

Bagging (Bootstrap Aggregating)

```
from sklearn.ensemble import BaggingClassifier
```

```
bootstrap=True (default)
```

- Improves model accuracy by reducing variance and preventing overfitting.
- It works by training multiple models independently on different random subsets of the dataset and averaging their predictions.

Bootstrap Sampling Explained

1. What is Bootstrap Sampling?

- Bootstrap sampling is a technique where you randomly select samples from the original dataset **with replacement**.
- "With replacement" means that after a sample is selected, it is **put back** into the dataset and can be selected again.

2. How Does It Work?

- Suppose you have a dataset with **100 rows**.
- To create a bootstrap sample:
 - Randomly pick one row from the dataset.
 - **Put it back** into the dataset.
 - Repeat this process **100 times** (since the subset must be the same size as the original dataset).

3. Why Are Some Samples Repeated and Some Left Out?

- Because of **random selection with replacement**, some rows may be picked multiple times, while others may not be picked at all.
- This is similar to drawing balls from a bag: if you draw a ball, note its color, and put it back, you might draw the same ball multiple times, while some balls may never be drawn.

Example of Bootstrap Sampling:

Let's say you have a small dataset of 5 data points:

{5,8,12,15,18}

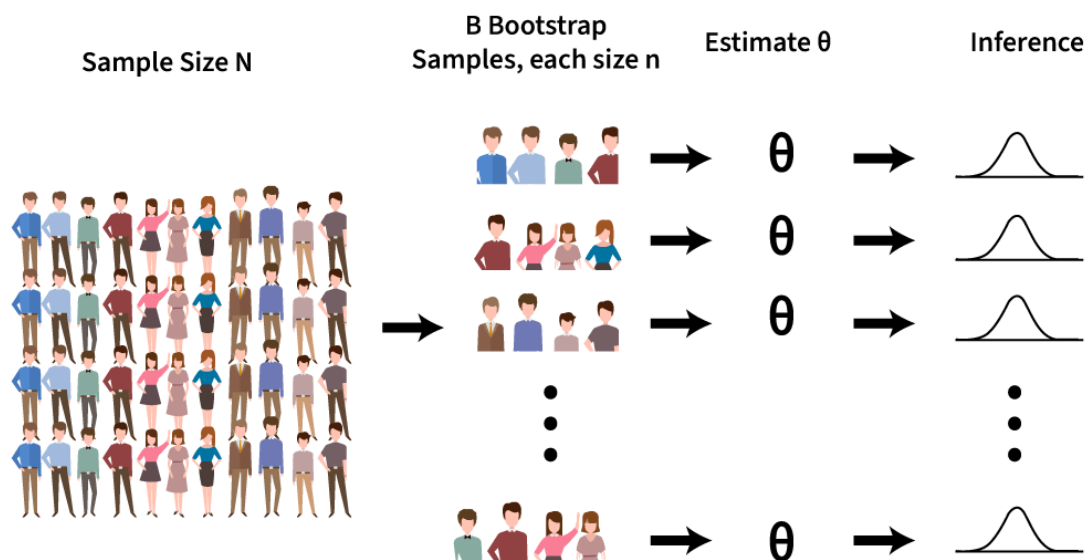
To create a bootstrap sample:

1. Randomly pick a data point, e.g., you might pick 8.
2. Pick another data point (with replacement), say 5.
3. Keep repeating until you have a sample of the same size as the original dataset. For instance, one possible bootstrap sample could be:

{8,5,18,12,8}

Notice that the data point 8 appeared twice, while 15 didn't appear at all.

Bootstrap Method




How Does Bagging Work?

1. Create Subsets of the Data:

- Randomly sample the training data **with replacement** (this is called **bootstrapping**).
- Each subset is the same size as the original dataset, but some samples may be repeated, and some may be left out.

2. Train Multiple Models:

- Train a separate model (e.g., Decision Tree) on each subset.
- The models are trained independently of each other.
-  **NOTE: All models will be same.**
 - If we pick Linear Regression, we will use LR in all models.

3. Combine Predictions:

- For **regression**: **Average** the predictions of all models.
- For **classification**: Take a **majority vote** of all models.

Pasting

- Type of bagging with `Replace=False`
- This means that, in contrast to bootstrapping, each data point in the dataset can only appear once in each subset that is created.

Advantages of Pasting:

- **More Diverse Subsets**: Since each data point only appears once in each subset, pasting creates more diverse subsets, and therefore the models trained on these subsets are less likely to be highly correlated.
- **Less Overfitting Risk**: Because there is no repetition of data points, pasting can reduce the risk of overfitting that can occur when certain data points are overrepresented in the resampled datasets (as in bootstrap sampling).

- **Useful When You Want to Maximize the Use of Data:** Since every data point is included exactly once in each subset, you're using all of your available data to train the models.

Disadvantages of Pasting:

- **Less Robustness Against Outliers:** Since each model is trained on a non-overlapping subset of the data, it may not generalize well to unseen data, especially if the subsets contain few or no outliers that could influence model learning.

When to Use Pasting:

Pasting is often used when:

- You want to ensure that each data point is used in training every model, but without duplication.
- You want models that are slightly less correlated with each other compared to those in bagging (which uses bootstrapping).
- You are dealing with smaller datasets and want to maximize the use of all the data points.

Random Subspaces:

- Randomly selects **features** (columns) for each model.
- Trains on a subset of features.

Random Patches:

- Randomly selects **both data points (rows)** and **features (columns)**.
- Trains on a subset of data and features.

Python code:

```

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

# Generate synthetic dataset
X, y = make_classification(n_samples=1000, n_features=20, random_state=42)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Bagging Classifier with Decision Trees
bagging = BaggingClassifier(estimator=DecisionTreeClassifier(), n_estimators=10, random_state=42)

# Train the model
bagging.fit(X_train, y_train)

# Predict on test data
y_pred = bagging.predict(X_test)

# Evaluate accuracy
print("Bagging Accuracy:", accuracy_score(y_test, y_pred))

```

Bagging Accuracy: 0.885

estimator → Model. By default it's `DecisionTreeClassifier`.

n_estimators refers to the number of models (estimators) in an ensemble method.

- In algorithms like **Random Forest** or **Gradient Boosting**, `n_estimators` controls how many individual trees (or models) are trained.
- More estimators generally improve model performance, but also increase computational cost and the risk of overfitting.
- **default value=10**

```
X, y = make_classification(n_samples=1000, n_features=20,
random_state=42)
```

- `n_samples=1000` : This specifies that the dataset will contain 1000 data points (rows).
- `n_features=20` : This indicates that each data point will have 20 features (columns).
- The features (values in `x`) are generated from a **normal distribution** (Gaussian distribution) with a mean of 0 and a standard deviation of 1 by default. So, most of the values will be within the range of approximately **-3 to +3**, but some may fall outside this range due to the nature of the Gaussian distribution.
- The target values (`y`) will be **integer class labels**. These labels are usually **0 or 1**



It's not mandatory to write these default values →

```
estimator=DecisionTreeClassifier() , n_estimators=10
```

Bagging Using SVM/SVC

```
from sklearn.ensemble import BaggingClassifier
from sklearn.svm import SVC
```

```

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Generate synthetic dataset
X, y = make_classification(n_samples=1000, n_features=20, random_state=42)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Bagging Classifier with Decision Trees
bagging = BaggingClassifier(estimator=SVC())

# Train the model
bagging.fit(X_train, y_train)

# Predict on test data
y_pred = bagging.predict(X_test)

# Evaluate accuracy
print("Bagging Accuracy:", accuracy_score(y_test, y_pred))

```

```
Bagging Accuracy: 0.86
```

- The accuracy seems to be decreased with SVM.

Pasting

bootstrap= False

```

from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

```

```

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Generate synthetic dataset
X, y = make_classification(n_samples=1000, n_features=20, random_state=42)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Bagging Classifier with Decision Trees
bagging = BaggingClassifier(bootstrap=False, random_state=42)

# Train the model
bagging.fit(X_train, y_train)

# Predict on test data
y_pred = bagging.predict(X_test)

# Evaluate accuracy
print("Bagging Accuracy:", accuracy_score(y_test, y_pred))

```

Bagging Accuracy: 0.87

Random Subspaces

```

from sklearn.ensemble import BaggingClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Generate synthetic dataset

```



```

X, y = make_classification(n_samples=1000, n_features=20, random_state=42)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Bagging Classifier with Decision Trees
bag = BaggingClassifier(n_estimators=500, bootstrap=False, max_features=0.5, bootstrap_features=True, random_state=42)

# Train the model
bag.fit(X_train, y_train)

# Predict on test data
y_pred = bag.predict(X_test)

# Evaluate accuracy
print("Bagging Accuracy:", accuracy_score(y_test, y_pred))

```

```
Bagging Accuracy: 0.89
```

```
bag.estimators_features_[1].shape
```

Output:
(10,)

- As we set `max_features=0.5`, only 10 out of 20 columns are used.

Random Patches:

```

from sklearn.ensemble import BaggingClassifier
from sklearn.datasets import make_classification

```

```

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Generate synthetic dataset
X, y = make_classification(n_samples=1000, n_features=20, random_state=42)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Bagging Classifier with Decision Trees
bag = BaggingClassifier(n_estimators=500, bootstrap=False, max_samples=0.5, max_features=0.5, bootstrap_features=True, random_state=42)

# Train the model
bag.fit(X_train, y_train)

# Predict on test data
y_pred = bag.predict(X_test)

# Evaluate accuracy
print("Bagging Accuracy:", accuracy_score(y_test, y_pred))

```

Bagging Accuracy: 0.895

OOB score (Out-Of-Bag score) (IMP for Interview)

`oob_score=True`

`model.oob_score_`

- It **evaluates** the performance of the model using data points that were **not included** in the bootstrap sample for each individual model.

- When training a **BaggingClassifier**, some data points in the training set will not be selected for a given bootstrap sample.
 - These data points are called **out-of-bag (OOB)** points, and their performance can be used to estimate the generalization error of the model without needing a separate validation set.
- You take these and do the prediction.
- Only works with `bootstrap=True`

```
from sklearn.ensemble import BaggingClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Generate synthetic dataset
X, y = make_classification(n_samples=1000, n_features=20, random_state=42)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Bagging Classifier with Decision Trees
bag = BaggingClassifier(n_estimators=500, max_samples=0.6, max_features=0.5, bootstrap=True, random_state=42, oob_score=True)

# Train the model
bag.fit(X_train, y_train)

# Predict on test data
y_pred = bag.predict(X_test)
```

```
# Evaluate accuracy  
print("Bagging Accuracy:", accuracy_score(y_test, y_pred))
```

```
Bagging Accuracy: 0.89
```

```
bag.oob_score_
```

```
Output:  
0.90125
```

- This is the accuracy on the points which are never used.

Bagging Tips

- Bagging generally gives better results than Pasting
- **Good results come around the 25% to 50% row sampling mark (`max_samples=0.25 to 0.5`)**
- Random patches and subspaces should be used while dealing with high dimensional data
- To find the correct hyperparameter values we can do GridSearchCV/RandomSearchCV

Bagging Regressor

```
from sklearn.ensemble import BaggingRegressor
```

- `BaggingRegressor` is the regression version of `BaggingClassifier`, used for **regression tasks**.

Same hyperparameters as `BaggingClassifier`

Key Parameters:

- `base_estimator` : The model used as the base estimator (e.g., `DecisionTreeRegressor`). If None, the default is `DecisionTreeRegressor`.
- `n_estimators` : Number of base models (estimators) in the ensemble.
- `max_samples` : The fraction of the training data used to train each base model. Can be set as an integer (number of samples) or float (fraction of data).
- `max_features` : The number of features to train each base model on. Can be set as an integer (number of features) or float (fraction of features).
- `bootstrap` : Whether to sample the data with replacement. Default is `True`.
- `oob_score` : Whether to use out-of-bag samples to estimate the generalization error. Default is `False`.
- `n_jobs` : Number of parallel jobs to run. `1` means using all processors.

```
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

# Generate synthetic regression dataset
X, y = make_regression(n_samples=1000, n_features=20, noise=0.2, random_state=42)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```

# Create Bagging Regressor using Decision Tree as base estimator
bag_reg = BaggingRegressor(
    n_estimators=100,      # Number of trees in the ensemble
    max_samples=0.8,      # Use 80% of data for each model
    max_features=0.5,     # Use 50% of features for each model
    random_state=42,
    n_jobs=-1             # Use all processors
)

# Train the model
bag_reg.fit(X_train, y_train)

# Predict on test data
y_pred = bag_reg.predict(X_test)

# Evaluate model performance using Mean Squared Error
r2 = r2_score(y_test, y_pred)
print(f"R2: {r2}")

```

R2: 0.6592299053742958

Bagging Regressor on *Boston Housing Dataset*:

```

from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
import pandas as pd

# Load the Boston Housing dataset
df = pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master/BostonHousing.csv')

```

```

X = df.iloc[:,0:-1]
y = df.iloc[:, -1]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create Bagging Regressor using Decision Tree as base estimator
bag_reg = BaggingRegressor(
    n_estimators=150,      # Number of trees in the ensemble
    max_samples=0.8,      # Use 80% of data for each model
    n_jobs=-1             # Use all processors
)

# Train the model
bag_reg.fit(X_train, y_train)

# Predict on test data
y_pred = bag_reg.predict(X_test)

# Evaluate model performance using Mean Squared Error
r2 = r2_score(y_test, y_pred)
print(f"R2: {r2}")

```

```
R2: 0.8852761233246079
```

Bagging Regressor vs Others

```

from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV

```

```

from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score

lr = LinearRegression()
dt = DecisionTreeRegressor()
knn = KNeighborsRegressor()

lr.fit(X_train,y_train)
dt.fit(X_train,y_train)
knn.fit(X_train,y_train)

y_pred1 = lr.predict(X_test)
y_pred2 = dt.predict(X_test)
y_pred3 = knn.predict(X_test)

print("R^2 score for LR",r2_score(y_test,y_pred1))
print("R^2 score for DT",r2_score(y_test,y_pred2))
print("R^2 score for KNN",r2_score(y_test,y_pred3))

```

```

R^2 score for LR 0.6687594935356317
R^2 score for DT 0.6772947602240251
R^2 score for KNN 0.6473640882039258

```

- R2 score for **Bagging Regressor** was **0.8796533342259916**

Hyperparameter Tuning:

```

params = {'n_estimators': [20,50,100,150],
          'max_samples': [0.5,0.7,0.8,0.9,1.0],
          'max_features': [0.5, 0.6, 0.7,0.8],
          'bootstrap': [True, False],
          'bootstrap_features': [True, False]
        }

```



```
bagging_regressor_grid = GridSearchCV(BaggingRegressor(n_jobs=-1), param_grid = params, cv=5, scoring='r2', n_jobs=-1)
bagging_regressor_grid.fit(X_train, y_train)
```

```
print(f"Best parameters: {bagging_regressor_grid.best_params_}")
print(f"Best score: {bagging_regressor_grid.best_score_}")
```

```
Best parameters: {'bootstrap': True, 'bootstrap_features': False, 'max_features': 0.8, 'max_samples': 0.8, 'n_estimators': 100}
Best score: 0.836345974357988
```

```
best_bag_reg = bagging_regressor_grid.best_estimator_
best_bag_reg
```

```
y_pred_best = best_bag_reg.predict(X_test)
r2_best = r2_score(y_test, y_pred_best)
print(f"Test R2 with GridSearch Best Model: {r2_best}")
```

```
Test R2 with GridSearch Best Model: 0.8603597677837459
```



If we remove `'bootstrap_features': [True, False]` , we get best results.