Bagging

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Ensemble Methods

Voting Classifier

- same training set,
- ≠ algorithms.

Bagging

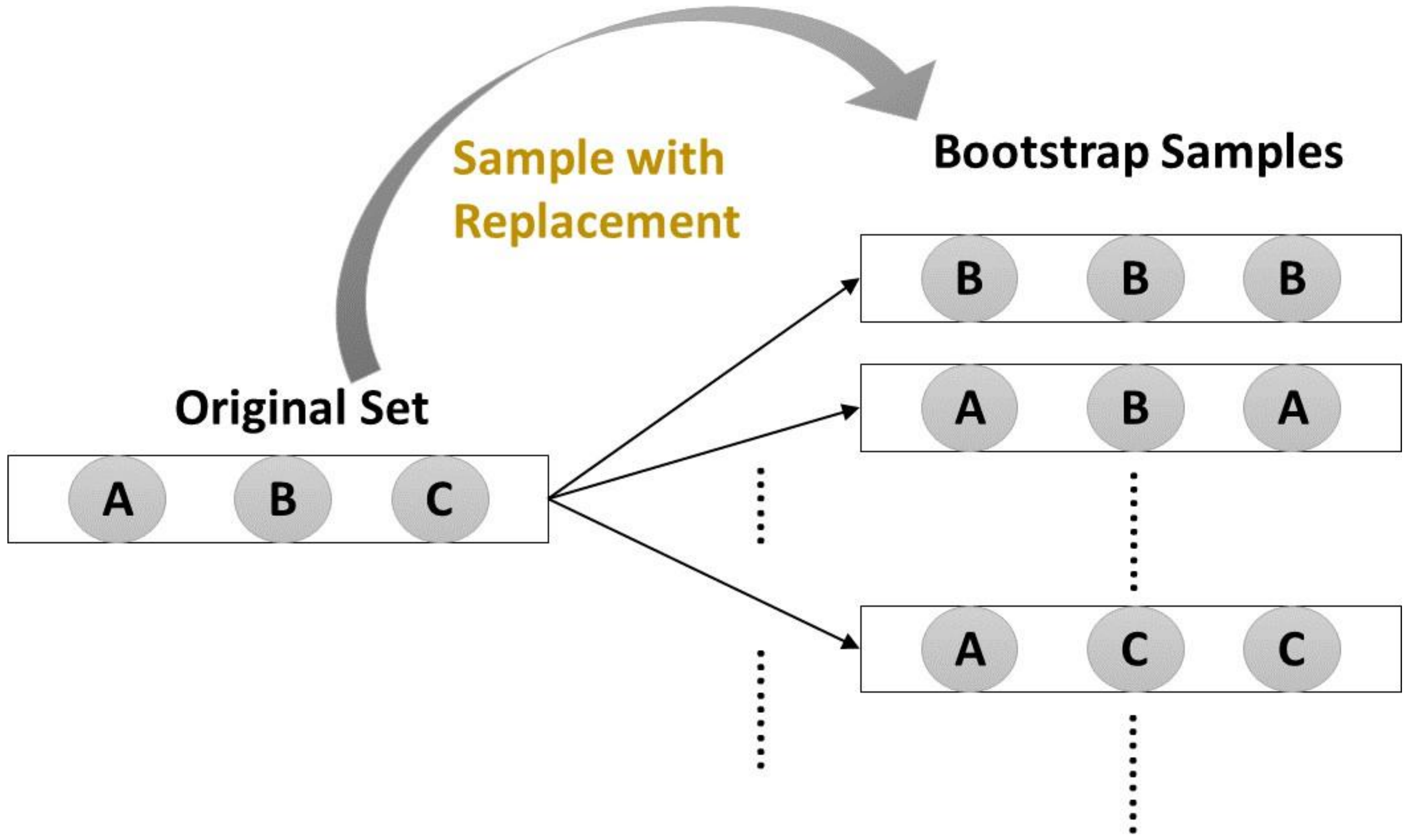
- one algorithm,
- ≠ subsets of the training set.

Bagging

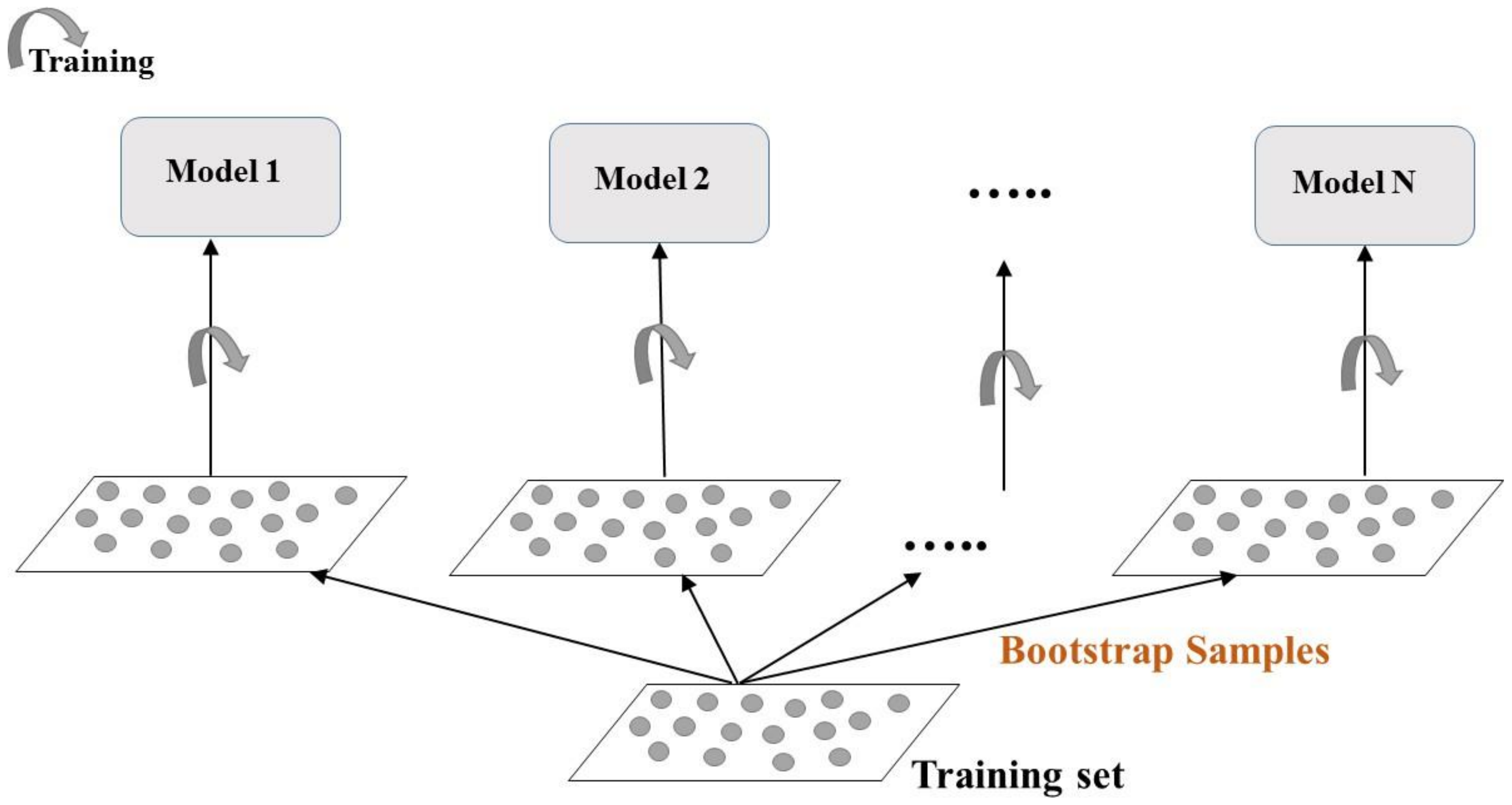
- Bagging: Bootstrap Aggregation.
- Uses a technique known as the bootstrap.
- Reduces variance of individual models in the ensemble.

Bagging, short for **Bootstrap Aggregating**, is an ensemble learning method designed to improve the stability and accuracy of machine learning algorithms.
It reduces variance and prevents overfitting by training multiple models on different subsets of the data and then combining their predictions.

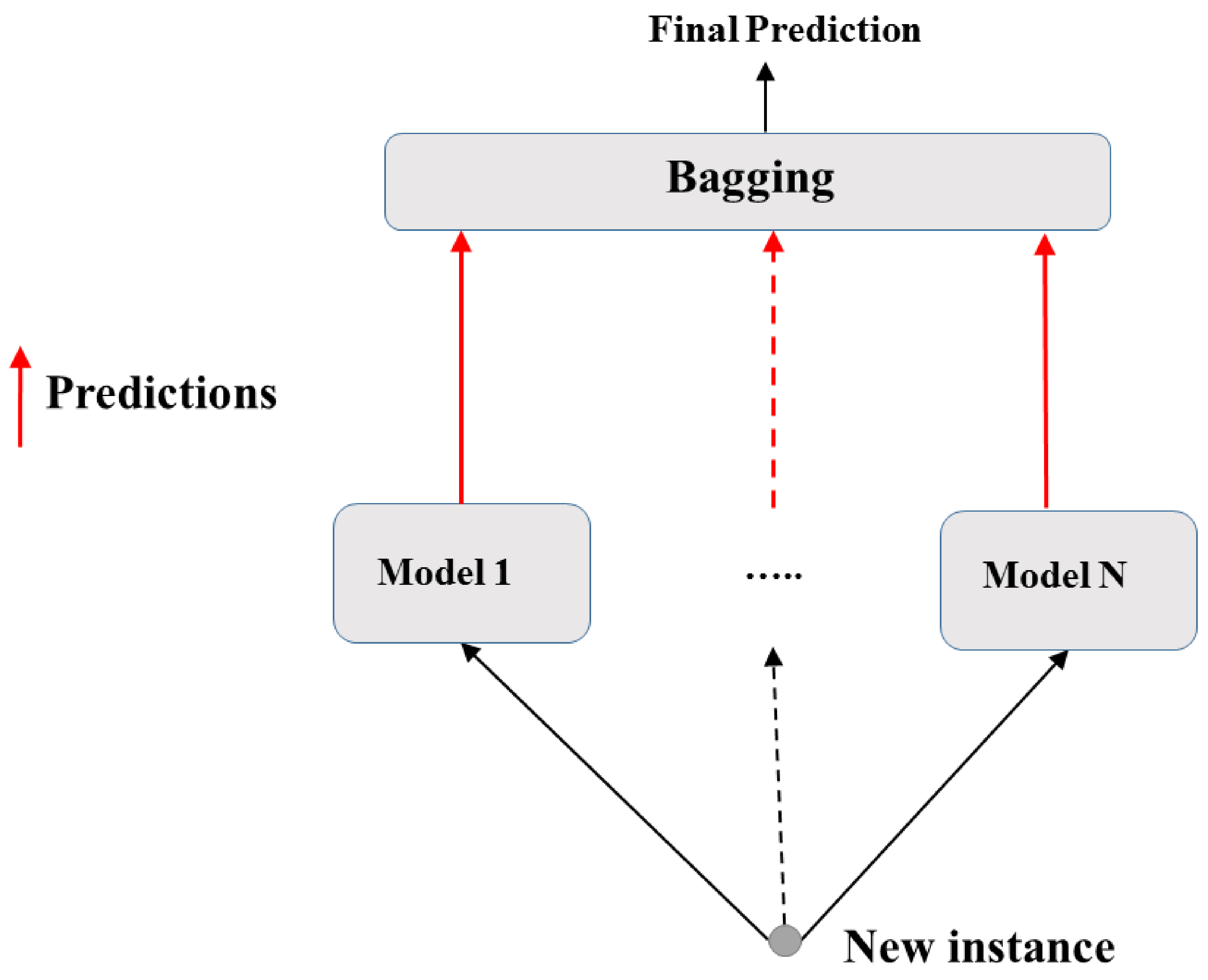
Bootstrap


Sample with Replacement
Bootstrap Samples
Original Set
A B C
B B B
A B A
A C C

Bagging: Training



Bagging: Prediction



Bagging: Classification & Regression

Classification:

- Aggregates predictions by majority voting.
- `BaggingClassifier` in scikit-learn.

Regression:

- Aggregates predictions through averaging.
- `BaggingRegressor` in scikit-learn.

Bagging Classifier in sklearn (Breast-Cancer dataset)

```python
# Import models and utility functions
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_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)
```

```python
# Instantiate a classification-tree 'dt'
dt = DecisionTreeClassifier(max_depth=4, min_samples_leaf=0.16, random_state=SEED)
# Instantiate a BaggingClassifier 'bc'
bc = BaggingClassifier(base_estimator=dt, n_estimators=300, n_jobs=-1)
# Fit 'bc' to the training set
bc.fit(X_train, y_train)
# Predict test set labels
y_pred = bc.predict(X_test)

# Evaluate and print test-set accuracy
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy of Bagging Classifier: {:.3f}'.format(accuracy))
```

```
Accuracy of Bagging Classifier: 0.936
```

Out Of Bag Evaluation

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

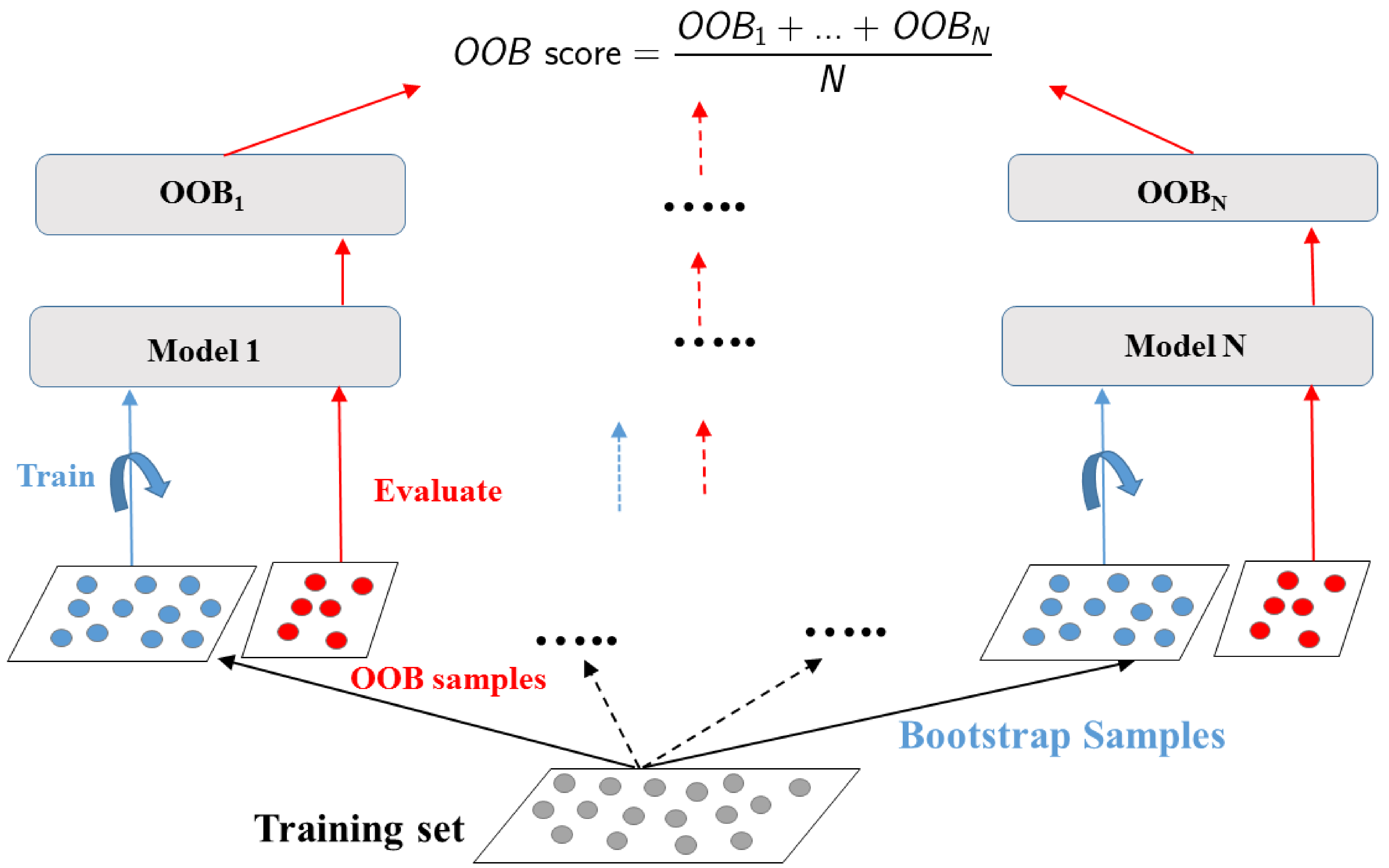
Bagging

- some instances may be sampled several times for one model,
- other instances may not be sampled at all.

Out Of Bag (OOB) instances

- On average, for each model, 63% of the training instances are sampled.
- The remaining 37% constitute the OOB instances.

OOB Evaluation



OOB Evaluation in sklearn (Breast Cancer Dataset)

```
# Import models and split utility function
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_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=4,
                            min_samples_leaf=0.16,
                            random_state=SEED)
# Instantiate a BaggingClassifier 'bc'; set oob_score = True
bc = BaggingClassifier(base_estimator=dt, n_estimators=300,
                       oob_score=True, n_jobs=-1)

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

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

```python
# Evaluate test set accuracy
test_accuracy = accuracy_score(y_test, y_pred)
# Extract the OOB accuracy from 'bc'
oob_accuracy = bc.oob_score_


# Print test set accuracy
print('Test set accuracy: {:.3f}'.format(test_accuracy))
```

```
Test set accuracy: 0.936
```

```python
# Print OOB accuracy

print('OOB accuracy: {:.3f}'.format(oob_accuracy))
```

```
OOB accuracy: 0.925
```

Random Forests

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Bagging

- Base estimator: Decision Tree, Logistic Regression, Neural Net, ...
- Each estimator is trained on a distinct bootstrap sample of the training set
- Estimators use all features for training and prediction

Random Forest is an ensemble learning algorithm that combines multiple decision trees to make robust predictions.

It is widely used for both **classification** and **regression** tasks.

Random Forest builds on the **bagging** (bootstrap aggregating) technique and introduces additional randomness during the training process, making it one of the most powerful and versatile machine learning algorithms.

How Random Forest Works

1. **Bootstrap Sampling**:
    1. Random Forest creates multiple **bootstrap samples** (subsets of the training data created by sampling with replacement).
    2. Each decision tree is trained independently on one of these subsets.

2. **Random Feature Selection**:
    1. At each split in a tree, Random Forest considers only a **random subset of features**, rather than all features.
    2. This introduces additional randomness, decorrelating the trees and reducing the risk of overfitting.

3. **Decision Tree Training**:
    1. Each decision tree is trained to completion (i.e., without pruning) on its bootstrap sample.
    2. Trees can grow deep and overfit to their respective samples, but the ensemble mitigates this by averaging predictions.

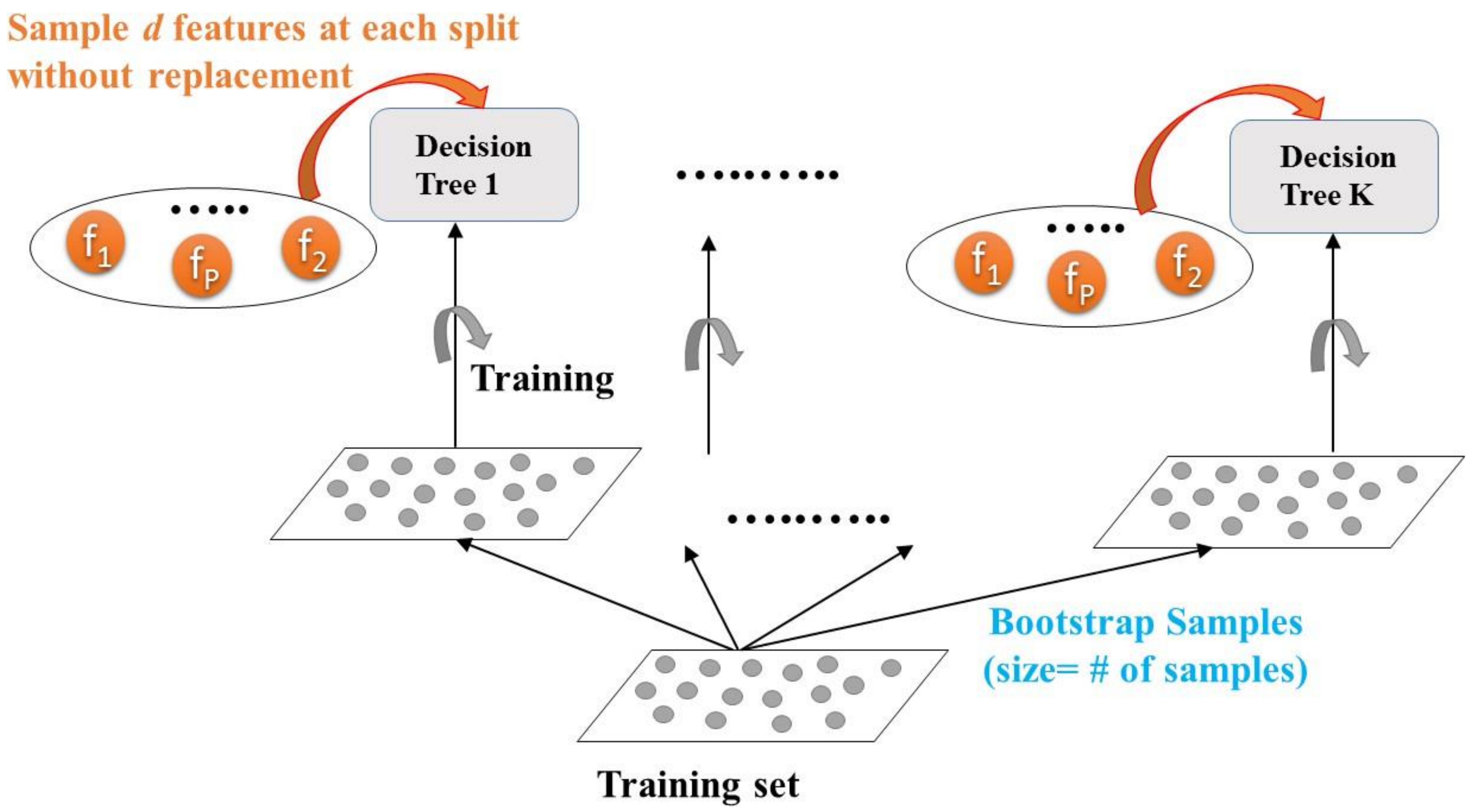
4. **Aggregation**:

    1. For **classification**: The predictions from all trees are aggregated using **majority voting**.
    2. For **regression**: The predictions from all trees are averaged.

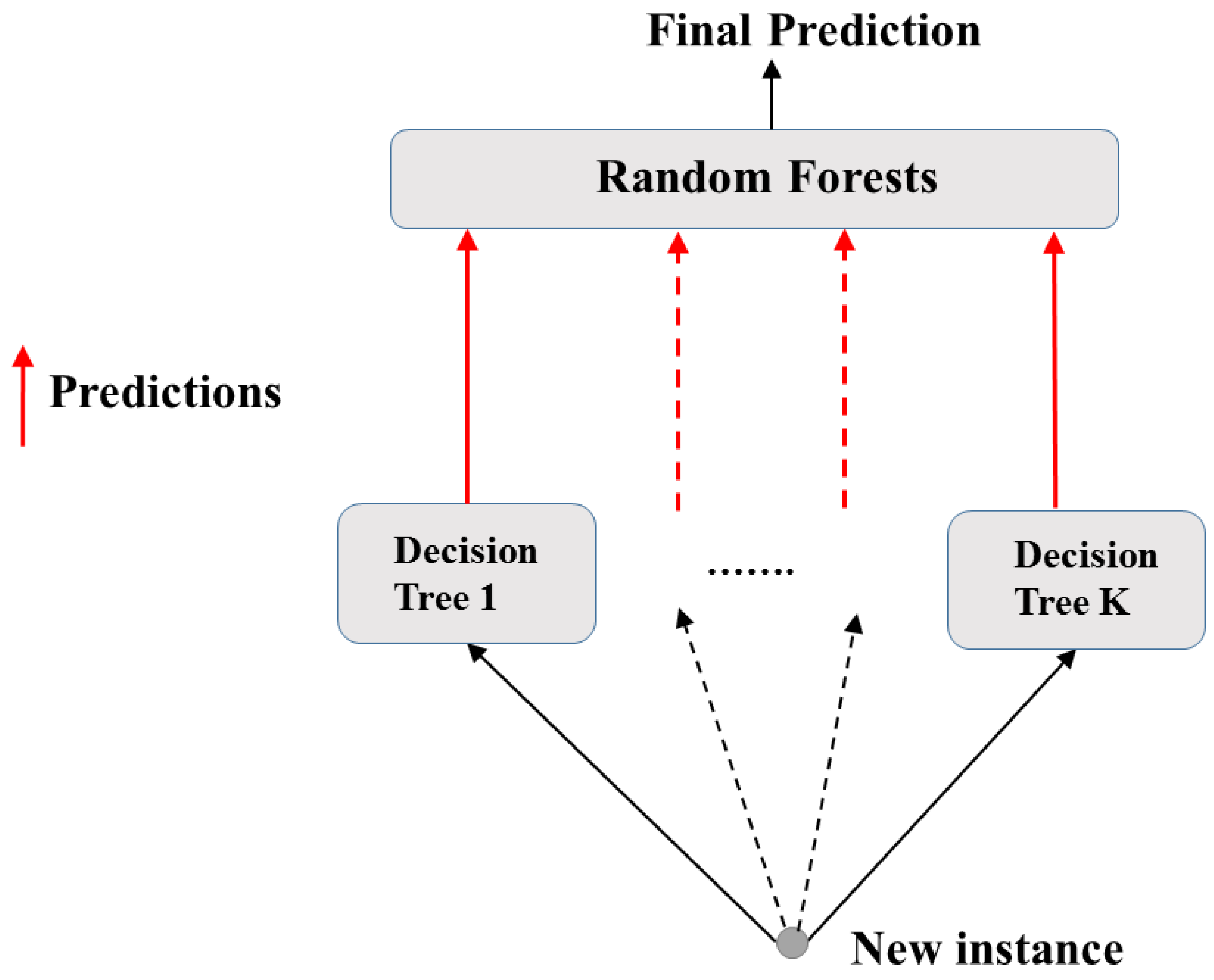
Further Diversity with Random Forests

- Base estimator: Decision Tree
- Each estimator is trained on a different bootstrap sample having the same size as the training set
- RF introduces further randomization in the training of individual trees
- d features are sampled at each node without replacement
  ($d <$ total number of features)

Random Forests: Training


Sample d features at each split without replacement
Decision Tree 1
f_1
f_P
f_2
Training
Decision Tree K
Bootstrap Samples (size= # of samples)
Training set

Random Forests: Prediction



Random Forests: Classification & Regression

Classification:

- Aggregates predictions by majority voting
- `RandomForestClassifier` in scikit-learn

Regression:

- Aggregates predictions through averaging
- `RandomForestRegressor` in scikit-learn

Random Forests Regressor in sklearn (auto dataset)

```
# Basic imports
from sklearn.ensemble import RandomForestRegressor
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)
```

datacamp

```python
# Instantiate a random forests regressor 'rf' 400 estimators
rf = RandomForestRegressor(n_estimators=400,
                                    min_samples_leaf=0.12,
                                    random_state=SEED)
# Fit 'rf' to the training set
rf.fit(X_train, y_train)
# Predict the test set labels 'y_pred'
y_pred = rf.predict(X_test)
```

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

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

```
Test set RMSE of rf: 3.98
```

Feature Importance

Tree-based methods: enable measuring the importance of each feature in prediction.

In `sklearn` :

Identify how much the tree nodes use a particular feature (weighted average) to reduce impurity by accessing the attribute `feature_importance_`

Feature Importance in sklearn

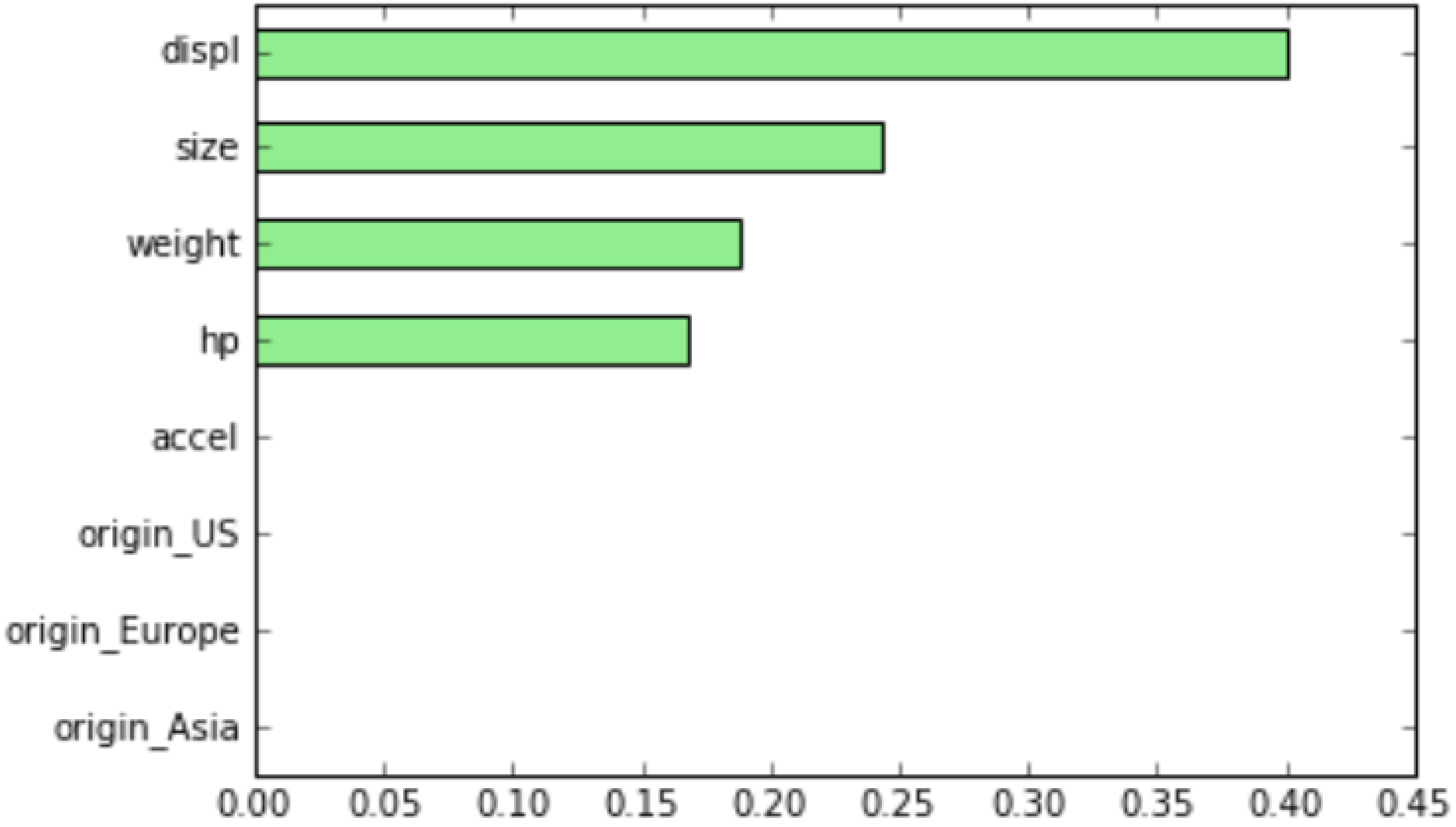
```python
import pandas as pd
import matplotlib.pyplot as plt

# Create a pd.Series of features importances
importances_rf = pd.Series(rf.feature_importances_, index = X.columns)

# Sort importances_rf
sorted_importances_rf = importances_rf.sort_values()

# Make a horizontal bar plot
sorted_importances_rf.plot(kind='barh', color='lightgreen'); plt.show()
```

Feature Importance in sklearn


displ
size
weight
hp
accel
origin_US
origin_Europe
origin_Asia