i) \exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))}{Z_t}$ % Update the distribution, where
% Z_t is a normalization factor which
% enables $\mathcal{D}_{t+1}$ to be distribution
8. **end**

Output: $H(\boldsymbol{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})\right)$

The AdaBoost Algorithm



The AdaBoost Algorithm

Weak-Learn:

Loop step: Call *WeakLearn*, given distribution D_t;
returns weak classifier $h_t : \mathcal{X} \rightarrow \{-1, 1\}$ from $\mathcal{H} = \{h(x)\}$

- Select a weak classifier with the smallest weighted error
  $h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$
- Prerequisite: $\epsilon_t < 1/2$ (otherwise stop)
- *WeakLearn* examples:
  - Decision tree builder, perceptron learning rule – $\mathcal{H}$ *infinite*
  - Selecting the best one from given *finite* set $\mathcal{H}$

The AdaBoost Algorithm

- The main objective is to minimize $\varepsilon_{tr} = \frac{1}{m}|\{i : H(x_i) \neq y_i\}|$
- It can be upper bounded by $\varepsilon_{tr}(H) \leq \prod_{t=1}^{T} Z_t$

How to set α_t?

- Select α_t to greedily minimize $Z_t(\alpha)$ in each step
- $Z_t(\alpha)$ is convex differentiable function with one extremum

  $\Rightarrow$ $h_t(x) \in \{-1, 1\}$ then optimal $\alpha_t = \frac{1}{2}\log(\frac{1+r_t}{1-r_t})$

  where $r_t = \sum_{i=1}^{m} D_t(i)h_t(x_i)y_i$
- $Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)} \leq 1$ for optimal α_t

  $\Rightarrow$ Justification of selection of h_t according to ϵ_t

Comments

- The process of selecting α_t and $h_t(x)$ can be interpreted as a single optimization step minimising the upper bound on the empirical error. Improvement of the bound is guaranteed, provided that $\epsilon_t < 1/2$.
- The process can be interpreted as a component-wise local optimization (Gauss-Southwell iteration) in the (possibly infinite dimensional!) space of $\bar{\alpha} = (\alpha_1, \alpha_2, \dots)$ starting from. $\bar{\alpha}_0 = (0, 0, \dots)$.

Does AdaBoost generalize?

Margins in SVM

$$\max \min_{(x,y)\in S} \frac{y(\vec{\alpha}\cdot\vec{h}(x))}{\|\vec{\alpha}\|_2}$$

Margins in AdaBoost

$$\max \min_{(x,y)\in S} \frac{y(\vec{\alpha}\cdot\vec{h}(x))}{\|\vec{\alpha}\|_1}$$

Maximizing margins in AdaBoost

$$P_S[yf(x) \le \theta] \le 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t^{1-\theta}(1-\epsilon_t)^{1+\theta}} \qquad \text{where } f(x) = \frac{\vec{\alpha}\cdot\vec{h}(x)}{\|\vec{\alpha}\|_1}$$

Upper bounds based on margin

$$P_{\mathcal{D}}[yf(x) \le 0] \le P_S[yf(x) \le \theta] + \mathcal{O}\left(\frac{1}{\sqrt{m}}\left(\frac{d\log^2(m/d)}{\theta^2} + \log(1/\delta)\right)^{1/2}\right)$$

Does AdaBoost generalize?

Advantages

- Very simple to implement
- Feature selection on very large sets of features
- Fairly good generalization

Disadvantages

- Suboptimal solution for $\bar{\alpha}$
- Can overfit in presence of noise

AdaBoost variants

- Discrete ($h : \mathcal{X} \rightarrow \{0, 1\}$)
- Multiclass AdaBoost.M1 ($h : \mathcal{X} \rightarrow \{0, 1, ..., k\}$)
- Multiclass AdaBoost.M2 ($h : \mathcal{X} \rightarrow [0, 1]^k$)
- Real valued AdaBoost.R ($Y = [0, 1]$, $h : \mathcal{X} \rightarrow [0, 1]$)

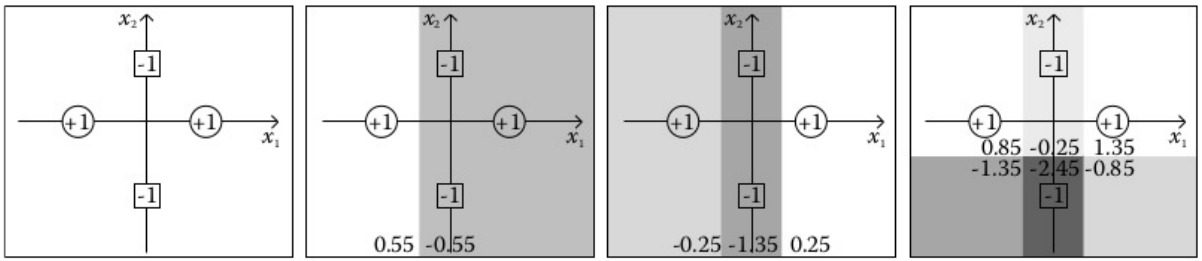
AdaBoost Solver: Solving XOR Problem

We consider an artificial data set in a two-dimensional space, that is,

$$\left\{\begin{array}{l}(\boldsymbol{x}_1 = (0, +1), y_1 = +1)\\ (\boldsymbol{x}_2 = (0, -1), y_2 = +1)\\ (\boldsymbol{x}_3 = (+1, 0), y_3 = -1)\\ (\boldsymbol{x}_4 = (-1, 0), y_4 = -1)\end{array}\right\}$$

$$h_1(\boldsymbol{x}) = \begin{cases} +1, & \text{if } (x_1 > -0.5)\\ -1, & \text{otherwise}\end{cases} \qquad h_2(\boldsymbol{x}) = \begin{cases} -1, & \text{if } (x_1 > -0.5)\\ +1, & \text{otherwise}\end{cases}$$

$$h_3(\boldsymbol{x}) = \begin{cases} +1, & \text{if } (x_1 > +0.5)\\ -1, & \text{otherwise}\end{cases} \qquad h_4(\boldsymbol{x}) = \begin{cases} -1, & \text{if } (x_1 > +0.5)\\ +1, & \text{otherwise}\end{cases}$$



(a) The XOR data (b) 1st round (c) 2nd round (d) 3rd round

$$h_5(\boldsymbol{x}) = \begin{cases} +1, & \text{if } (x_2 > -0.5)\\ -1, & \text{otherwise}\end{cases} \qquad h_6(\boldsymbol{x}) = \begin{cases} -1, & \text{if } (x_2 > -0.5)\\ +1, & \text{otherwise}\end{cases}$$

$$h_7(\boldsymbol{x}) = \begin{cases} +1, & \text{if } (x_2 > +0.5)\\ -1, & \text{otherwise}\end{cases} \qquad h_8(\boldsymbol{x}) = \begin{cases} -1, & \text{if } (x_2 > +0.5)\\ +1, & \text{otherwise}\end{cases}$$

The following Python code:

```
#Importing Libraries and Modules
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

#Loading and Splitting the Dataset
data = load_iris()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Defining the Weak Learner
base_classifier = DecisionTreeClassifier(max_depth=1)

#Creating and Training the AdaBoost Classifier
adaboost_classifier = AdaBoostClassifier( base_classifier, n_estimators=50,
learning_rate=1.0, random_state=42)
adaboost_classifier.fit(X_train, y_train)

#Making Predictions and Calculating Accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

THANK YOU