

Ensemble Learning in Machine Learning

What is Ensemble Learning?

- **Ensemble learning** is a machine learning technique that combines multiple individual models (called base learners or weak learners) to create a more robust and accurate predictive model.
- The idea is that while a single model might struggle with certain aspects of the data, combining the strengths of several models can improve overall performance and reduce errors.
- Ensemble methods are widely used in both classification and regression tasks
- Ensemble methods especially effective in overcoming challenges like overfitting, underfitting, and noise in the data.

Ensemble Learning

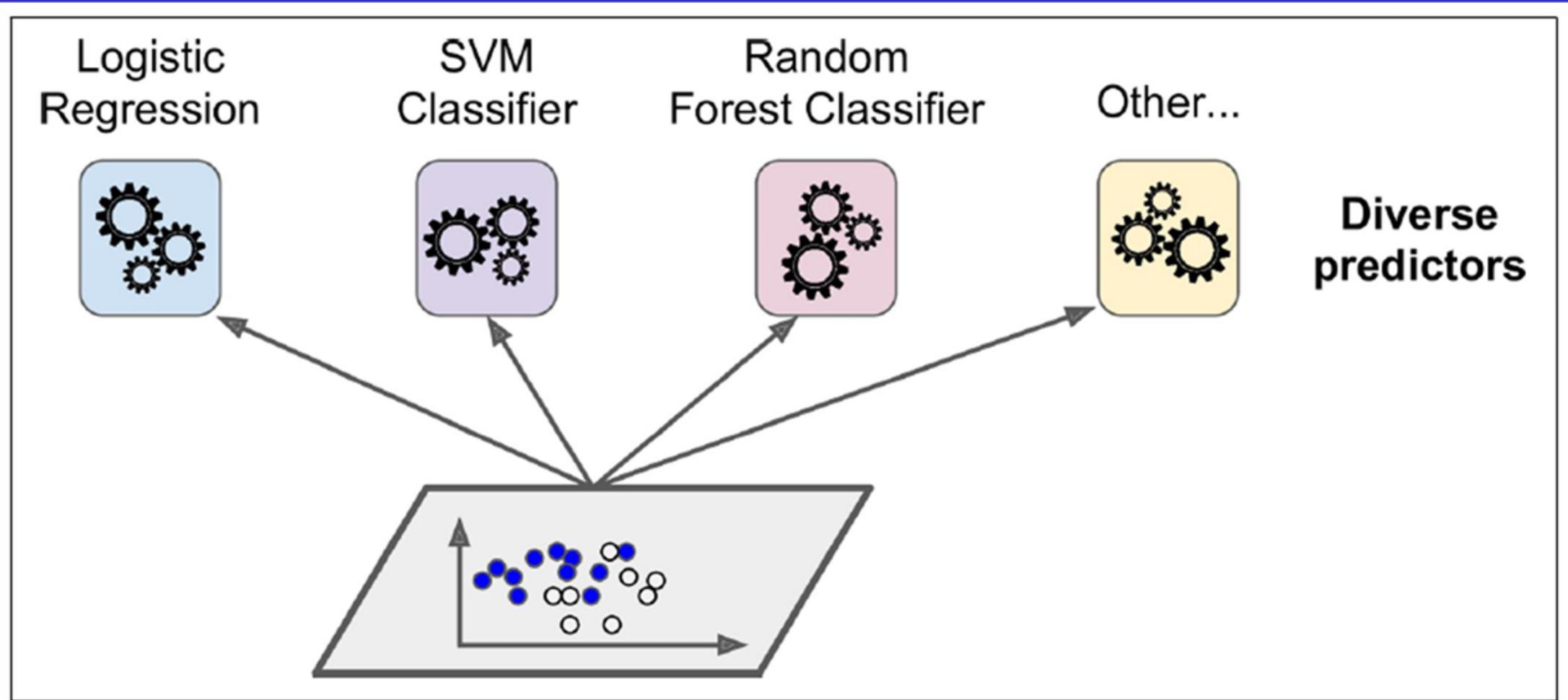


Figure 7-1. Training diverse classifiers

Why Use Ensemble Learning?

1. Improved Accuracy:

- Combining multiple models reduces the likelihood of errors that a single model might make.

2. Reduced Variance and Bias:

- By aggregating predictions, ensembles can address the issues of high variance (overfitting) or high bias (underfitting) in individual models.

3. Robustness:

- Ensembles are less sensitive to outliers or noise in the data compared to individual models.

4. Flexibility:

- Works with various types of base models, allowing for diverse approaches to solving a problem.

Key Concepts in Ensemble Learning

1. Base Learners (Weak Learners):

- These are the individual models that make up the ensemble.
- Examples: Decision Trees, Logistic Regression, SVM, Neural Networks, etc.

2. Diversity:

- The strength of ensemble methods comes from combining diverse models. If all models make the same errors, the ensemble won't perform better than an individual model.
- Diversity can be achieved by:
 - Using different algorithms (e.g., combining decision trees, SVMs, and logistic regression).
 - Training models on different subsets of data.
 - Using different hyperparameter configurations.

3. Aggregation:

- Predictions from base learners are aggregated to produce the final result.
- Techniques include:
 - **Voting** (for classification): Hard voting or soft voting.
 - **Averaging** (for regression): Simple average or weighted average.
 - **Stacking**: Combining predictions using a meta-model.

Types of Ensemble Learning Methods

1. Bagging (Bootstrap Aggregating):

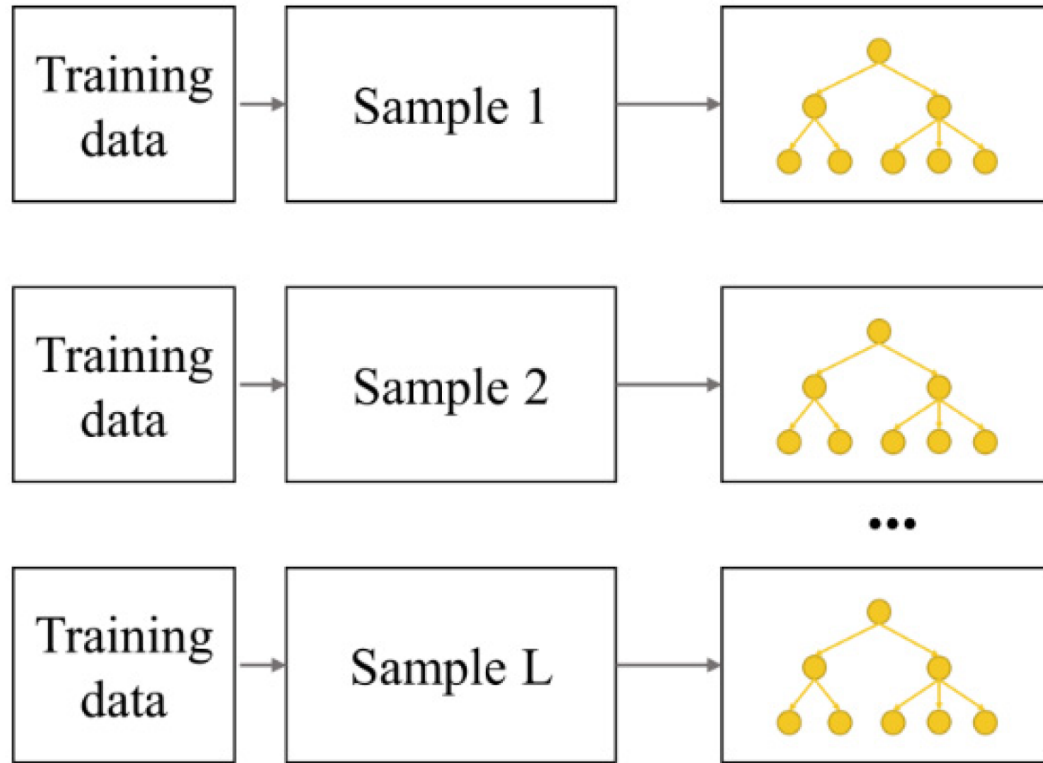
- Reduces variance by training multiple models on different random subsets (with replacement) of the data and averaging their predictions.
- Example: **Random Forest**.
- Key feature: Models are trained independently.

2. Boosting:

- Reduces bias by sequentially training models, where each new model focuses on the errors made by the previous models.
- Example: **Gradient Boosting, AdaBoost, XGBoost, LightGBM, CatBoost**.
- Key feature: Models are trained sequentially, and weights are adjusted iteratively.



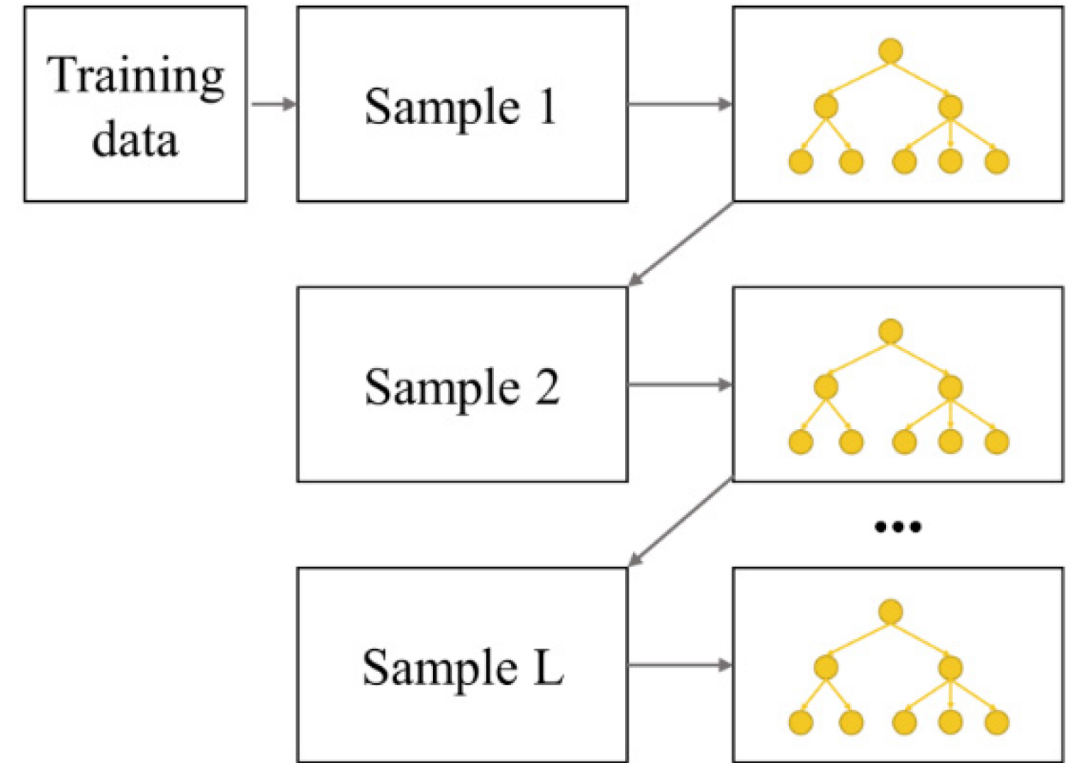
Bagging



Independent weak learning training

VS

Boosting



Iterative weak learner training

3. Stacking (Stacked Generalization):

- Combines predictions from multiple models using a meta-model (a higher-level model) to make the final prediction.
- Example: Using Logistic Regression to combine predictions from a Random Forest and an SVM.

Advantages of Ensemble Learning

- **Higher accuracy:** Often achieves better performance than individual models.
- **Flexibility:** Can combine simple and complex models.
- **Applicability:** Works for both supervised (classification, regression) and unsupervised tasks.

Disadvantages of Ensemble Learning

1. Complexity:

- Ensembles are harder to interpret than single models.
- Managing multiple models increases computational cost.

2. Training Time:

- Requires significant computational resources, especially for methods like Random Forest or Boosting with a large number of models.

3. Overfitting:

- Ensembles like boosting may overfit if the base learners are too complex or the ensemble is too large.

Real-World Applications of Ensemble Learning

1. **Finance:** Credit scoring, fraud detection.
2. **Healthcare:** Disease prediction, drug discovery.
3. **Marketing:** Customer segmentation, recommendation systems.
4. **Technology:** Spam filters, search engine rankings.

Hard Voting

- **Definition:** In hard voting, the final prediction is based on the **majority class** predicted by individual classifiers. Each classifier casts a "vote" for a specific class, and the class with the most votes is chosen as the ensemble's output.
- **How it works:**
 1. Each classifier makes a discrete class prediction (e.g., 0, 1, or 2).
 2. The predictions are aggregated, and the class with the highest count of votes is selected.
- **Use case:** Hard voting is useful when you only need the most commonly predicted class and do not have access to prediction probabilities.

- **Example:** If three classifiers predict the following classes:

```
sql
```

```
Classifier 1: 0
```

```
Classifier 2: 1
```

```
Classifier 3: 1
```

The final prediction using hard voting would be **1** because it has the majority of votes.

Voting Classifiers

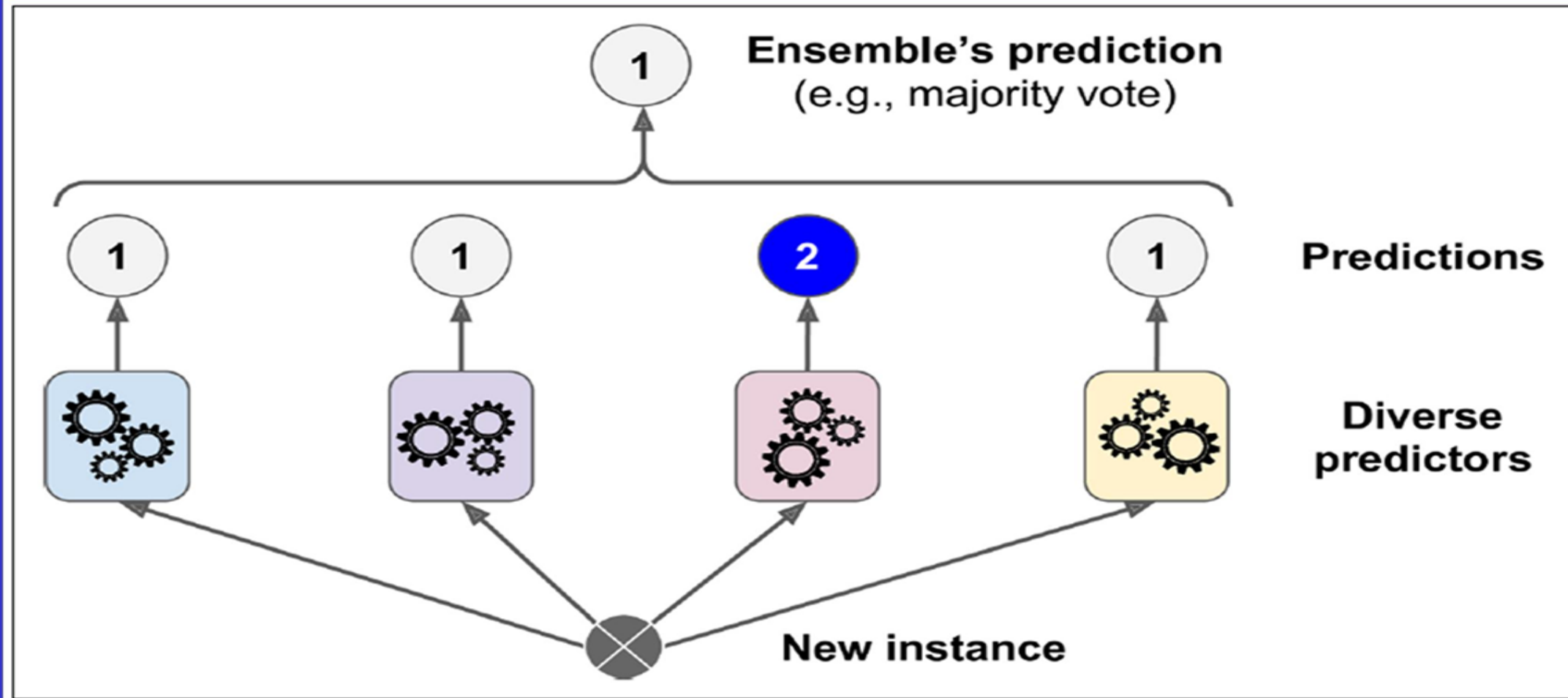


Figure 7-2. Hard voting classifier predictions

Soft Voting

- **Definition:** In soft voting, the final prediction is based on the **average predicted probabilities** from all classifiers. The class with the highest average probability is chosen as the ensemble's output.
- **How it works:**
 1. Each classifier outputs a probability distribution over the classes (e.g., `[0.2, 0.8]` for binary classification).
 2. The probabilities are averaged across all classifiers for each class.
 3. The class with the highest average probability is selected.
- **Use case:** Soft voting is more powerful when classifiers provide reliable probability estimates, as it takes into account the confidence of predictions.

- **Example:** If three classifiers provide the following probability distributions:

Classifier 1: [0.7, 0.3] (70% for class 0, 30% for class 1)

Classifier 2: [0.4, 0.6] (40% for class 0, 60% for class 1)

Classifier 3: [0.2, 0.8] (20% for class 0, 80% for class 1)

Average probabilities:

vbnet

Class 0: $(0.7 + 0.4 + 0.2) / 3 = 0.433$

Class 1: $(0.3 + 0.6 + 0.8) / 3 = 0.566$

The final prediction using soft voting would be **1**, as it has the higher average probability.

Key Differences

Aspect	Hard Voting	Soft Voting
Basis	Majority class votes	Average predicted probabilities
Input	Class predictions	Probability estimates
When to use	When probabilities are unavailable	When probability estimates are reliable
Strength	Simpler, less sensitive to outliers	Takes confidence of classifiers into account

Ensemble Learning

- Ensemble methods work best when the predictors are as independent from one another as possible.
- One way to get diverse classifiers is to train them using ***very different algorithms***.
- This increases the chance that they will make very different types of errors, improving the ensemble's accuracy.

Voting Classifiers in sklearn

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

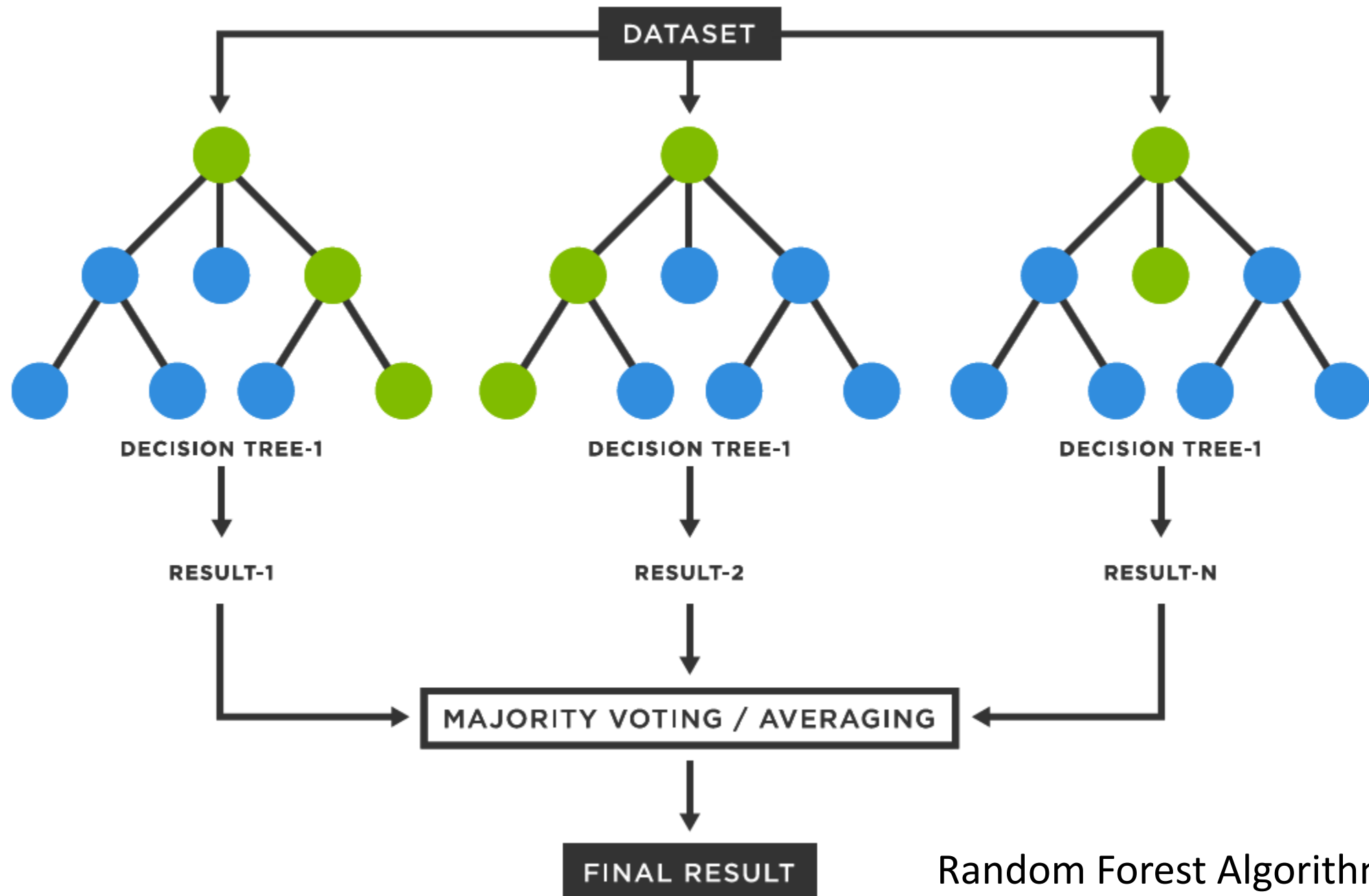
log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC()

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='hard')
voting_clf.fit(X_train, y_train)
```

Voting Classifiers in sklearn

```
>>> from sklearn.metrics import accuracy_score
>>> for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
...     clf.fit(X_train, y_train)
...     y_pred = clf.predict(X_test)
...     print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
...
LogisticRegression 0.864
RandomForestClassifier 0.896
SVC 0.888
VotingClassifier 0.904
```

- There you have it! --- on the moons dataset
- The voting classifier slightly outperforms all the individual classifiers.



Random Forest Algorithm

Key Characteristics of Extra-Trees:

1. Randomness in Split Selection:

- In Random Forest, the best split at each node is selected based on a subset of features.
- In Extra-Trees, the splits are chosen completely at random from the feature space and a random threshold is selected for splitting, rather than optimizing for the best threshold.

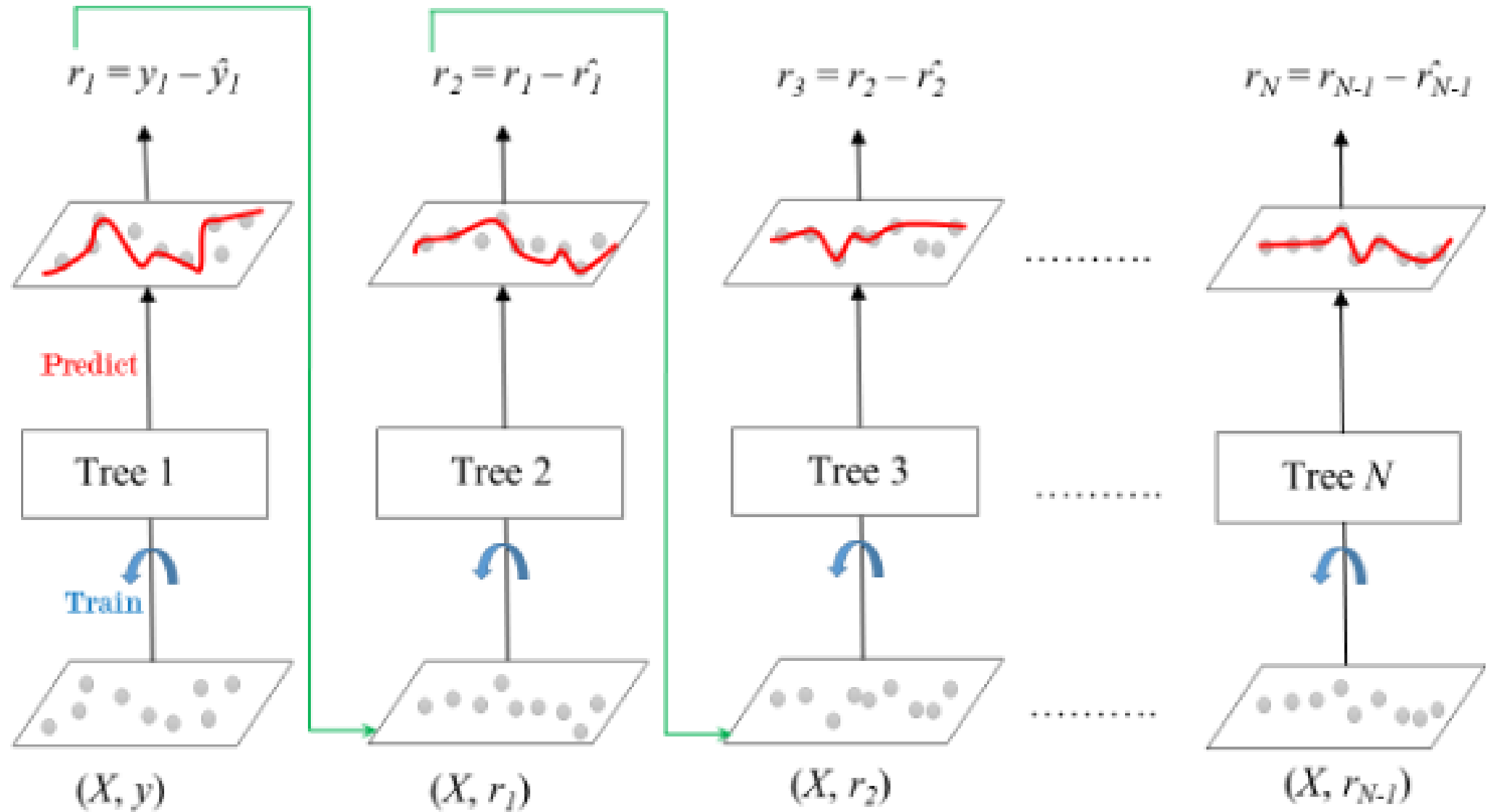
2. Bootstrap Sampling:


- While Random Forest uses bootstrapping (sampling with replacement), Extra-Trees often uses the entire dataset for training each tree (though it can also use bootstrapping if desired).

Hyperparameters in Random Forest

Key parameters you can tune to improve performance:

1. `n_estimators` : Number of trees in the forest.
2. `max_features` : Maximum number of features considered at each split.
3. `max_depth` : Maximum depth of the trees.
4. `min_samples_split` : Minimum samples required to split a node.
5. `min_samples_leaf` : Minimum samples required to be a leaf node.



- **AdaBoost:**
 - Focuses on adjusting the weights of the training samples.
 - After each iteration, misclassified samples get higher weights so that the next weak learner pays more attention to them.
 - Combines models sequentially by assigning a weight to each weak learner based on its accuracy.
- **Gradient Boosting:**
 - Uses gradient descent to minimize a loss function.
 - At each step, it fits a new model to the **residual errors** (the difference between predicted and actual values) of the current ensemble.
 - Learners are trained to predict  the residuals rather than directly adjusting sample weights.

2. Error Handling

- AdaBoost:
 - Emphasizes correcting misclassified examples by increasing their importance.
 - Targets **classification error** directly (e.g., misclassified versus correctly classified).
- Gradient Boosting:
 - Minimizes a specific **loss function** (e.g., mean squared error for regression or log-loss for classification).
 - Focuses on optimizing the loss function rather than explicitly correcting misclassifications.



3. Weak Learner Updates

- **AdaBoost:**
 - Each weak learner is trained independently, with the dataset re-weighted after every iteration.
 - The final model combines all learners with weighted voting (classification) or a weighted sum (regression).
- **Gradient Boosting:**
 - Each weak learner corrects the residual errors made by the previous ensemble.
 - The final model combines learners using additive predictions.

4. Handling Outliers

- **AdaBoost:**
 - Sensitive to outliers because it keeps increasing their weights in subsequent iterations.
 - This can cause overfitting if the data contains noisy labels.
- **Gradient Boosting:**
 - Less sensitive to outliers because it optimizes the loss function and does not solely focus on individual instances.

5. Computation and Complexity

- **AdaBoost:**
 - Simpler and faster to implement as it primarily involves re-weighting the training data.
 - Typically works well with shallow decision trees (stumps).
- **Gradient Boosting:**
 - Computationally more intensive as it involves optimizing the loss function at each iteration using gradient descent.
 - Usually employs deeper trees, which require more resources.

6. Use Cases

- **AdaBoost:**
 - Works well for simple datasets and problems with fewer features.
 - Popular in classification tasks.
- **Gradient Boosting:**
 - Excels in handling complex datasets and problems.
 - Commonly used in regression and classification tasks, especially with advanced implementations like XGBoost, LightGBM, or CatBoost.

Aspect	AdaBoost	Gradient Boosting
Core Idea	Adjust sample weights	Minimize loss function via residuals
Focus	Misclassified samples	Residual errors
Outlier Sensitivity	High	Lower
Computation	Faster	Slower
Use Case	Simpler problems	Complex problems