AdaBoost

AdaBoost (short for Adaptive Boosting) is a popular ensemble learning technique that combines multiple weak learners (usually decision stumps) to create a strong learner.

It focuses on sequentially improving the weak learners by assigning higher weights to misclassified samples, so subsequent learners focus on these harder-to-classify instances.

Boosting

- **Boosting**: Ensemble method combining several weak learners to form a strong learner.
- **Weak learner**: Model doing slightly better than random guessing.
- Example of weak learner: Decision stump (CART whose maximum depth is 1).

Boosting

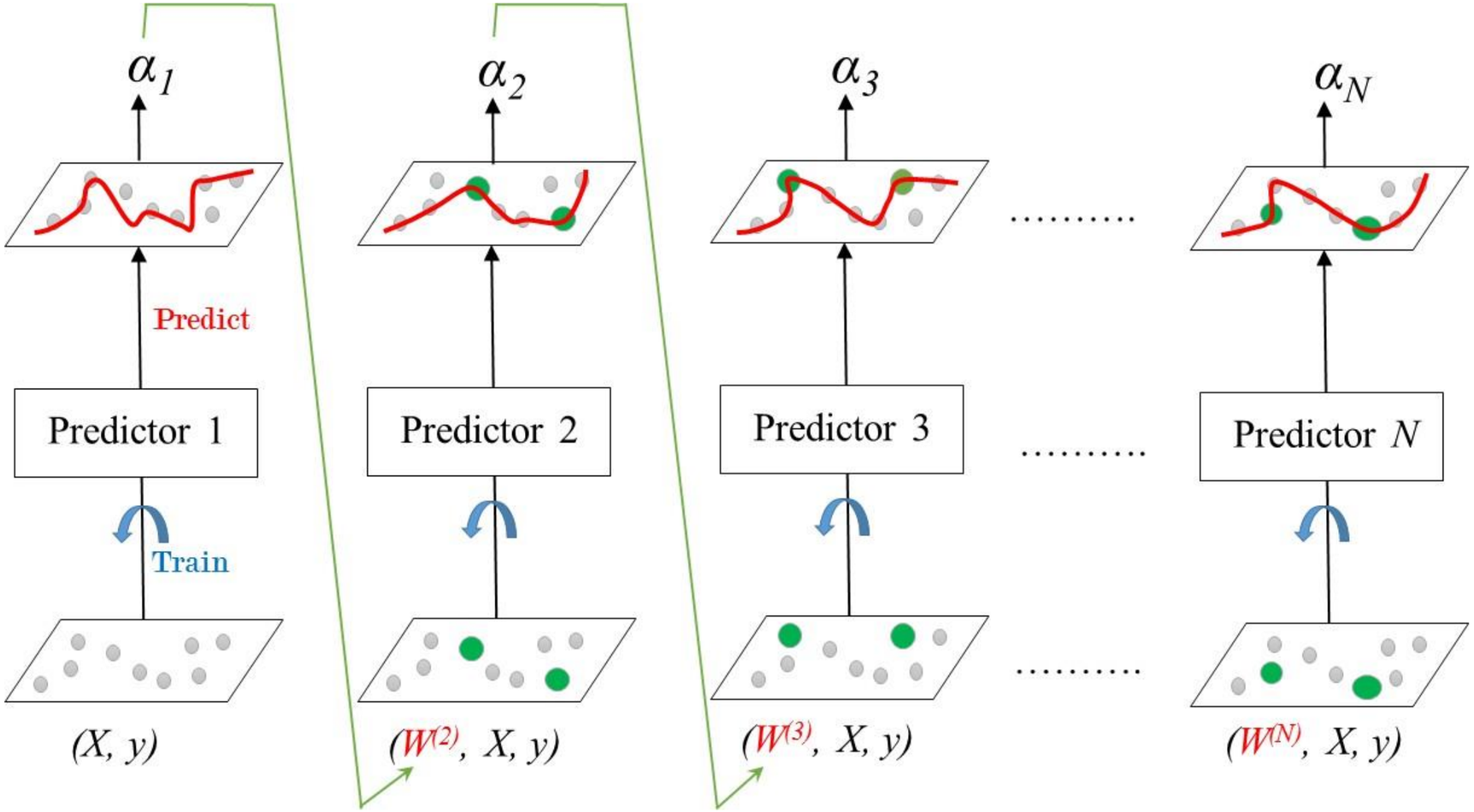
- Train an ensemble of predictors sequentially.
- Each predictor tries to correct its predecessor.
- Most popular boosting methods:
  - AdaBoost,
  - Gradient Boosting.

Adaboost

- Stands for **Ada**ptive **Boost**ing.
- Each predictor pays more attention to the instances wrongly predicted by its predecessor.
- Achieved by changing the weights of training instances.
- Each predictor is assigned a coefficient α.
- α depends on the predictor's training error.

AdaBoost: Training

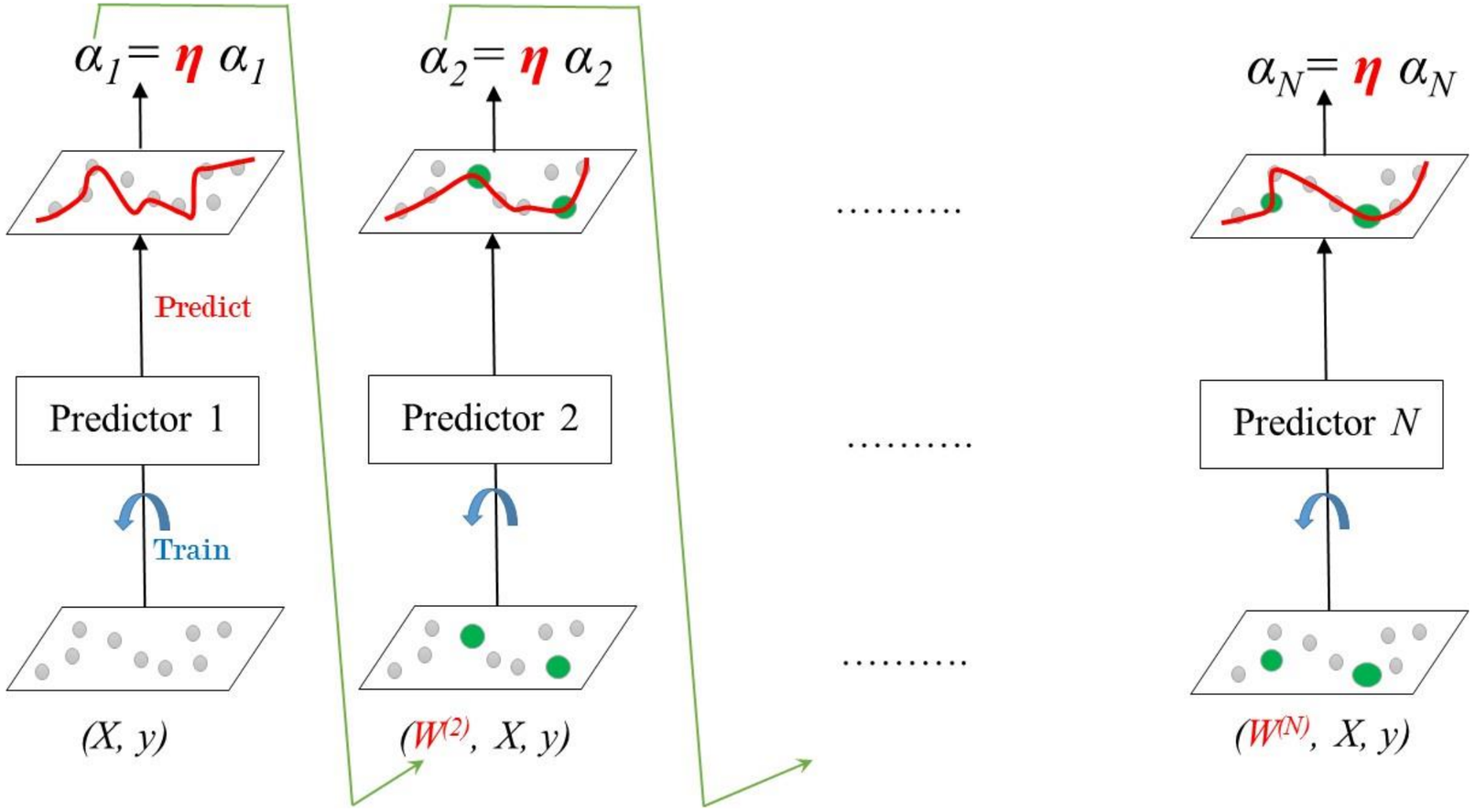
The diagram shows how AdaBoost sequentially trains multiple weak learners (predictors) by focusing on **misclassified samples** and combines their predictions using weighted voting.



- Each box labeled **Predictor 1**, **Predictor 2**, etc., represents a weak learner (e.g., decision stump).They predict and improve sequentially by focusing on the mistakes of previous predictors.
- Arrows labeled "Train" indicate that each predictor is trained on weighted data.
- The dots (green or gray) on the dataset panels represent individual samples.Misclassified samples (gray dots) have their weights increased, making them larger in subsequent panels.
- The arrows labeled α1,α2,α3,…\alpha_1, \alpha_2, \alpha_3, \dotsα1,α2,α3,… indicate the weight of each predictor in the final ensemble. Stronger predictors (lower error) are given higher weights.

Learning Rate

Learning rate: $0 < \eta \leq 1$



Learning Rate: The learning rate (η) scales the contribution of each predictor (alpha). Smaller η leads to more gradual learning, improving robustness.
Sequential Training: Weak learners focus more on misclassified samples, as seen by the increasing size of green dots (sample weights).
Weighted Predictions: The predictors' contributions to the final ensemble are adjusted by both their performance and the learning rate.

AdaBoost: Prediction

- Classification:
  - Weighted majority voting.
  - In sklearn: `AdaBoostClassifier`.
- Regression:
  - Weighted average.
  - In sklearn: `AdaBoostRegressor`.

AdaBoost Classification in sklearn (Breast Cancer dataset)

```
# Import models and utility functions
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split

# Set seed for reproducibility
SEED = 1

# Split data into 70% train and 30% test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    stratify=y,
                                                    random_state=SEED)
```

```
# Instantiate a classification-tree 'dt'
dt = DecisionTreeClassifier(max_depth=1, random_state=SEED)

# Instantiate an AdaBoost classifier 'adab_clf'
adb_clf = AdaBoostClassifier(base_estimator=dt, n_estimators=100)


# Fit 'adb_clf' to the training set
adb_clf.fit(X_train, y_train)

# Predict the test set probabilities of positive class
y_pred_proba = adb_clf.predict_proba(X_test)[:,1]


# Evaluate test-set roc_auc_score
adb_clf_roc_auc_score = roc_auc_score(y_test, y_pred_proba)
```

AdaBoost Classification in sklearn (Breast Cancer dataset)

```
# Print adb_clf_roc_auc_score
print('ROC AUC score: {:.2f}'.format(adb_clf_roc_auc_score))
```

```
ROC AUC score: 0.99
```

Gradient Boosting (GB)

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

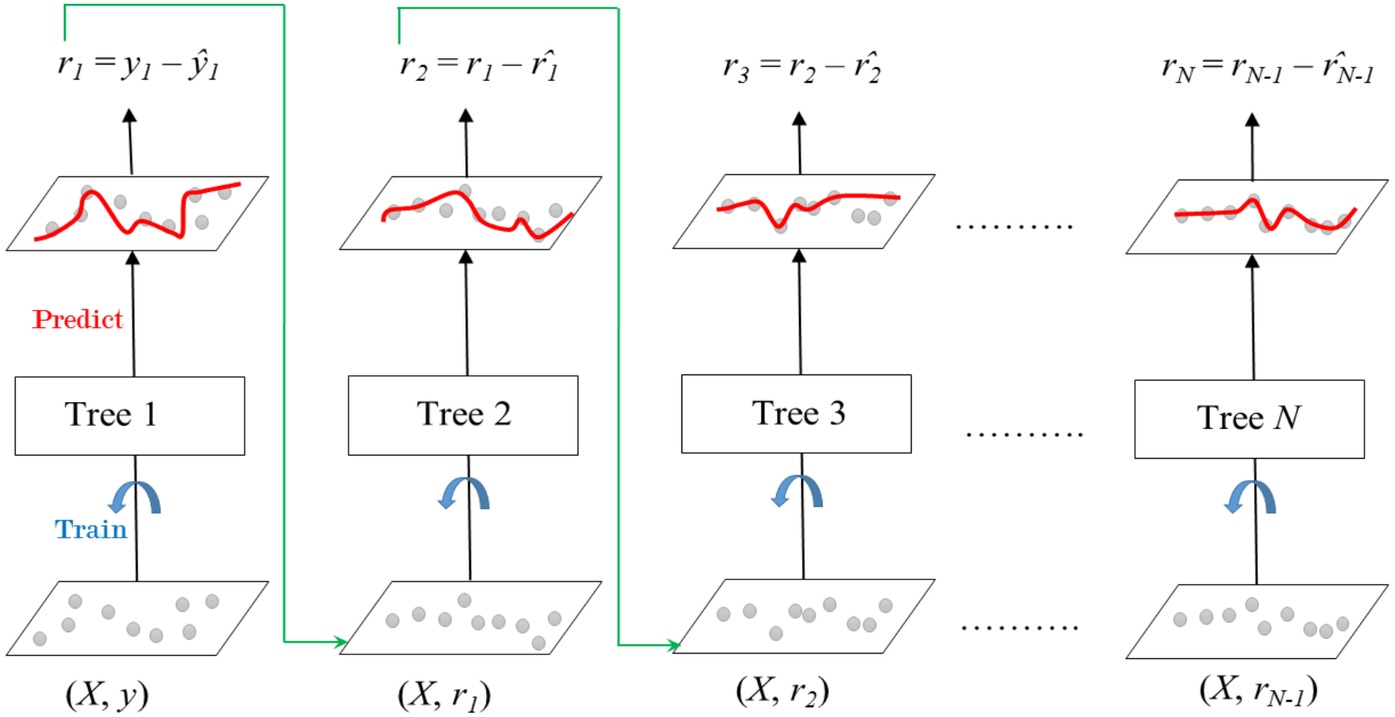
Gradient Boosted Trees

- Sequential correction of predecessor's errors.
- Does not tweak the weights of training instances.
- Fit each predictor is trained using its predecessor's residual errors as labels.
- Gradient Boosted Trees: a CART is used as a base learner.

Gradient Boosted Trees (GBT) is an ensemble learning technique that builds a predictive model by sequentially combining multiple decision trees. Unlike Random Forests, which train trees independently, Gradient Boosting focuses on **minimizing the residual errors** of the previous trees by building new trees that predict the gradient of a loss function.

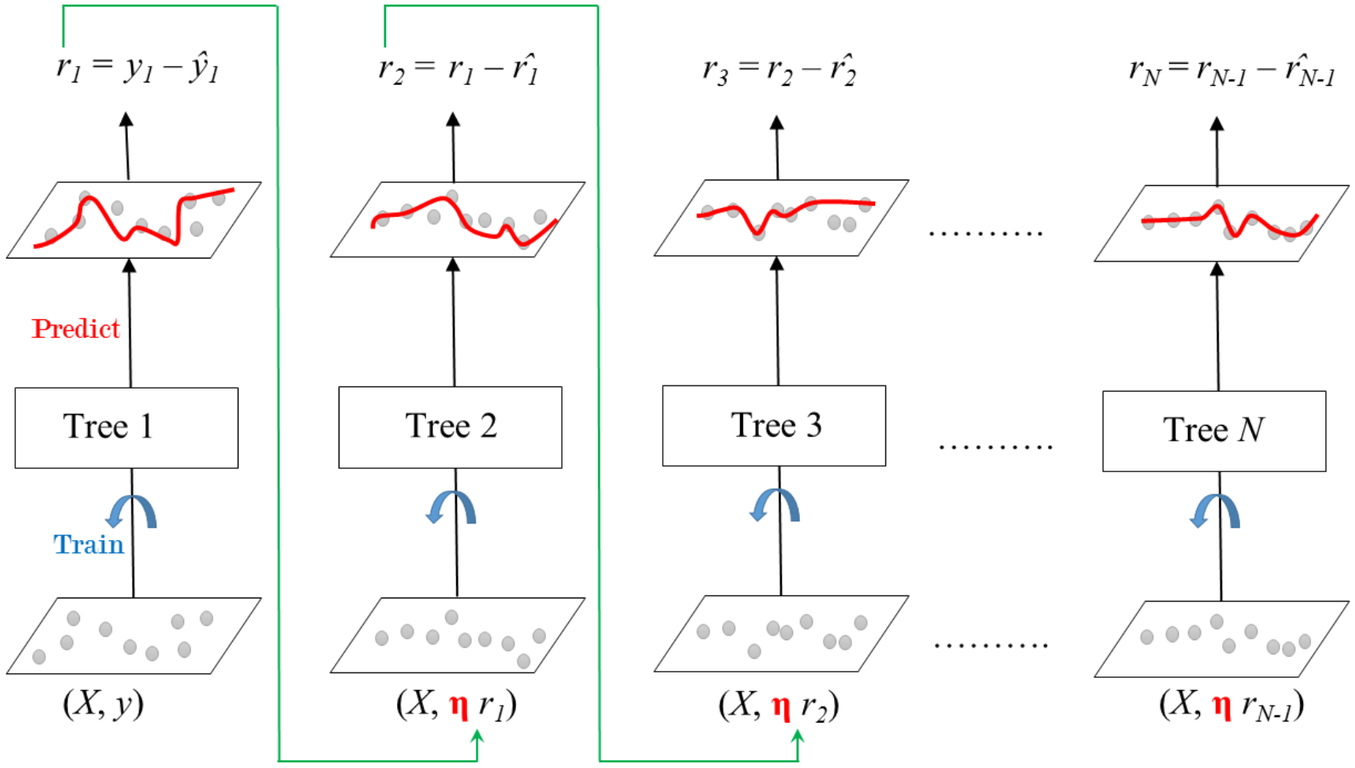
AdaBoost	Gradient Boosted Trees
Focuses on **reweighting samples**. Misclassified samples get higher weights so subsequent weak learners focus more on them.	Focuses on **minimizing a loss function** by fitting new trees to the residual errors of the previous model.
Error-driven: Learners are trained sequentially to correct misclassifications of prior learners.	Gradient-driven: Learners are trained to minimize the gradient of the loss function.
Typically uses **decision stumps** (one-level trees) as weak learners.	Uses **decision trees**, often shallow but deeper than stumps, as weak learners.
Each weak learner has a weight (α) based on its performance.	Trees are directly scaled by a **learning rate** and optimized based on residuals.
Adjusts **sample weights** dynamically. Misclassified samples are given more importance in subsequent rounds.	Focuses on reducing **residual errors** (difference between actual and predicted values) by fitting new trees.
Errors are handled at the **sample level**.	Errors are handled at the **prediction level** by minimizing the loss function.

Gradient Boosted Trees for Regression: Training



Step 1: Initialize the Model
Start by predicting a constant value (e.g., the mean of y) for all samples; Compute the initial residuals (r_1); These residuals serve as the target for the first tree
Step 2: Train the First Tree
Train **Tree 1** using the residuals r_1) as the target variable; The tree learns patterns in the features (X) that explain the residual errors from the initial model.
Step 3: Update Predictions
Use the predictions of **Tree 1** to update the overall predictions
Step 4: Compute New Residuals
Compute new residuals (r_2) based on the updated predictions
Step 5: Train the Next Tree
Train **Tree 2** to predict the updated residuals (r_2); Update the overall predictions using **Tree 2's** predictions
Step 6: Repeat
Repeat the process for a predefined number of iterations (N) or until the residual errors are minimized.
Step 7: Final Prediction
The final prediction combines the contributions of all trees

Shrinkage



The figure illustrates the concept of **shrinkage** in the context of **Gradient Boosted Trees (GBT)**. Shrinkage is implemented through the use of a **learning rate (η\etaη)**, which controls the contribution of each tree to the final ensemble. This mechanism ensures more gradual learning, improving the model's generalization and robustness.

Tree 1:
The first tree is trained to predict the initial residuals (r_1); Its predictions are scaled by η, and the overall predictions are updated.
Tree 2:
The second tree is trained on the updated residuals (r_2); Again, the predictions of Tree 2 are scaled by η, and the overall predictions are updated
Subsequent Trees:
Each subsequent tree learns from the scaled residuals and contributes a weighted correction to the overall prediction.
Final Prediction:
The final ensemble model combines the predictions of all trees, each scaled by η

Gradient Boosted Trees: Prediction

- Regression:
  - $y_{pred} = y_1 + \eta r_1 + \ldots + \eta r_N$
  - In sklearn: `GradientBoostingRegressor`.
- Classification:
  - In sklearn: `GradientBoostingClassifier`.

Gradient Boosting in sklearn (auto dataset)

```
# Import models and utility functions
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error as MSE

# Set seed for reproducibility
SEED = 1

# Split dataset into 70% train and 30% test
X_train, X_test, y_train, y_test = train_test_split(X,y,
                                                    test_size=0.3,
                                                    random_state=SEED)
```

```
# Instantiate a GradientBoostingRegressor 'gbt'
gbt = GradientBoostingRegressor(n_estimators=300, max_depth=1, random_state=SEED)

# Fit 'gbt' to the training set
gbt.fit(X_train, y_train)

# Predict the test set labels
y_pred = gbt.predict(X_test)

# Evaluate the test set RMSE
rmse_test = MSE(y_test, y_pred)**(1/2)

# Print the test set RMSE
print('Test set RMSE: {:.2f}'.format(rmse_test))
```

```
Test set RMSE: 4.01
```

n_estimators=300:Specifies the **number of decision trees** (or weak learners) in the ensemble. A higher number of estimators can improve accuracy but also increases computational cost and the risk of overfitting. The choice of this value often depends on cross-validation.

Lower RMSE values indicate better model performance, as RMSE measures the magnitude of the prediction errors (the differences between the actual and predicted values)

Stochastic Gradient Boosting (SGB)

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Stochastic Gradient Boosting (SGB) is an ensemble learning method that combines the principles of **Gradient Boosted Trees (GBT)** and **random sampling** (stochasticity). While Gradient Boosted Trees optimize a loss function by sequentially adding decision trees, SGB enhances this process by training each tree on a **random subset** of the data.

The introduction of stochasticity makes SGB more robust to overfitting and allows it to perform better on noisy datasets or datasets with many features.

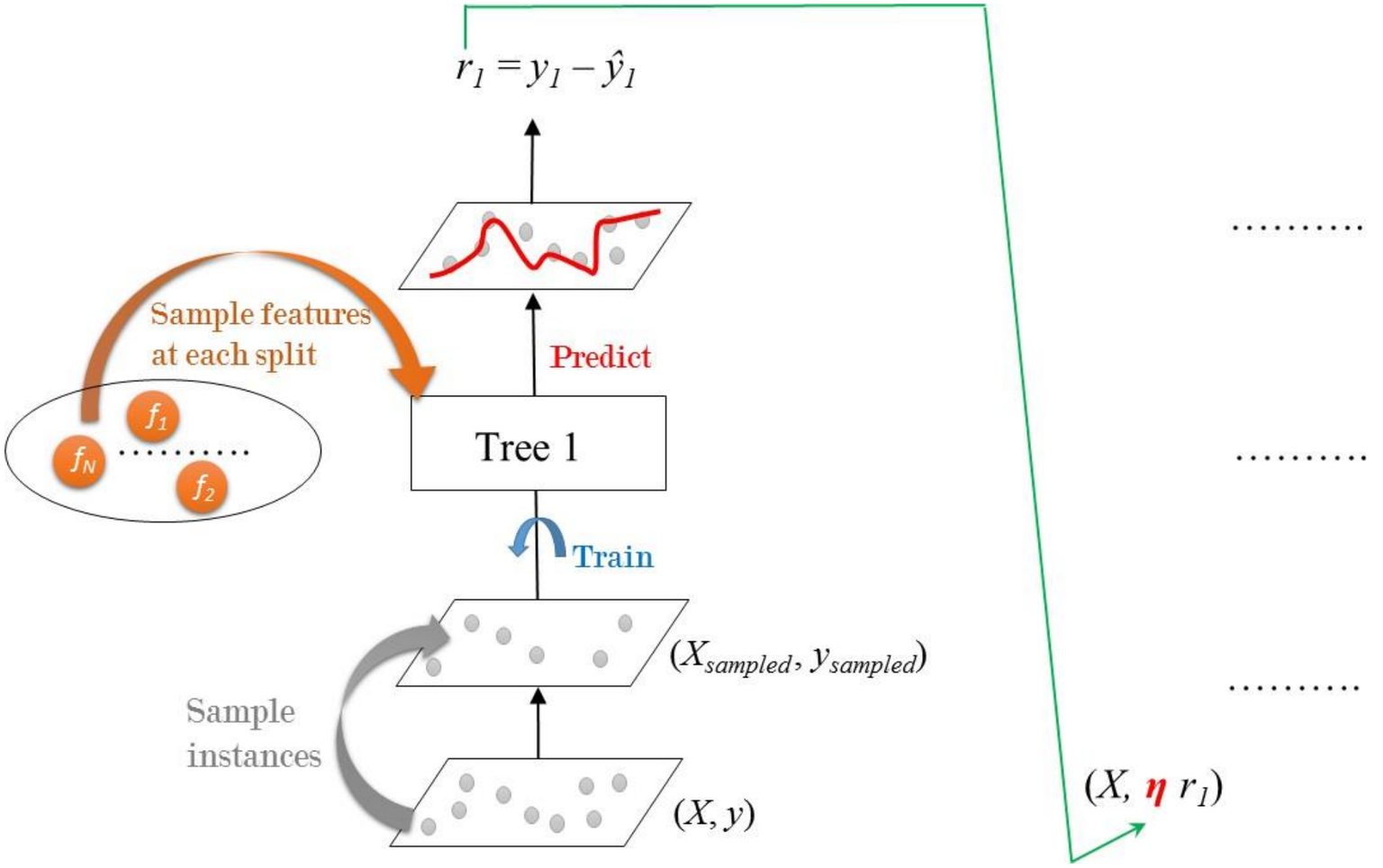
Gradient Boosting: Cons

- GB involves an exhaustive search procedure.
- Each CART is trained to find the best split points and features.
- May lead to CARTs using the same split points and maybe the same features.

Stochastic Gradient Boosting

- Each tree is trained on a random subset of rows of the training data.
- The sampled instances (40%-80% of the training set) are sampled without replacement.
- Features are sampled (without replacement) when choosing split points.
- Result: further ensemble diversity.
- Effect: adding further variance to the ensemble of trees.

Stochastic Gradient Boosting: Training



The figure provides a visual explanation of the **Stochastic Gradient Boosting (SGB)** training process. It highlights how randomness (stochasticity) is introduced at two levels: by **subsampling instances** and by **sampling features at each split**, making SGB more robust and generalizable.

Stochastic Gradient Boosting in sklearn (auto dataset)

```
# Import models and utility functions
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error as MSE

# Set seed for reproducibility
SEED = 1

# Split dataset into 70% train and 30% test
X_train, X_test, y_train, y_test = train_test_split(X,y,
                                                    test_size=0.3,
                                                    random_state=SEED)
```

Stochastic Gradient Boosting in sklearn (auto dataset)

```
# Instantiate a stochastic GradientBoostingRegressor 'sgbt'
sgbt = GradientBoostingRegressor(max_depth=1,
                                 subsample=0.8,
                                 max_features=0.2,
                                 n_estimators=300,
                                 random_state=SEED)


# Fit 'sgbt' to the training set
sgbt.fit(X_train, y_train)


# Predict the test set labels
y_pred = sgbt.predict(X_test)
```

subsample=0.8:
Indicates that **80% of the training data** is randomly sampled (without replacement) to train each tree.
Subsampling introduces randomness at the data level, which:

- Reduces overfitting.
- Encourages diversity among the trees in the ensemble.

max_features=0.2:
Specifies that only **20% of the features** are randomly selected at each split within a tree.
This feature sampling introduces randomness at the feature level, further reducing overfitting and decorrelating the trees.
For example, if there are 10 features in the dataset, only 2 features will be considered for each split.

Stochastic Gradient Boosting in sklearn (auto dataset)

```
# Evaluate test set RMSE 'rmse_test'
rmse_test = MSE(y_test, y_pred)**(1/2)


# Print 'rmse_test'
print('Test set RMSE: {:.2f}'.format(rmse_test))
```

```
Test set RMSE: 3.95
```