**Ensemble Learning**

**What is Ensemble Learning:**

Ensemble learning is a method where we use many small models instead of just one. Each of these models may not be very strong on its own, but when we put their results together, we get a better and more accurate answer. It's like asking a group of people for advice instead of just one person—each one might be a little wrong, but together, they usually give a better answer.

**Why use ensemble learning?**

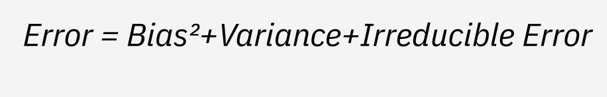
**Bias-variance tradeoff**

Bias-variance tradeoff is a well-known problem in machine learning and a motivating principle behind many [regularization](https://www.ibm.com/topics/regularization) techniques. We can define them as:

- **Bias** measures the average difference between predicted values and true values. As bias increases, a model predicts less accurately on a training dataset. High bias refers to high error in training. Optimization signifies attempts to reduce bias.

- **Variance** measures the difference between predictions across various realizations of a given model. As variance increases, a model predicts less accurately on unseen data. High variance refers to high error during testing and validation. Generalization refers to attempts to reduce variance.

Bias and variance thus inversely represent model accuracy on training and test data respectively.5 They are two of three terms that comprise a model’s total error rate, the third being irreducible error. This third term denotes error resulting from inherent randomness in a dataset. Total model error can be defined by the formula:6



**Many models versus one**

Any one model training algorithm consists of numerous variables—e.g. training data, hyperparameters, and so forth—that affect the consequent model’s total error. Thus, even a single training algorithm can produce different models, each with their own bias, variance, and irreducible error rates. By combining several diverse models, ensemble algorithms can yield a lower overall error rate while retaining each individual model’s own complexities and advantages, such as notably low bias for a specific data subset.7

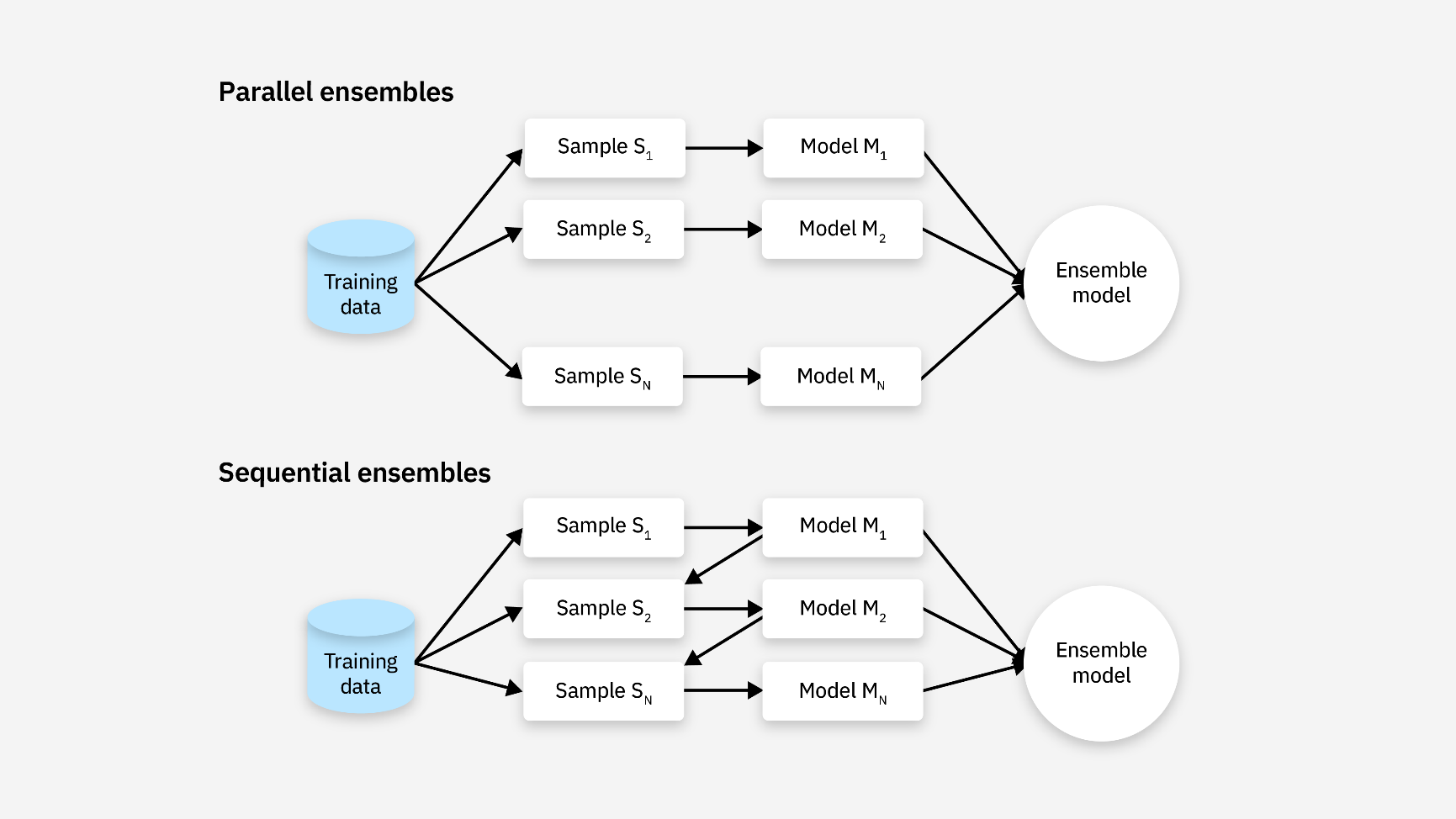
Research suggests that, in general, the greater diversity among combined models, the more accurate the resulting ensemble model. Ensemble learning can thus address regression problems such as [overfitting](https://www.ibm.com/topics/overfitting) without trading away model bias. Indeed, research suggests that ensembles comprised of diverse under-regularized models (i.e. models that overfit to their training data) outperform single regularized models.8 Moreover, ensemble learning techniques can help resolve issues stemming from high-dimensional data, and so effectively serve as an alternative to [dimensionality reduction](https://www.ibm.com/topics/dimensionality-reduction).

Types of ensemble models

Literature widely categorizes ensemble learning methods in machine learning into two groups: parallel and sequential.

-**Parallel**methods train each base learner apart from the others of the others. Per its name, then, parallel ensembles train base learners in parallel and independent of one another.

- **Sequential** methods train a new base learner so that it minimizes errors made by the previous model trained in the preceding step. In other words, sequential methods construct base models sequentially in stages.9



Parallel methods are further divided into homogenous and heterogenous methods. Homogenous parallel ensembles use the same base learning algorithm to produce all of the component base learners. Heterogenous parallel ensembles use different algorithms to produce base learners.10

**Voting**

How do ensemble methods combine base learners into a final learner? Some techniques—e.g. stacking—use separate machine learning algorithms to train an ensemble learner from the base learners. But one common method for consolidating base learner predictions is voting—and more precisely, majority voting.

Majority voting considers each base learner’s prediction for a given data instance and outputs a final prediction determined by whatever the majority of learners predict. For instance, in a binary classification problem, majority voting takes predictions from each base classifier for a given data instance and uses the majority prediction as the end prediction. Weighted majority voting is an extension of this technique that gives greater weight to certain learner’s predictions over others.11

Ensemble learning techniques

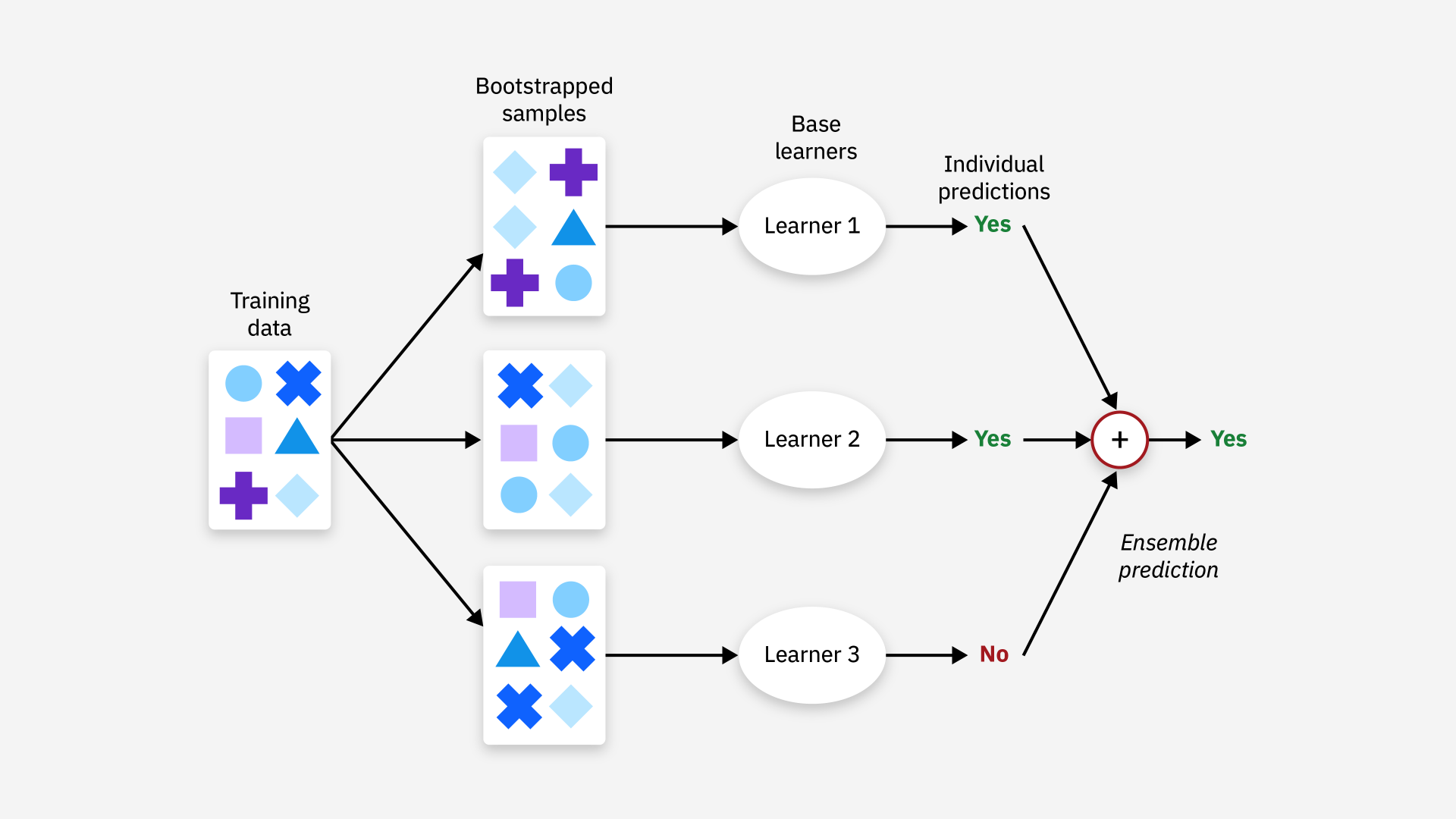
Perhaps three of the most popular ensemble learning techniques are bagging, boosting, and stacking. In fact, these together exemplify distinctions between sequential, parallel, homogenous, and heterogenous types of ensemble methods.

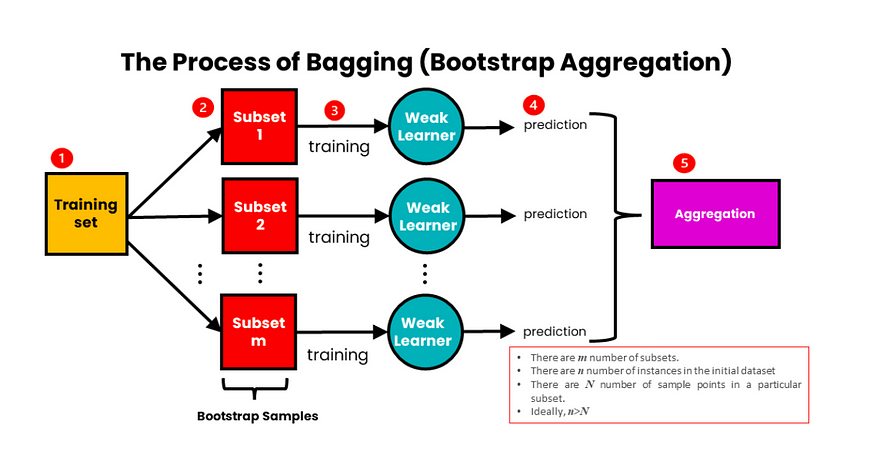
Note that this overview is not exhaustive; there are several additional ensemble methods, such as blending and weighted average ensembles. This is merely meant to survey some of the more prominent methods in literature.

**Bagging**

[Bagging](https://www.ibm.com/topics/bagging) is a homogenous parallel method sometimes called *bootstrap aggregating*. It uses modified replicates of a given training data set to train multiple base learners with the same training algorithm.12 Scikit-learn’s ensemble module in Python contains functions for implementing bagging, such as BaggingClassifier.

More specifically, bagging uses a technique called bootstrap resampling to derive multiple new datasets from one initial training dataset in order to train multiple base learners. How does this work? Say a training dataset contains *n*training examples. Bootstrap resampling copies*n* data instances from that set into a new subsample dataset, with some initial instances appearing more than once and others excluded entirely. These are bootstrap samples. Repeating this process *x* times produces *x* iterations of the original dataset, each containing *n*samples from the initial set. Each iteration of the initial set is then used to train a separate base learner with the same learning algorithm.13





Steps:

These are the steps involved in bagging:  
  
An initial training dataset with n instances is available to us.  
We take the training set and divide it into m-number subgroups.

For every subset, we select a subset of N sample points from the original dataset.

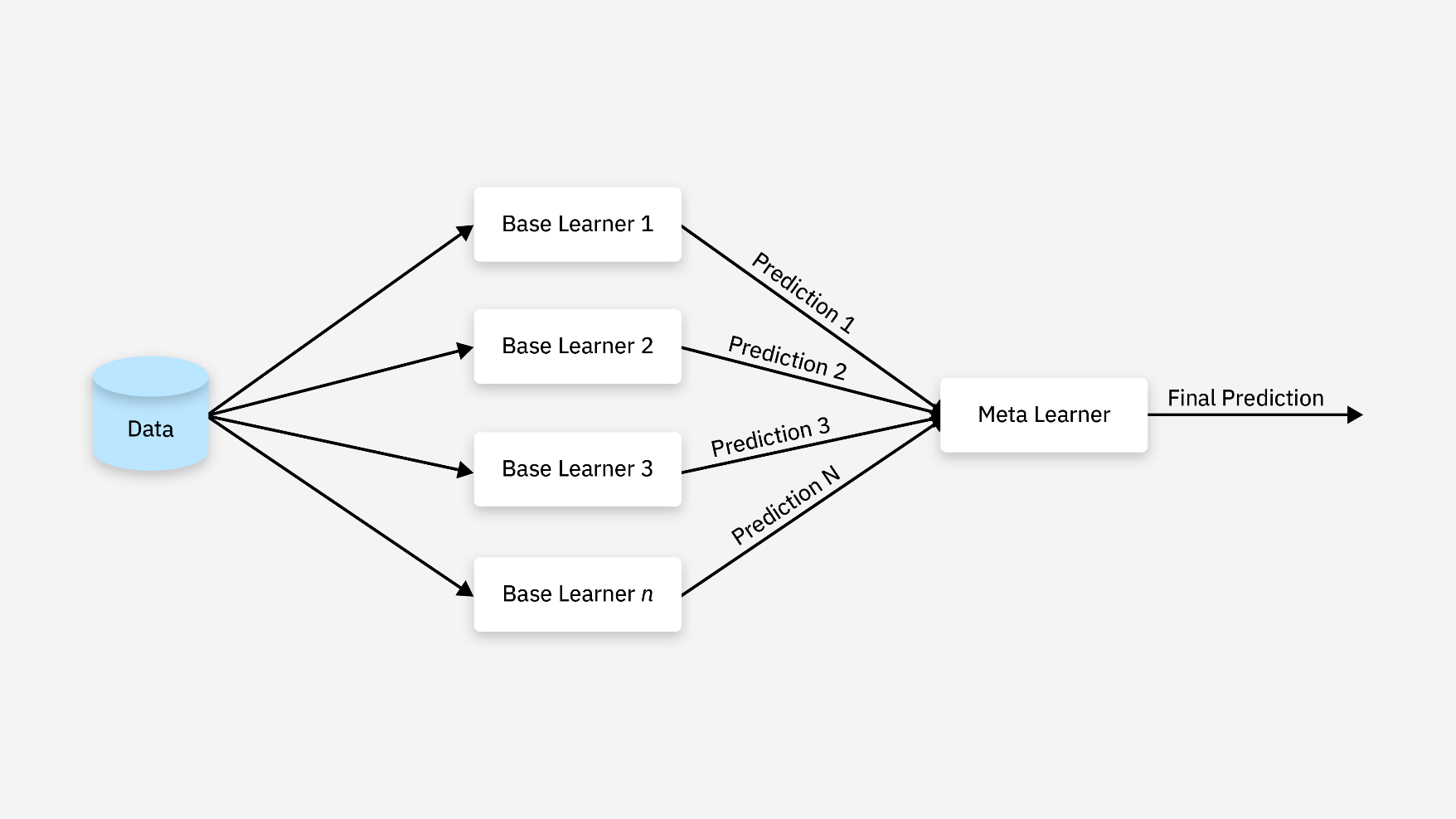
We take each subset and replace it. This implies that multiple samples can be taken from a single data point.

We train the associated weak learners independently for each subgroup of the data. Since these models are all of the same type, they are homogeneous.  
Every model projects a future state.  
One forecast is created by combining all of the predictions. Either maximum voting or average are applied in this case.

Random forest is an extension of bagging that specifically denotes the use of bagging to construct ensembles of randomized [decision trees](https://www.ibm.com/topics/decision-trees). This differs from standard decision trees in that the latter samples every feature to identify the best for splitting. By contrast, random forests iteratively sample random subsets of features to create a decision node.14

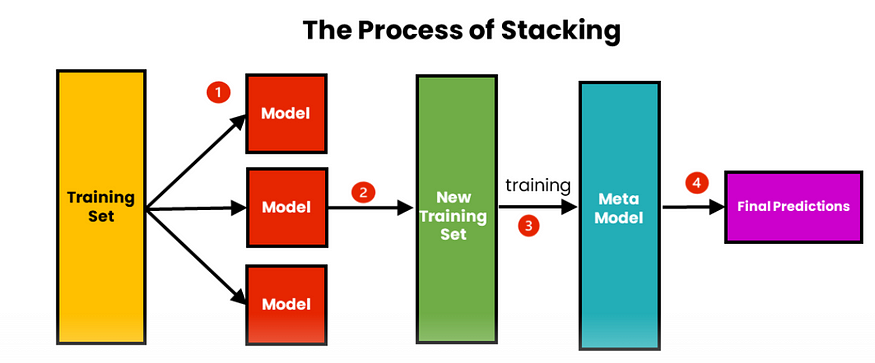
**Stacking**

Stacking, or stacked generalization,15 is a heterogenous parallel method that exemplifies what is known as meta-learning. Meta-learning consists of training a meta-learner from the output of multiple base learners. Stacking specifically trains several base learners from the same dataset using a different training algorithm for each learner. Each base learner makes predictions on an unseen dataset. These first model predictions are then compiled and used to train a final model, being the meta-model.16



Note the importance of using a different dataset from that used to train the base learners in order to train the meta-learner. Using the same dataset to train the base learners and the meta-learner can result in overfitting. This can require excluding data instances from the base learner training data to serve as its test set data, which in turn becomes training data for the meta-learner. Literature often recommends techniques such as cross-validation to ensure these datasets do not overlap.17

Stacking, also known as stacked generalization, is the process of training several models on the same set of data, followed by the training of a meta-model to produce a final prediction that is based on the earlier models’ predictions.



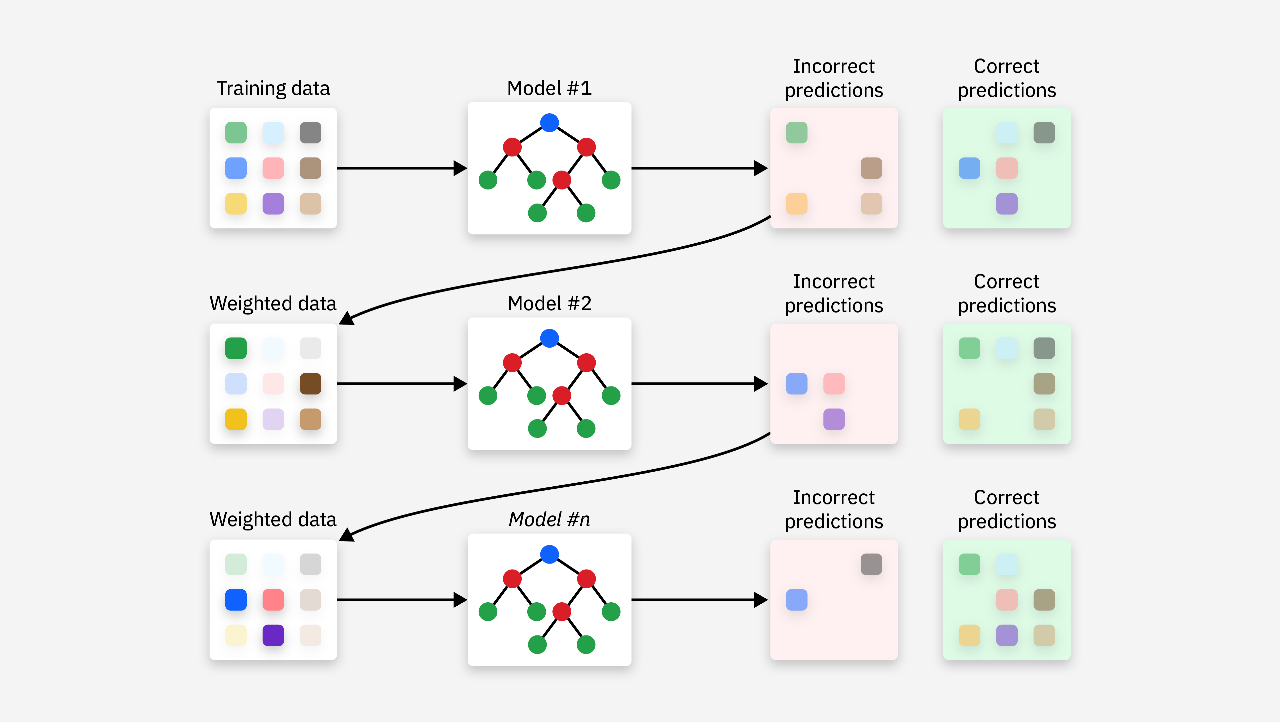
Here are the steps involved in stacking:  
  
We train m-number of algorithms using the original training set of data.  
Every algorithm’s output is used to generate a fresh training set.

We develop a meta-model algorithm with the fresh training set.  
Our ultimate forecast is based on the meta-model’s findings. Weighted averaging is utilized to aggregate the outcomes.

Much as bagging, the sklearn.ensemble module in Python provides various functions for implementing stacking techniques.

**Boosting**

[Boosting](https://www.ibm.com/topics/boosting) algorithms are a sequential ensemble method. Boosting has many variations, but they all follow the same general procedure. Boosting trains a learner on some initial dataset, *d*. The resultant learner is typically weak, misclassifying many samples in the dataset. Much like bagging, boosting then samples instances from the initial dataset to create a new dataset (*d2*). Unlike bagging, however, boosting prioritizes misclassified data instances from the first model or learner. A new learner is trained on this new dataset *d2*. Then a third dataset (*d3*) is then compiled from *d1* and *d2*, prioritizes the second learner’s misclassified samples and instances in which *d1* and *d2* disagree. The process repeats *n* times to produce *n* learners. Boosting then combines and weights the all the learners together to produce final predictions.18



Boosting algorithms largely differ in how they prioritize erroneously predicted data instances when creating a new dataset. Two of the most prominent boosting methods may illustrate this:

- **Adaptive boosting** (AdaBoost) weights model errors. That is, when creating a new iteration of a dataset for training the next learner, AdaBoost adds weights to the previous learner’s misclassified samples, causing the next learner to prioritize those misclassified samples.

- **Gradient boosting** uses residual errors when training new learners. Rather than weight misclassified samples, gradient boosting uses residual errors from a previous model to set target predictions for the next model. In this way, it attempts to close the gap of error left by one model.19

Unfortunately, sklearn contains no pre-defined functions for implementing boosting. The Extreme Gradient Boosting (XGBoost) open-source library, however, provides code for implementing gradient boosting in Python.

Read this also:

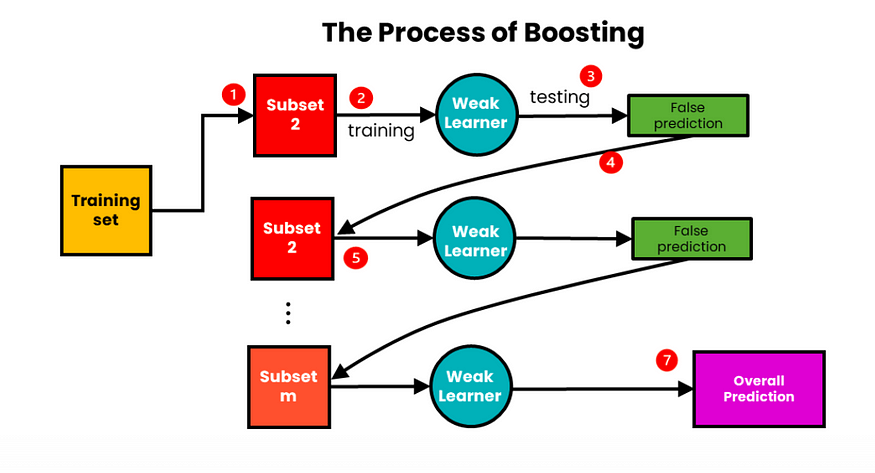
Boosting involves successively training models so that each one tries to fix the mistakes of the one before it. Predictions are derived from the weighted sum of the models, which are ranked according to how accurate they are. AdaBoost, gradient boosting, and XGBoost are a few examples.

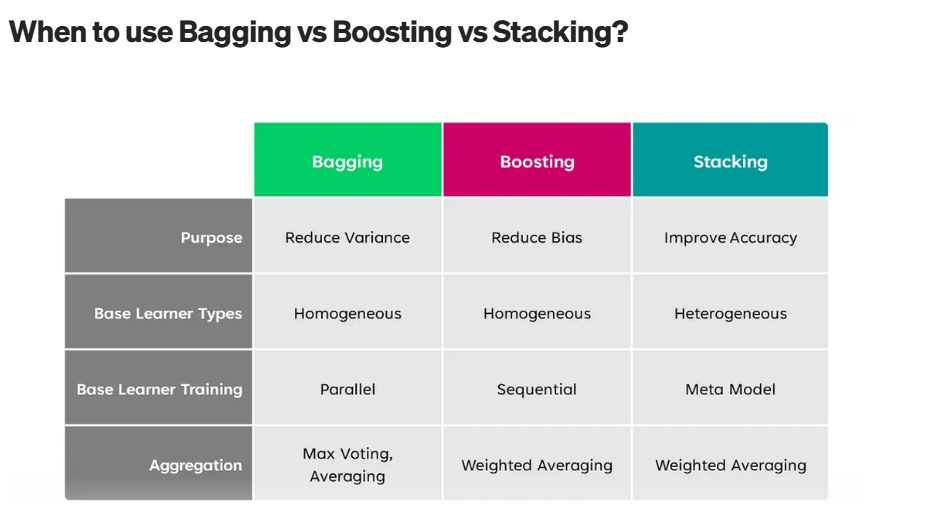
Boosting involves sequentially training weak learners. Here, each subsequent learner improves the errors of previous learners in the sequence. A sample of data is first taken from the initial [dataset](https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/).

This sample is used to train the first model, and the model makes its prediction. The samples can either be correctly or incorrectly predicted. The samples that are wrongly predicted are reused for training the next model. In this way, subsequent models can improve on the errors of previous models.

Unlike bagging, which aggregates prediction results at the end, boosting aggregates the results at each step. They are aggregated using weighted averaging.

Using weighted averaging, each model is assigned a different weight based on how well it predicts the future. Stated differently, it assigns greater weight to the model possessing the highest predictive power. This is due to the fact that the learner deemed most significant is the one with the most predictive power.



****

**Benefits of Ensemble Learning in Machine Learning**

Ensemble learning is a versatile approach that can be applied to machine learning model for: -

* **Reduction in Overfitting**: By aggregating predictions of multiple model's ensembles can reduce overfitting that individual complex models might exhibit.
* **Improved Generalization**: It generalizes better to unseen data by minimizing variance and bias.
* **Increased Accuracy**: Combining multiple models gives higher predictive accuracy.
* **Robustness to Noise**: It mitigates the effect of noisy or incorrect data points by averaging out predictions from diverse models.
* **Flexibility**: It can work with diverse models including decision trees, neural networks and support vector machines making them highly adaptable.
* **Bias-Variance Tradeoff**: Techniques like bagging reduce variance, while boosting reduces bias leading to better overall performance.

There are various ensemble learning techniques we can use as each one of them has their own pros and cons.

**Ensemble Learning Techniques**

| **Technique** | **Category** | **Description** |
| --- | --- | --- |
| **Random Forest** | Bagging | [Random forest](https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/) constructs multiple decision trees on bootstrapped subsets of the data and aggregates their predictions for final output, reducing overfitting and variance. |
| **Random Subspace Method** | Bagging | Trains models on random subsets of input features to enhance diversity and improve generalization while reducing overfitting. |
| **Gradient Boosting Machines (GBM)** | Boosting | [Gradient Boosting Machines](https://www.geeksforgeeks.org/ml-gradient-boosting/) sequentially builds decision trees, with each tree correcting errors of the previous ones, enhancing predictive accuracy iteratively. |
| **Extreme Gradient Boosting (XGBoost)** | Boosting | [XGBoost](https://www.geeksforgeeks.org/xgboost/) do optimizations like tree pruning, regularization, and parallel processing for robust and efficient predictive models. |
| **AdaBoost (Adaptive Boosting)** | Boosting | [AdaBoost](https://www.geeksforgeeks.org/implementing-the-adaboost-algorithm-from-scratch/) focuses on challenging examples by assigning weights to data points. Combines weak classifiers with weighted voting for final predictions. |
| **CatBoost** | Boosting | [CatBoost](https://www.geeksforgeeks.org/catboost-ml/) specialize in handling categorical features natively without extensive preprocessing with high predictive accuracy and automatic overfitting handling. |

**Simple Ensemble Techniques**

In this section, we will look at a few simple but powerful techniques, namely:

**1. Max Voting**

The maximum vote method is commonly used for classification problems. This technique uses multiple models to make predictions for each data point. Predictions from each model are considered “votes.” The predictions from most of the models are used as final predictions.

For example,

Let’s take an example where we have three classifiers with the following predictions:

Classifier 1 – Class B

Classifier 2 – Class B

Classifier 3 – Class A

The final prediction here will be class B with the most votes.

Such a method is suitable for binary classification problems where there are only two candidates that the classifier can vote for. However, many classes fail due to problems, as often, every class has a clear majority of votes.

**2. Averaging**

With averaging, the final output will be the average of all predictions. This applies to regression problems. For example, in random forest regression, the final result is the average of predictions from individual decision trees.

Let’s look at an example of three regression models that predict commodity prices as follows:

Regressor 1 – 500

Regressor 2 – 200

Regressor 3 – 100

The final prediction will be the average of 500, 200, and 100.

**3. Weighted Averaging**

A weighted average emphasizes the underlying model with high predictive power. In the price forecast example, each regressor is assigned a weight. Because the model weights are only small positive values ​​and the sum of all weights equals 1, the weights can indicate each model’s confidence or expected performance percentage.

Suppose the regressors are given weights of 0.35, 0.2, and 0.1, respectively. The final model prediction can be computed as

P=W1\*p1+ W2\*p2+ W3\*p3

Where W1, W2, W3 are the weights.

And p1,p2, and p3 are predictions by different models.

P=Final prediction

 0.35 \* 100 + 0.2 \* 200 + 0.1 \* 500 = 285.

**Row Sampling & Column Sampling (With or Without Replacement)**

1. **Row Sampling** (a.k.a. **Bootstrap Sampling**):
   * You select a subset of the **training data rows (samples)**.
   * **With Replacement**: A sample can be chosen more than once.
     + Often used in **Bagging**.
   * **Without Replacement**: A sample is chosen only once.
     + Used in **Pasting**.
2. **Column Sampling**:
   * You select a subset of **features (columns)** instead of or along with rows.
   * Helps in reducing overfitting and creating diverse models.

**🔹 Bagging (Bootstrap Aggregating)**

* **Bagging = Row Sampling with Replacement + Model Training**
* Each model is trained on a **different bootstrap sample** of the training data.
* Reduces **variance** and helps in avoiding overfitting.

**🔸 Used With:**

* **Classification**: Majority voting
* **Regression**: Averaging predictions

**Example:** Random Forest is an ensemble of Decision Trees trained with bagging.

**🔹 Pasting**

* Similar to Bagging, **but without replacement**.
* Still involves row sampling but ensures each data point is used only once per sample.
* Often less variance than Bagging but can have more bias.

**🔹 Random Subspaces**

* **Only features (columns) are randomly selected**, not rows.
* All models get the same data instances but different feature subsets.
* Useful when the feature space is large.

**🔹 Random Patches**

* Combines both **row sampling and column sampling**.
* Models train on **random subsets of both samples and features**.
* Increases diversity among models and reduces overfitting.

**Note:** Random Forest can use Random Patches if both max\_samples and max\_features are set < 1.0.

**🔹 Bagging with Classification and Regression**

* **Classification**:
  + Each model votes for a class.
  + Final output = **majority vote**.
* **Regression**:
  + Each model gives a numeric prediction.
  + Final output = **average** of predictions.

**🔹 Voting Ensemble**

A simple way to combine multiple classifiers or regressors.

**1. Voting Classifier**

* Combines predictions from different models.
* Two types:
  + **Hard Voting** (Majority Voting): Most common class label is chosen.
  + **Soft Voting**: Averages predicted probabilities and selects the highest.

python

CopyEdit

from sklearn.ensemble import VotingClassifier

ensemble = VotingClassifier(estimators=[('lr', model1), ('rf', model2)], voting='soft')

**2. Voting Regressor**

* Averages the outputs from multiple regression models.

python

CopyEdit

from sklearn.ensemble import VotingRegressor

ensemble = VotingRegressor(estimators=[('lr', model1), ('rf', model2)])

**✅ Summary Table**

| **Technique** | **Rows Sampled** | **Columns Sampled** | **With Replacement** | **Typical Use** |
| --- | --- | --- | --- | --- |
| **Bagging** | ✅ | ❌ (or all) | ✅ | Random Forest |
| **Pasting** | ✅ | ❌ | ❌ | Custom ensembles |
| **Random Subspaces** | ❌ | ✅ | N/A | Feature reduction |
| **Random Patches** | ✅ | ✅ | ✅ or ❌ | Variant of Random Forest |
| **Voting Ensemble** | ❌ | ❌ | N/A | Combine different models |
| **1. BaggingClassifier / BaggingRegressor**  python  CopyEdit  from sklearn.ensemble import BaggingClassifier  from sklearn.tree import DecisionTreeClassifier  model = BaggingClassifier(  estimator=DecisionTreeClassifier(),  n\_estimators=10,  max\_samples=0.8,  max\_features=1.0,  bootstrap=True,  bootstrap\_features=False,  oob\_score=True,  n\_jobs=-1,  random\_state=42  )   | **Hyperparameter** | **Type** | **Detailed Description** | | --- | --- | --- | | estimator | Estimator | The base model used in the ensemble. E.g., DecisionTreeClassifier. All models should support fit and predict. | | n\_estimators | int | Number of models to train. Higher values reduce variance but increase training time. Default: 10. | | max\_samples | float/int | If float (0 < x ≤ 1), it's the fraction of training samples to draw **per base model**. If int, it's the fixed number of samples. Allows control over the training set size for diversity. | | bootstrap | bool | If True, samples are drawn **with replacement** (bagging). If False, samples are drawn without replacement (pasting). | | max\_features | float/int | Number or fraction of features to draw per model. Promotes model diversity by giving each model a different "view" of the data. | | bootstrap\_features | bool | If True, **features are sampled with replacement**. Rarely used. Mainly adds further randomness to the model training process. | | oob\_score | bool | If True and bootstrap=True, evaluates model on the **out-of-bag** (unused) samples for performance estimation, acting like built-in cross-validation. | | n\_jobs | int | Controls parallel execution. -1 = use all CPU cores. 1 = no parallelism. | | random\_state | int | Ensures reproducibility by setting the seed for random sampling. | | verbose | int | Controls how much output you see. 0 = silent, higher values = more logs. |   **✅ 2. VotingClassifier / VotingRegressor**  python  CopyEdit  from sklearn.ensemble import VotingClassifier  from sklearn.linear\_model import LogisticRegression  from sklearn.tree import DecisionTreeClassifier  from sklearn.svm import SVC  model = VotingClassifier(  estimators=[  ('lr', LogisticRegression()),  ('dt', DecisionTreeClassifier()),  ('svc', SVC(probability=True))  ],  voting='soft',  weights=[1, 1, 2],  n\_jobs=-1,  verbose=True  )   | **Hyperparameter** | **Type** | **Detailed Description** | | --- | --- | --- | | estimators | list of tuples | List of (name, model) pairs. Each model should be a fully configured and fitted model supporting fit/predict. | | voting | str ('hard' / 'soft') | 'hard': majority vote (classification labels). 'soft': average of predicted probabilities; requires models with predict\_proba(). 'soft' usually gives better results if supported. | | weights | list of numbers | Optional list assigning **importance** to each model in voting. Larger weight = more influence on final prediction. Only used with 'soft' voting. | | n\_jobs | int | Parallelism; use -1 for all cores, useful when training large base models. | | verbose | int/bool | Logging level. Set True or 1 to see training progress for each estimator. |   For **VotingRegressor**, voting is not 'hard' or 'soft', it's always **average of predictions**, so it skips voting.  **✅ 3. StackingClassifier / StackingRegressor**  python  CopyEdit  from sklearn.ensemble import StackingClassifier  from sklearn.linear\_model import LogisticRegression  from sklearn.svm import SVC  from sklearn.tree import DecisionTreeClassifier  model = StackingClassifier(  estimators=[  ('svc', SVC(probability=True)),  ('dt', DecisionTreeClassifier())  ],  final\_estimator=LogisticRegression(),  cv=5,  passthrough=False,  n\_jobs=-1  )   | **Hyperparameter** | **Type** | **Detailed Description** | | --- | --- | --- | | estimators | list of tuples | List of (name, model) base learners. Their predictions are used as features for the final model. | | final\_estimator | estimator | The **meta-model** trained on the base learners’ outputs. For classification, often LogisticRegression; for regression, LinearRegression or others. | | cv | int / cross-validator | Controls **cross-validation** used to get predictions from base models to train the final estimator. This ensures the final estimator gets **unbiased** training data. | | passthrough | bool | If True, original input features are **passed along** with base model predictions to the meta-model. This can sometimes improve accuracy. | | n\_jobs | int | Parallel training of base learners. -1 for all processors. | | verbose | int | Set to non-zero to get more logs. | | stack\_method | str/list | (optional) Method to use when getting output from base models. Defaults to predict\_proba for classifiers, predict for regressors. |   **✅ 4. Bootstrapping Concepts (Under the Hood)**  **1. Bootstrapping is a Concept, Not a Class**  In **Scikit-learn**, **bootstrapping** refers to:   * Drawing samples **with replacement** from the original dataset. * Each model in an ensemble (e.g., Bagging or Random Forest) gets a different random subset of data.   🔸 However, **Scikit-learn does not provide a class called Bootstrap**. Instead, it lets you **enable/disable bootstrapping** using parameters like:  BaggingClassifier(..., bootstrap=True)  RandomForestClassifier(..., bootstrap=True)  **✅ 1. Built-in scikit-learn classes with bootstrap flag:**   | **Class** | **Bootstrapping Supported** | **How** | | --- | --- | --- | | BaggingClassifier | ✅ Yes | bootstrap=True | | BaggingRegressor | ✅ Yes | bootstrap=True | | RandomForestClassifier | ✅ Yes | Internally uses Bagging with bootstrap=True | | RandomForestRegressor | ✅ Yes | Same as above |   **Important Parameters Affecting Bootstrapping:**   | **Parameter** | **Where Used** | **Detailed Description** | | --- | --- | --- | | bootstrap=True | BaggingClassifier, RandomForest | Enables bootstrapping (sampling with replacement). Creates unique training data for each model. Helps reduce variance. | | oob\_score=True | Bagging/RandomForest | Activates **Out-of-Bag** error estimation. Trains on bootstrapped data but evaluates on unused data points. Acts like built-in cross-validation. | | max\_samples | Bagging | Number of data points to sample per model. Smaller values = more model diversity. | | max\_features | Bagging | Same idea but for columns/features. Useful when training similar models that might overfit the same data. |   **✅ Summary Table**   | **Ensemble Method** | **Key Hyperparameters** | **Use Case** | | --- | --- | --- | | **BaggingClassifier** | n\_estimators, max\_samples, bootstrap, oob\_score | Improves stability, reduces variance | | **VotingClassifier** | voting, weights, estimators | Combines different models' strengths | | **StackingClassifier** | cv, final\_estimator, passthrough | Learns to best combine models via meta-learner | | **Bootstrapping** | bootstrap, oob\_score, max\_samples | Creates diverse training sets per model | |  |  |  |  |

**Learn more with coding:**

URL: <https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/>

<https://www.geeksforgeeks.org/a-comprehensive-guide-to-ensemble-learning/>

<https://github.com/campusx-official/voting-ensemle>

<https://github.com/campusx-official/bagging-ensemble>

try first:- <https://medium.com/@sumbatilinda/ensemble-learning-in-machine-learning-bagging-boosting-and-stacking-a00c6bae971f>

**Pay attention on below terms:**

Row sampling, column sampling With or without replacement

Bagging, Pasting, Random subspaces, Random Patches,

Bagging with classification and Regression

Voting ensemble with Classification and Regression

GridSearchCV to test with multiple different hyperparameter values

When to use ensemble learning