Decision-Tree for Classification

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

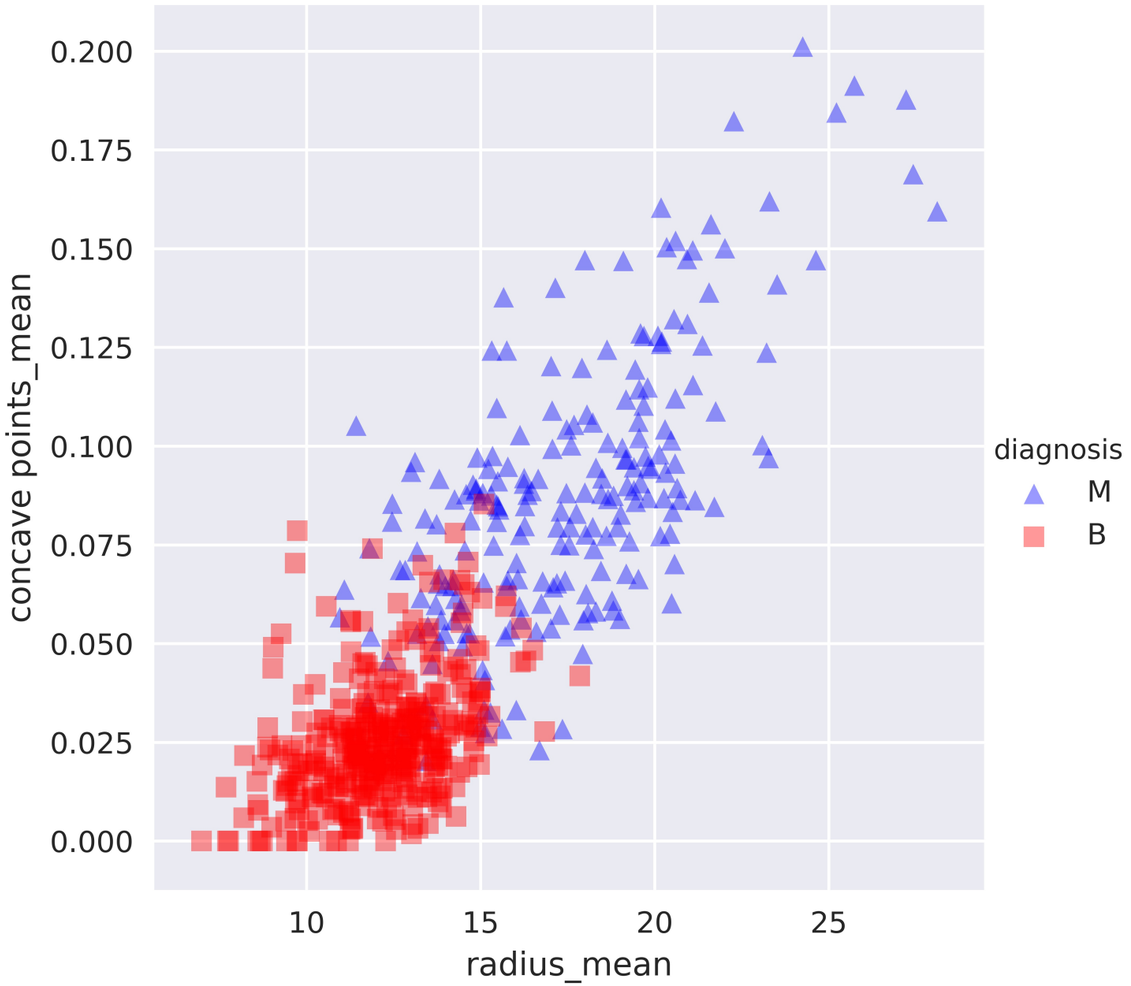
Course Overview

- **Chap 1**: Classification And Regression Tree (CART)
- **Chap 2**: The Bias-Variance Tradeoff
- **Chap 3**: Bagging and Random Forests
- **Chap 4**: Boosting
- **Chap 5**: Model Tuning

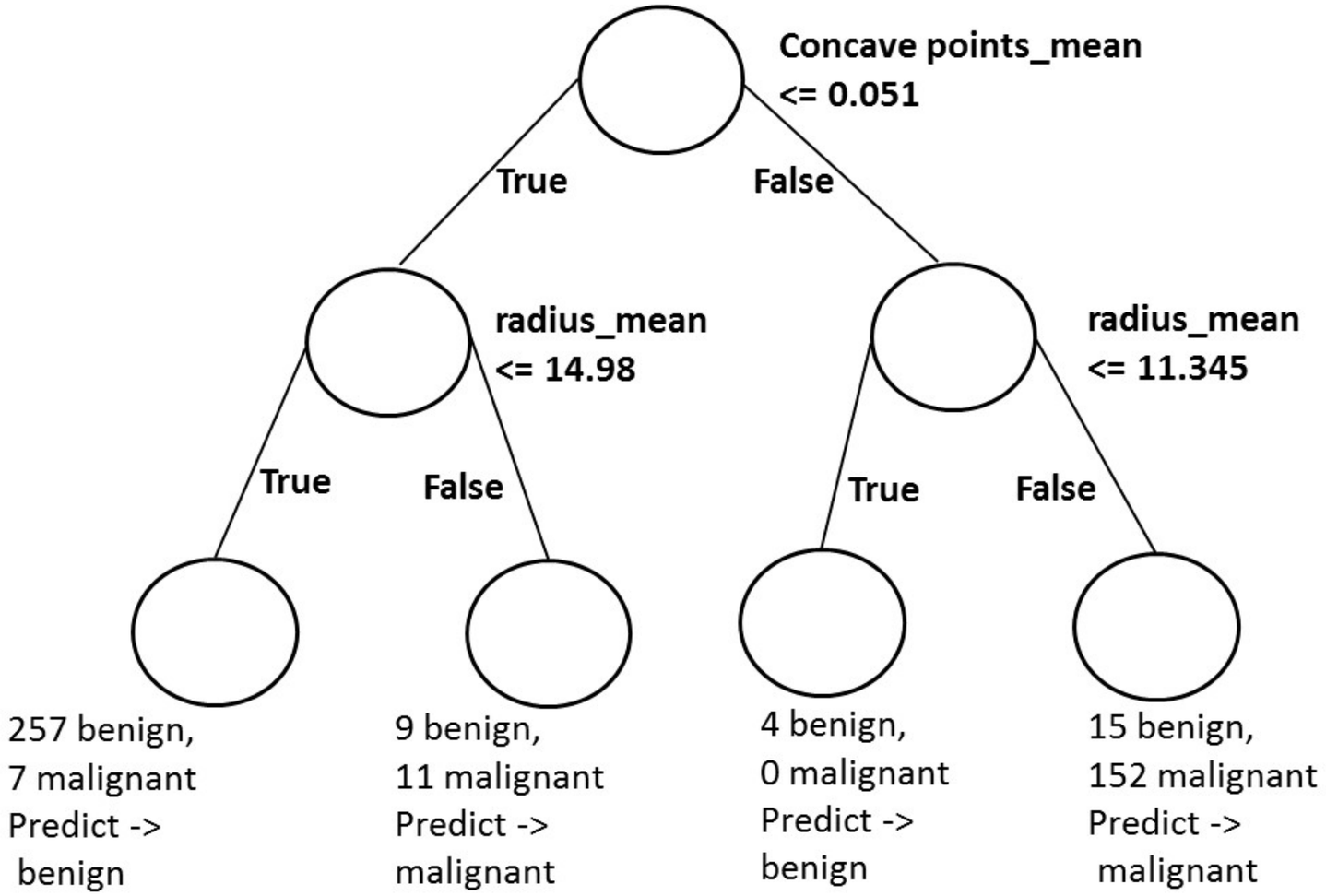
Classification-tree

- Sequence of if-else questions about individual features.
- **Objective**: infer class labels.
- Able to capture non-linear relationships between features and labels.
- Don't require feature scaling (ex: Standardization, ..)

Breast Cancer Dataset in 2D



Decision-tree Diagram



Classification-tree in scikit-learn

```
# Import DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier
# Import train_test_split
from sklearn.model_selection import train_test_split
# Import accuracy_score
from sklearn.metrics import accuracy_score

# Split dataset into 80% train, 20% test
X_train, X_test, y_train, y_test= train_test_split(X, y,
                                                   test_size=0.2,
                                                   stratify=y,
                                                   random_state=1)

# Instantiate dt
dt = DecisionTreeClassifier(max_depth=2, random_state=1)
```

Classification-tree in scikit-learn

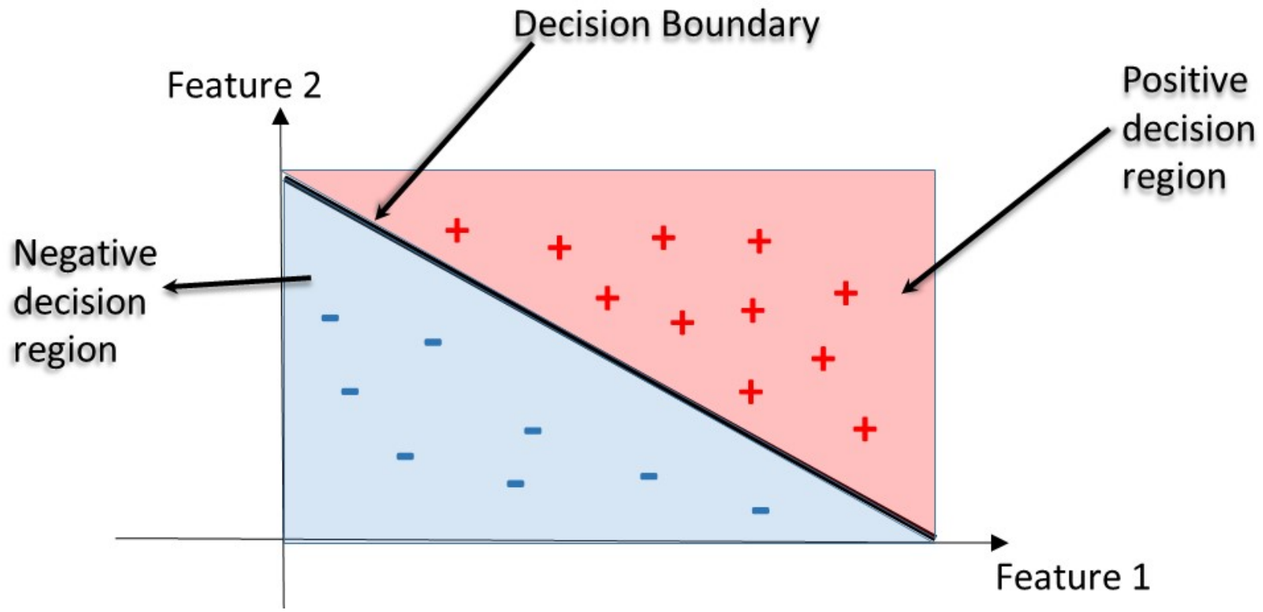
```
# Fit dt to the training set
dt.fit(X_train,y_train)


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

# Evaluate test-set accuracy
accuracy_score(y_test, y_pred)
```

```
0.903508771929824 59
```

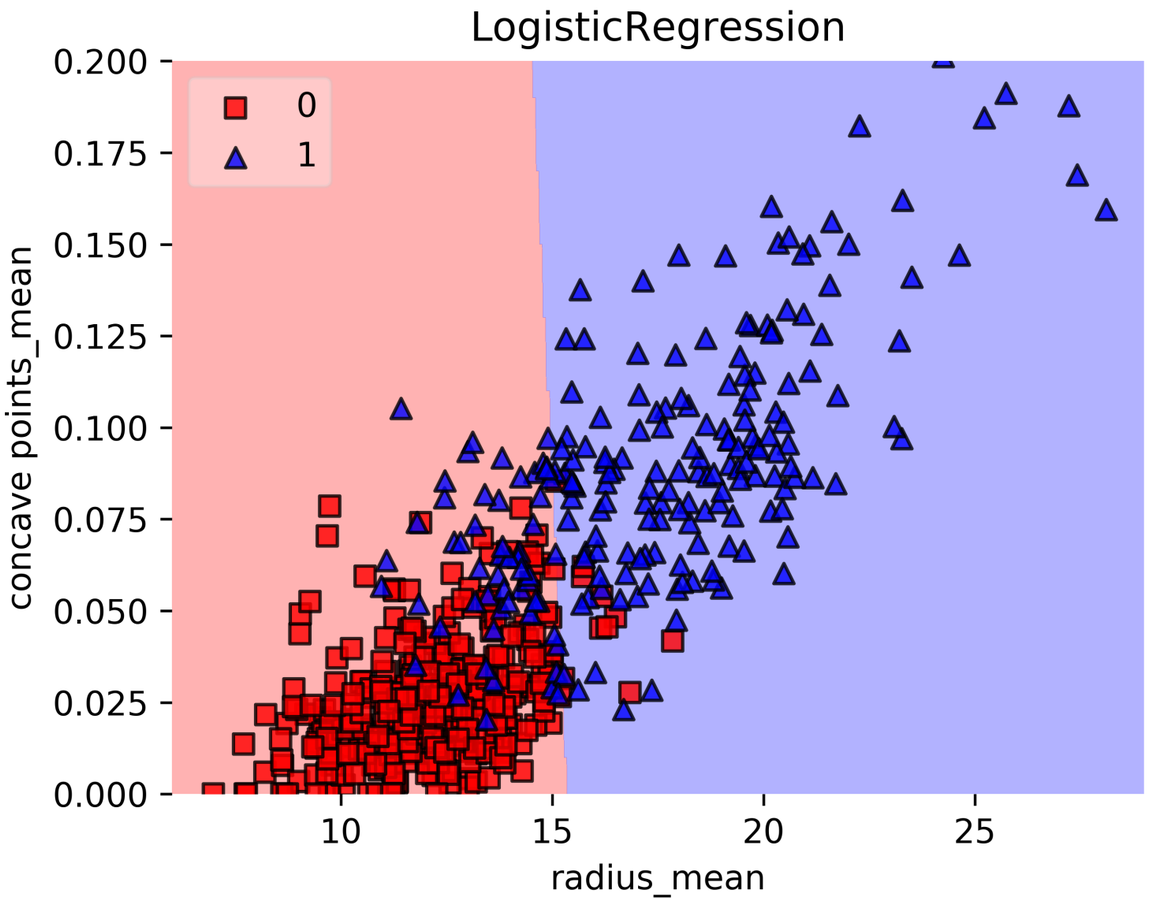
Decision Regions

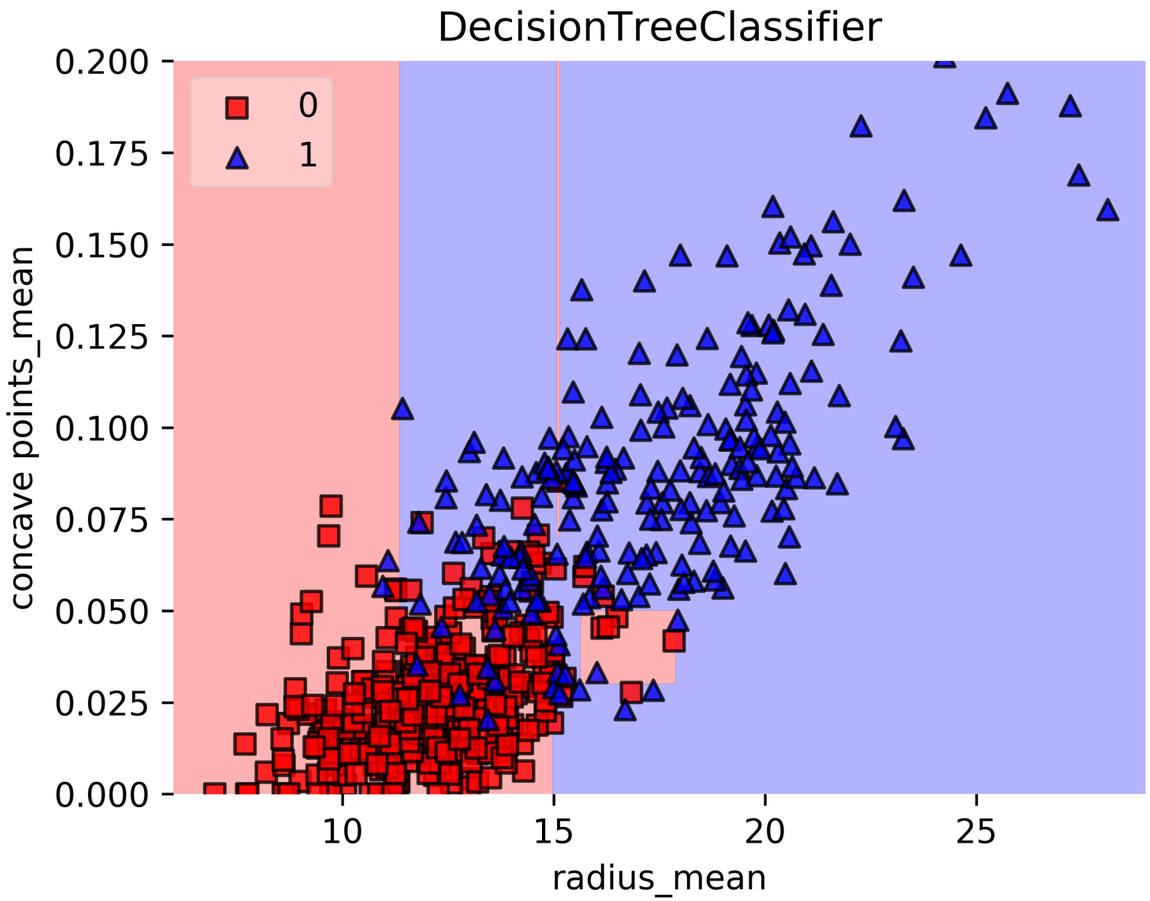
Decision region: region in the feature space where all instances are assigned to one class label.

Decision Boundary: surface separating different decision regions.



Decision Regions: CART vs. Linear Model





Let's practice!

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Classification-Tree Learning

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

Building Blocks of a Decision-Tree

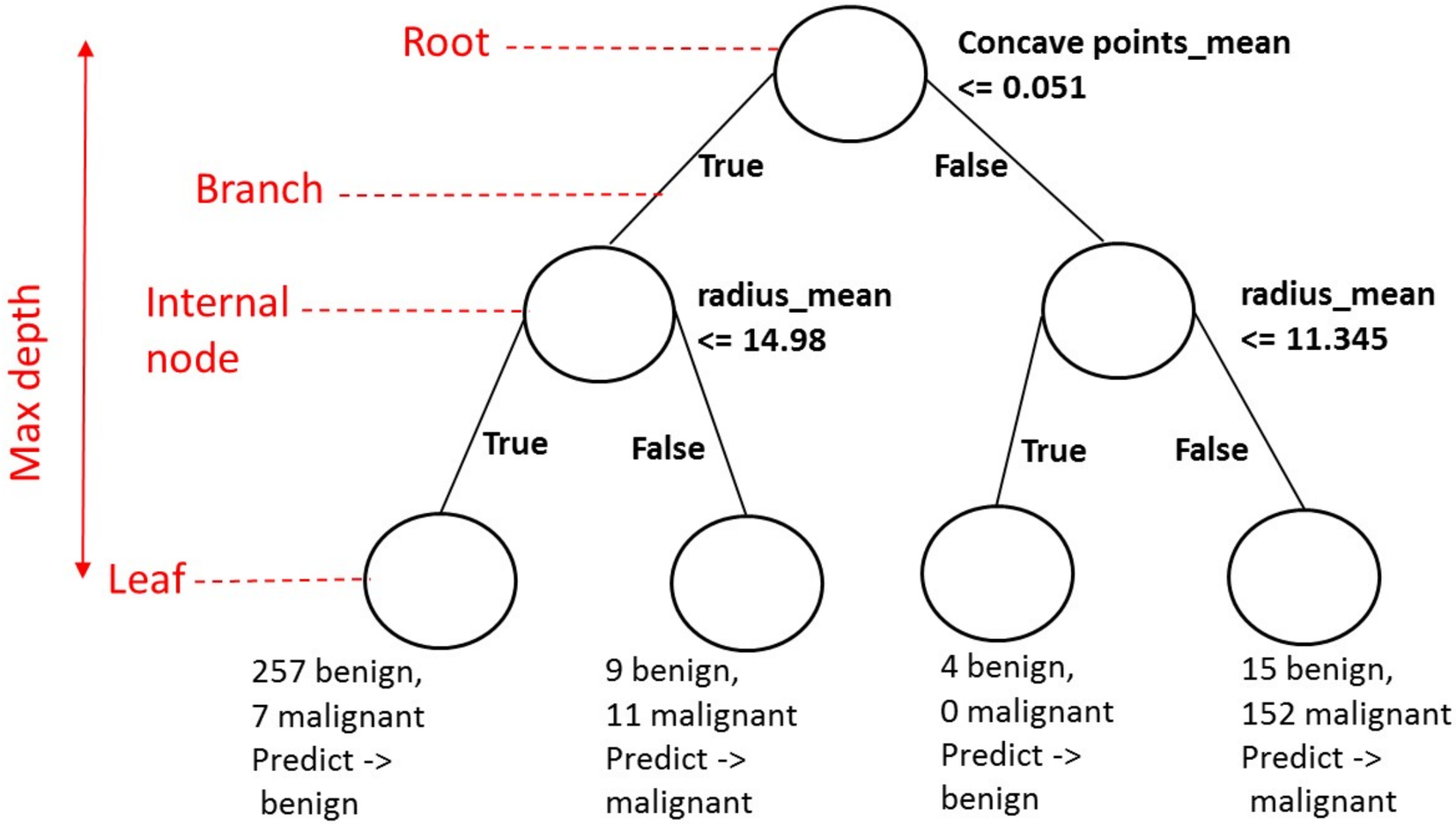
- **Decision-Tree**: data structure consisting of a hierarchy of nodes.
- **Node**: question or prediction.

Building Blocks of a Decision-Tree

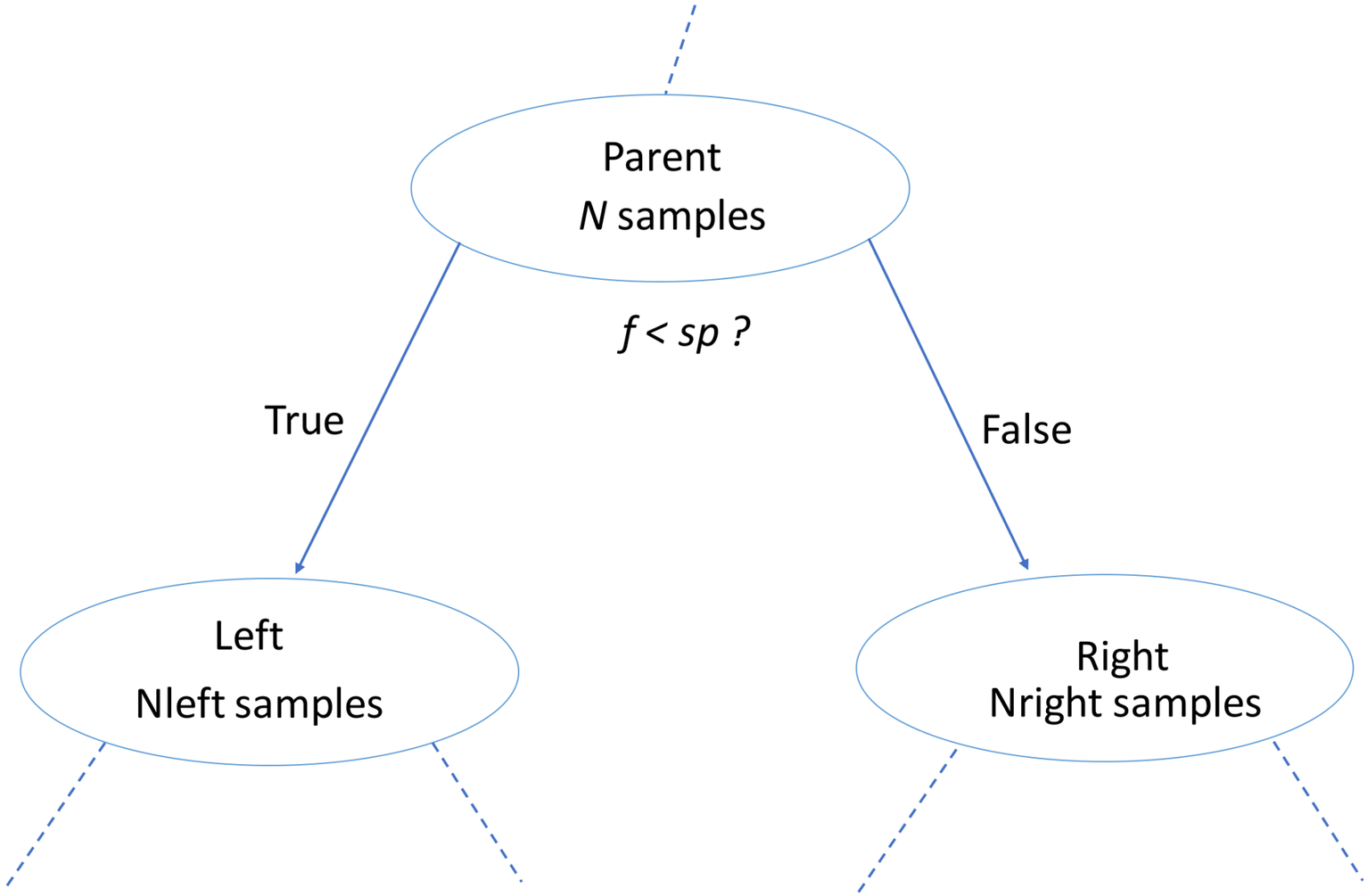
Three kinds of nodes:

- **Root**: *no* parent node, question giving rise to *two* children nodes.
- **Internal node**: *one* parent node, question giving rise to *two* children nodes.
- **Leaf**: *one* parent node, *no* children nodes --> *prediction*.

Prediction


Root
Concave points_mean
<= 0.051
True
False
Branch
Internal
node
radius_mean
<= 14.98
radius_mean
<= 11.345
True
False
True
False
Max depth
Leaf
257 benign,
7 malignant
Predict ->
benign
9 benign,
11 malignant
Predict ->
malignant
4 benign,
0 malignant
Predict ->
benign
15 benign,
152 malignant
Predict ->
malignant

Information Gain (IG)



Information Gain (IG)

$$IG(\underbrace{f}_{feature}, \underbrace{sp}_{split-point}) = I(parent) - \left(\frac{N_{left}}{N} I(left) + \frac{N_{right}}{N} I(right) \right)$$

Criteria to measure the impurity of a node $I(node)$:

- gini index,
- entropy. ...

Classification-Tree Learning

- Nodes are grown recursively.
- At each node, split the data based on:
  - feature f and split-point sp to maximize $IG(\text{node})$.
  - If $IG(\text{node})$= 0, declare the node a leaf. ...

```
# Import DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier
# Import train_test_split
from sklearn.model_selection import train_test_split
# Import accuracy_score
from sklearn.metrics import accuracy_score
# Split dataset into 80% train, 20% test
X_train, X_test, y_train, y_test= train_test_split(X, y,
                                                   test_size=0.2,
                                                   stratify=y,
                                                   random_state=1)

# Instantiate dt, set 'criterion' to 'gini'
dt = DecisionTreeClassifier(criterion='gini', random_state=1)
```

Information Criterion in scikit-learn

```
# Fit dt to the training set
dt.fit(X_train,y_train)

# Predict test-set labels
y_pred= dt.predict(X_test)

# Evaluate test-set accuracy
accuracy_score(y_test, y_pred)
```

```
0.9210526315789473
```

Let's practice!

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Decision-Tree for Regression

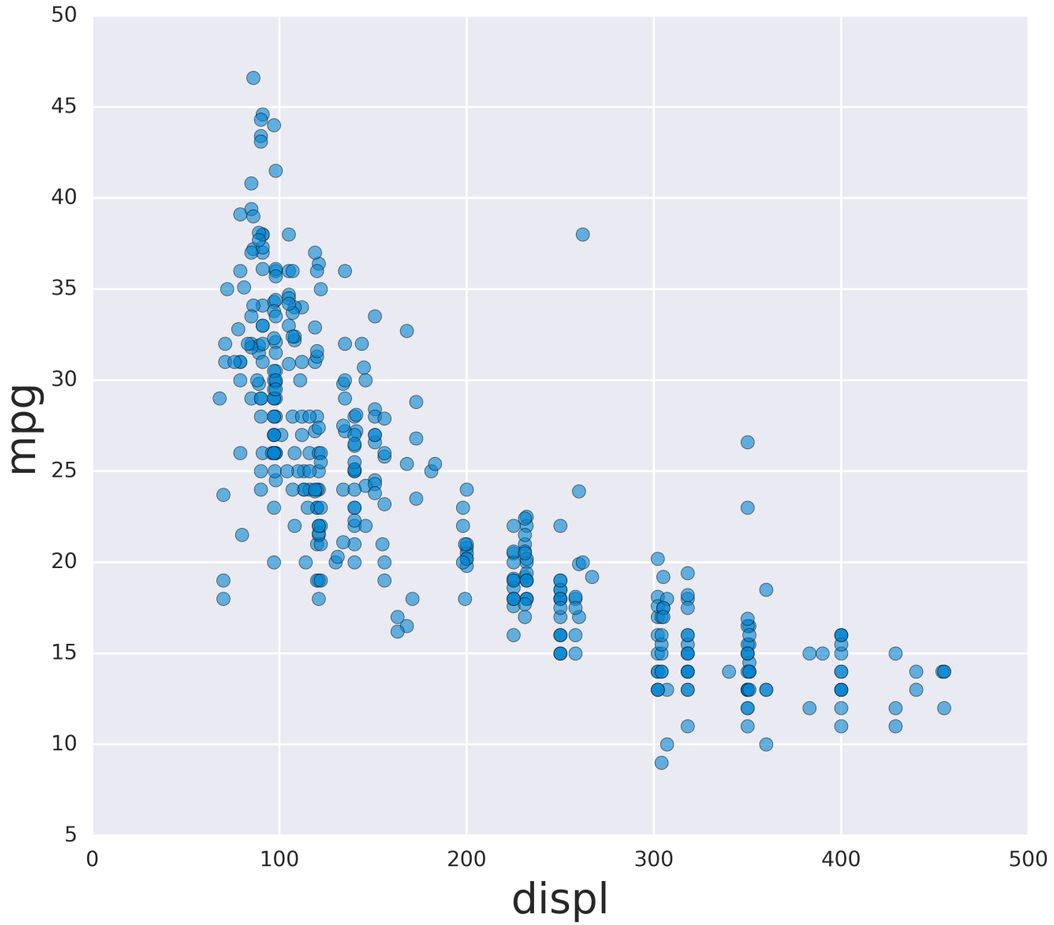
MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

Auto-mpg Dataset

	mpg	displ	hp	weight	accel	origin	size
0	18.0	250.0	88	3139	14.5	US	15.0
1	9.0	304.0	193	4732	18.5	US	20.0
2	36.1	91.0	60	1800	16.4	Asia	10.0
3	18.5	250.0	98	3525	19.0	US	15.0
4	34.3	97.0	78	2188	15.8	Europe	10.0
5	32.9	119.0	100	2615	14.8	Asia	10.0

Auto-mpg with one feature



Regression-Tree in scikit-learn

```
# Import DecisionTreeRegressor
from sklearn.tree import DecisionTreeRegressor
# Import train_test_split
from sklearn.model_selection import train_test_split
# Import mean_squared_error as MSE
from sklearn.metrics import mean_squared_error as MSE

# Split data into 80% train and 20% test
X_train, X_test, y_train, y_test= train_test_split(X, y,
                                                   test_size=0.2,
                                                   random_state=3)

# Instantiate a DecisionTreeRegressor 'dt'
dt = DecisionTreeRegressor(max_depth=4,
                           min_samples_leaf=0.1,
                           random_state=3)
```

Regression-Tree in scikit-learn

```
# Fit 'dt' to the training-set
dt.fit(X_train, y_train)
# Predict test-set labels
y_pred = dt.predict(X_test)

# Compute test-set MSE
mse_dt =  MSE(y_test, y_pred)

# Compute test-set RMSE
rmse_dt = mse_dt**(1/2)

# Print rmse_dt
print(rmse_dt)
```

```
5.1023068889
```

Information Criterion for Regression-Tree

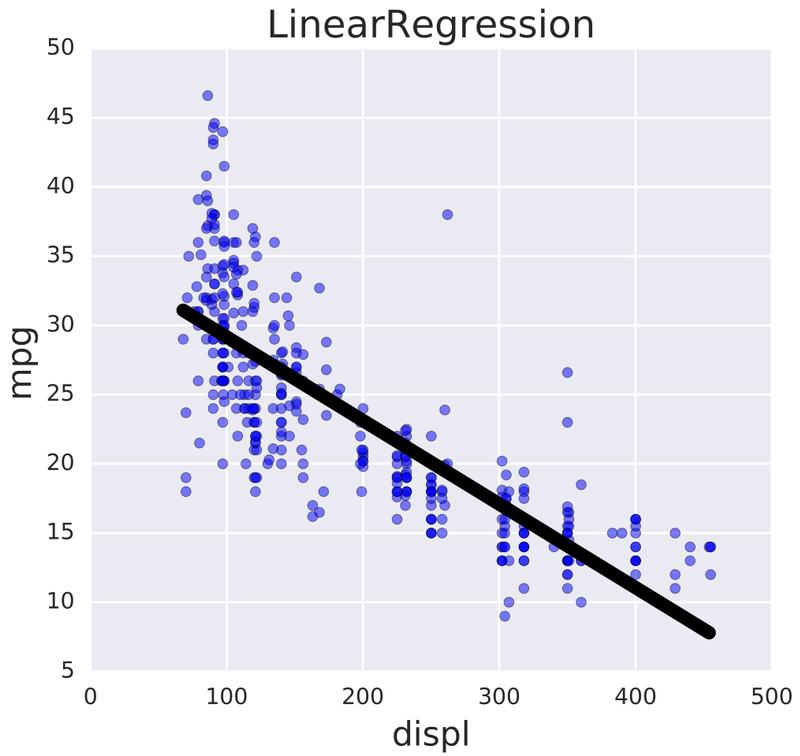
$$I(\text{node}) = \underbrace{\text{MSE}(\text{node})}_{mean-squared-error} = \frac{1}{N_{node}} \sum_{i \in node} \left(y^{(i)} - \hat{y}_{node}\right)^2$$

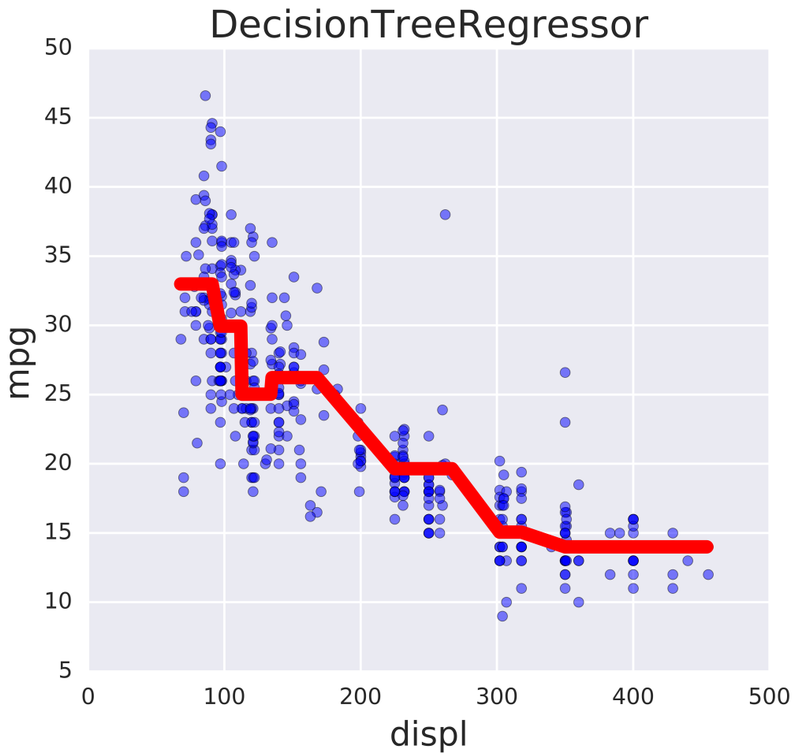
$$\underbrace{\hat{y}_{node}}_{mean-target-value} = \frac{1}{N_{node}} \sum_{i \in node} y^{(i)}$$

Prediction

$$\hat{y}_{pred}(\text{leaf}) = \frac{1}{N_{leaf}} \sum_{i \in leaf} y^{(i)}$$

Linear Regression vs. Regression-Tree





Let's practice!

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

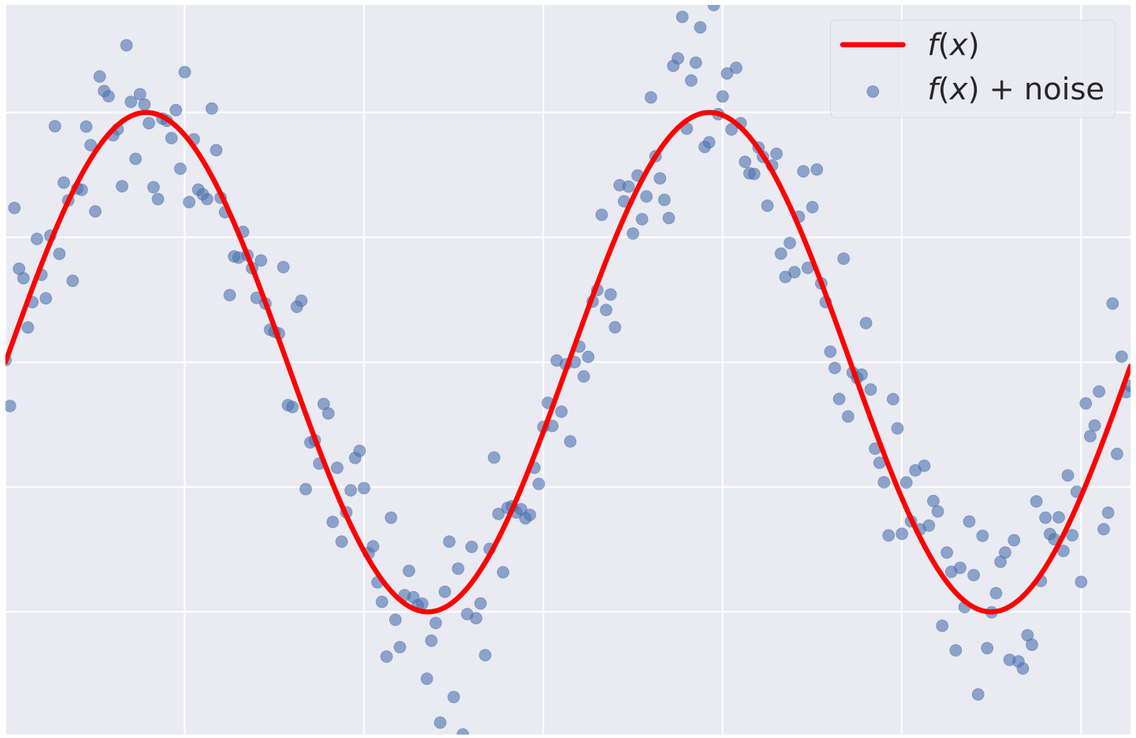
Generalization Error

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

Supervised Learning - Under the Hood

- Supervised Learning: $y = f(x)$, f is unknown.



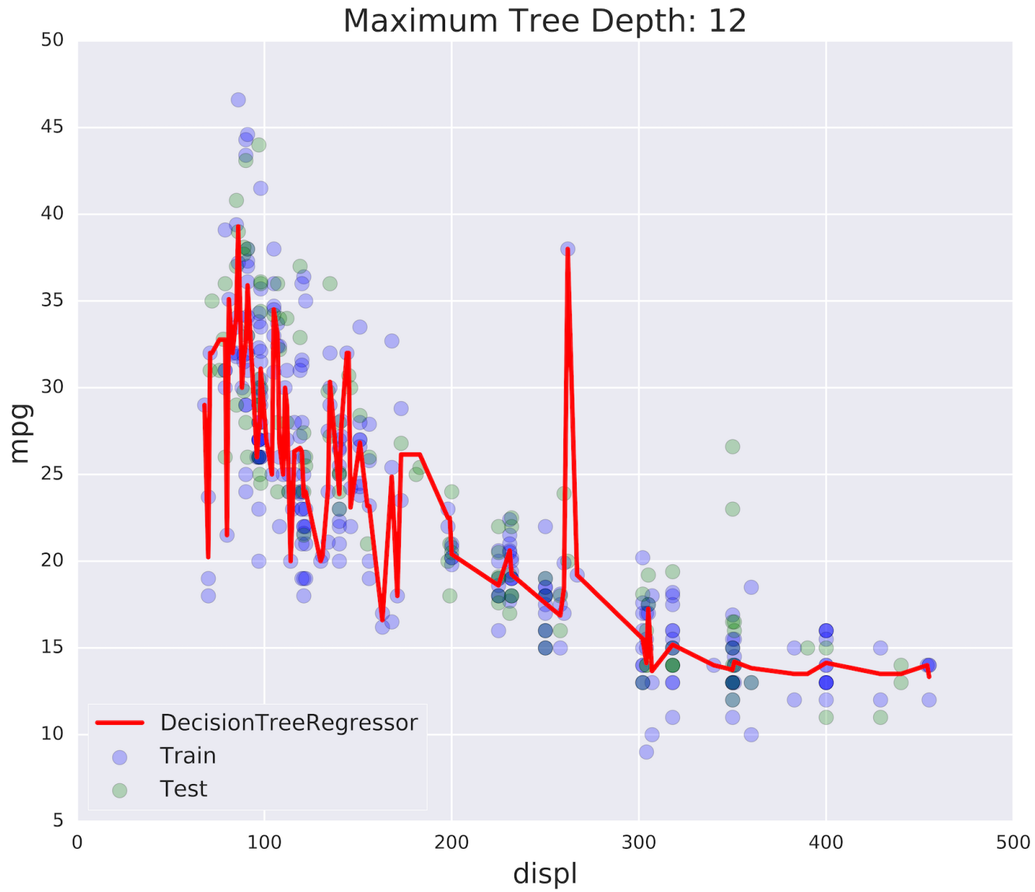
Goals of Supervised Learning

- Find a model $\hat{f}$ that best approximates f: $\hat{f} \approx f$
- $\hat{f}$ can be Logistic Regression, Decision Tree, Neural Network ...
- Discard noise as much as possible.
- **End goal**: $\hat{f}$ should acheive a low predictive error on unseen datasets.

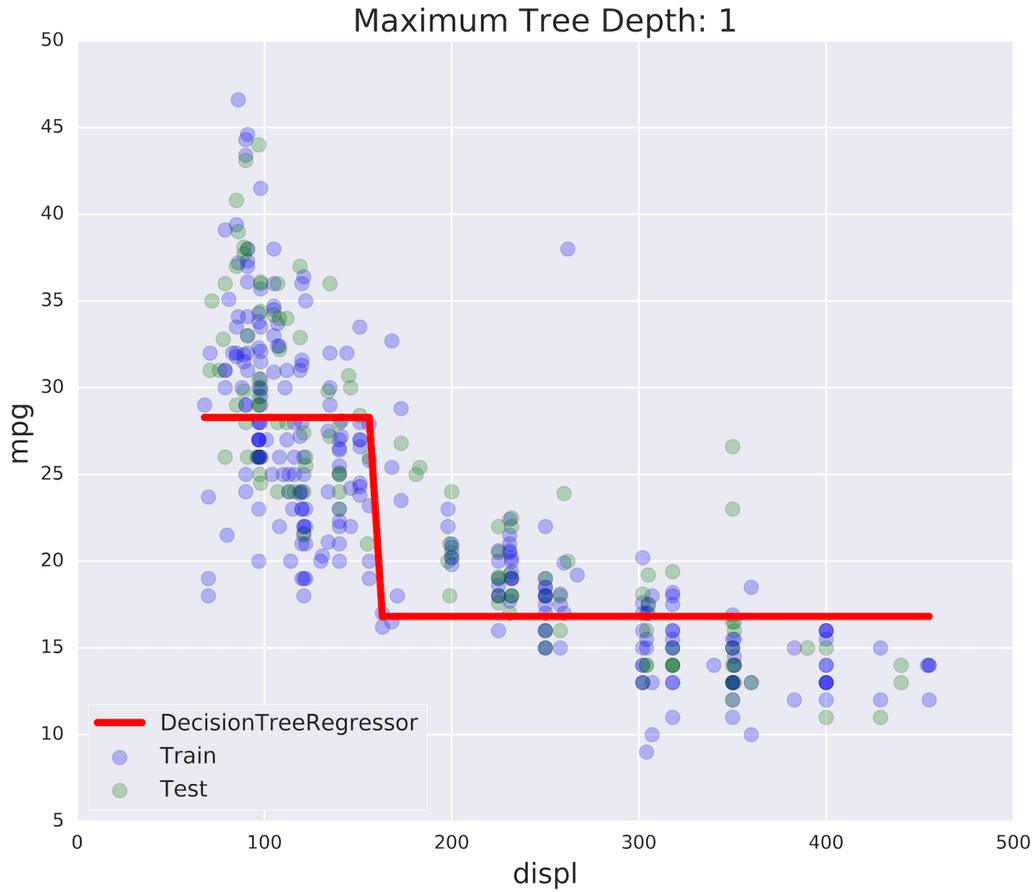
Difficulties in Approximating f

- **Overfitting**: $\hat{f}(x)$ fits the training set noise.
  - **Underfitting**: $\hat{f}$ is not flexible enough to approximate f.

Overfitting



Underfitting

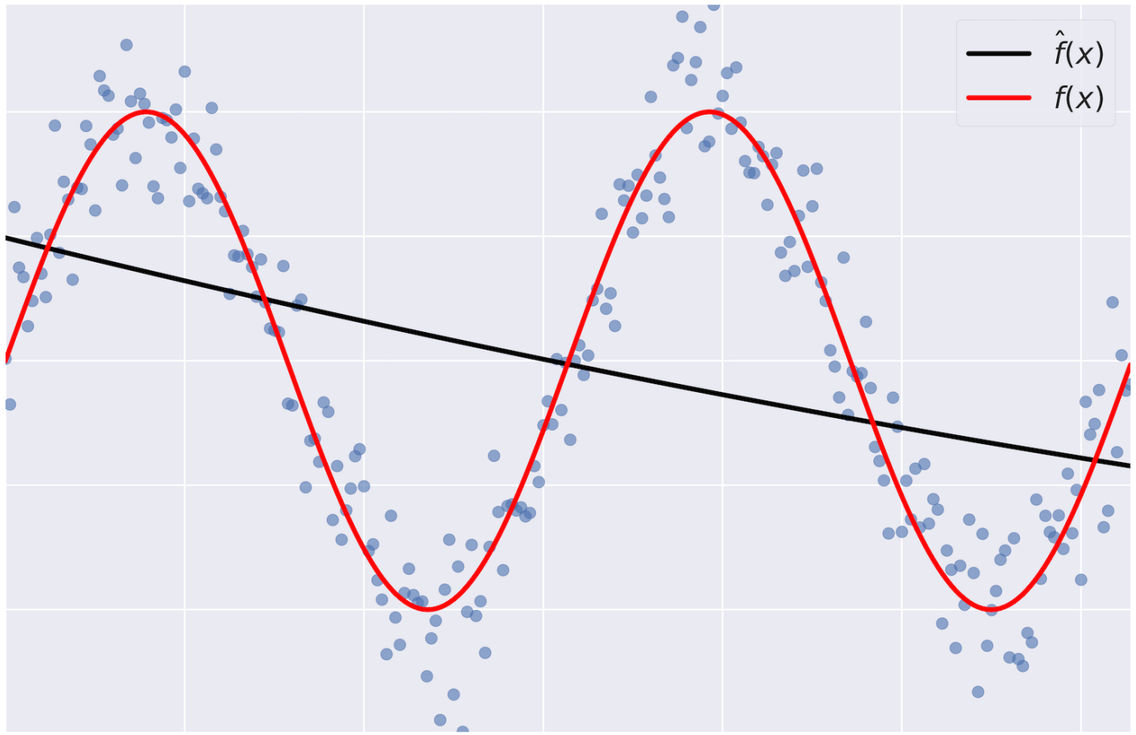


Generalization Error

- **Generalization Error of** $\hat{f}$: Does $\hat{f}$ generalize well on unseen data?
- It can be decomposed as follows: Generalization Error of $\hat{f} = bias^2 + variance + \text{irreducible error}$

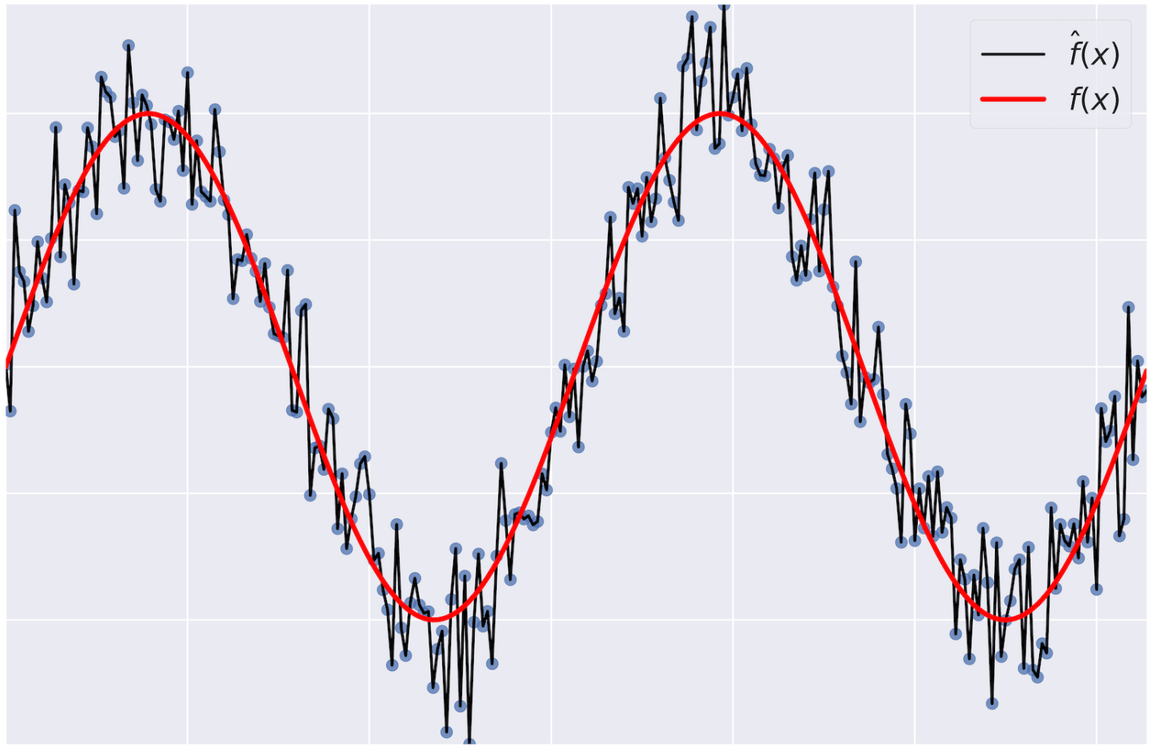
Bias

- **Bias**: error term that tells you, on average, how much $\hat{f} \neq f$.



Variance

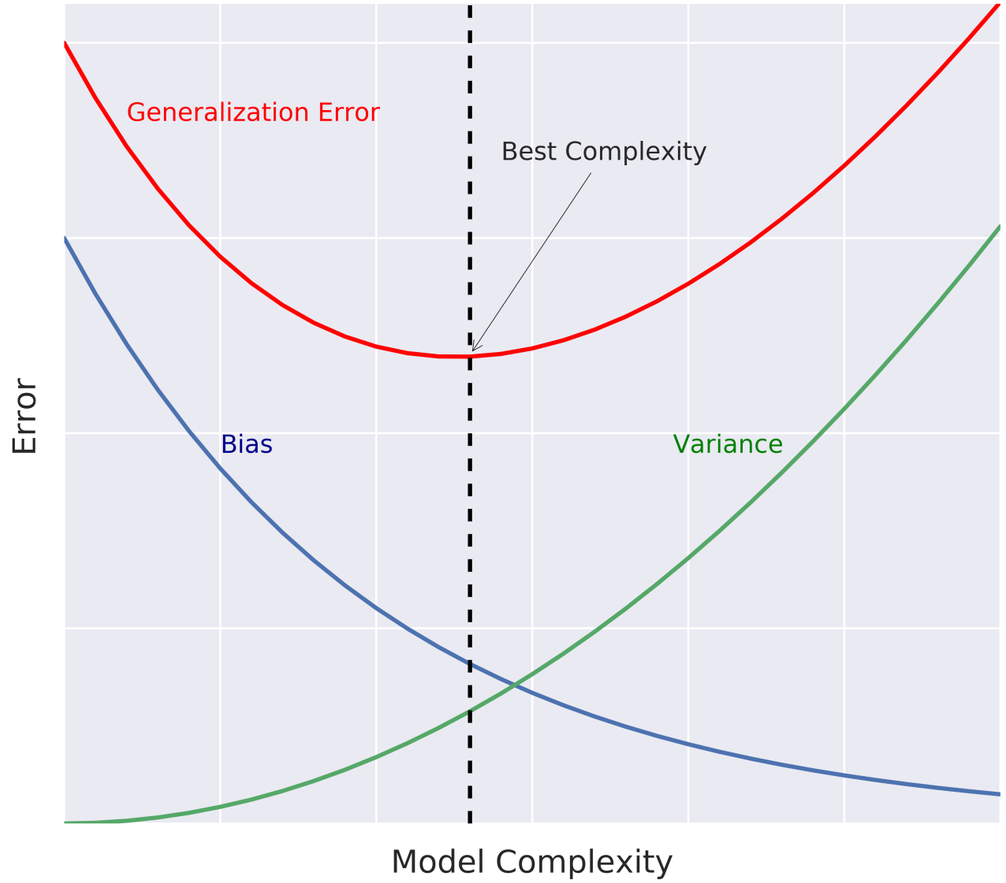
- **Variance**: tells you how much $\hat{f}$ is inconsistent over different training sets.



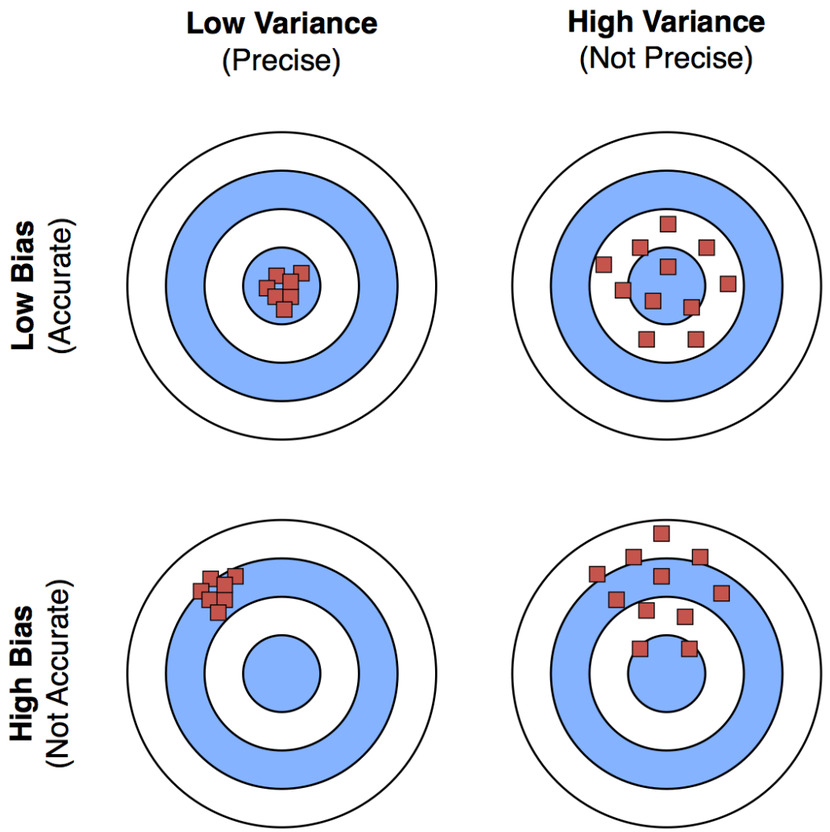
Model Complexity

- **Model Complexity**: sets the flexibility of $\hat{f}$.
- Example: Maximum tree depth, Minimum samples per leaf, ...

Bias-Variance Tradeoff



Bias-Variance Tradeoff: A Visual Explanation



Let's practice!

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Diagnosing Bias and Variance Problems

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

Estimating the Generalization Error

- How do we estimate the generalization error of a model?
- Cannot be done directly because:
  - f is unknown,
  - usually you only have one dataset,
  - noise is unpredictable.

Estimating the Generalization Error

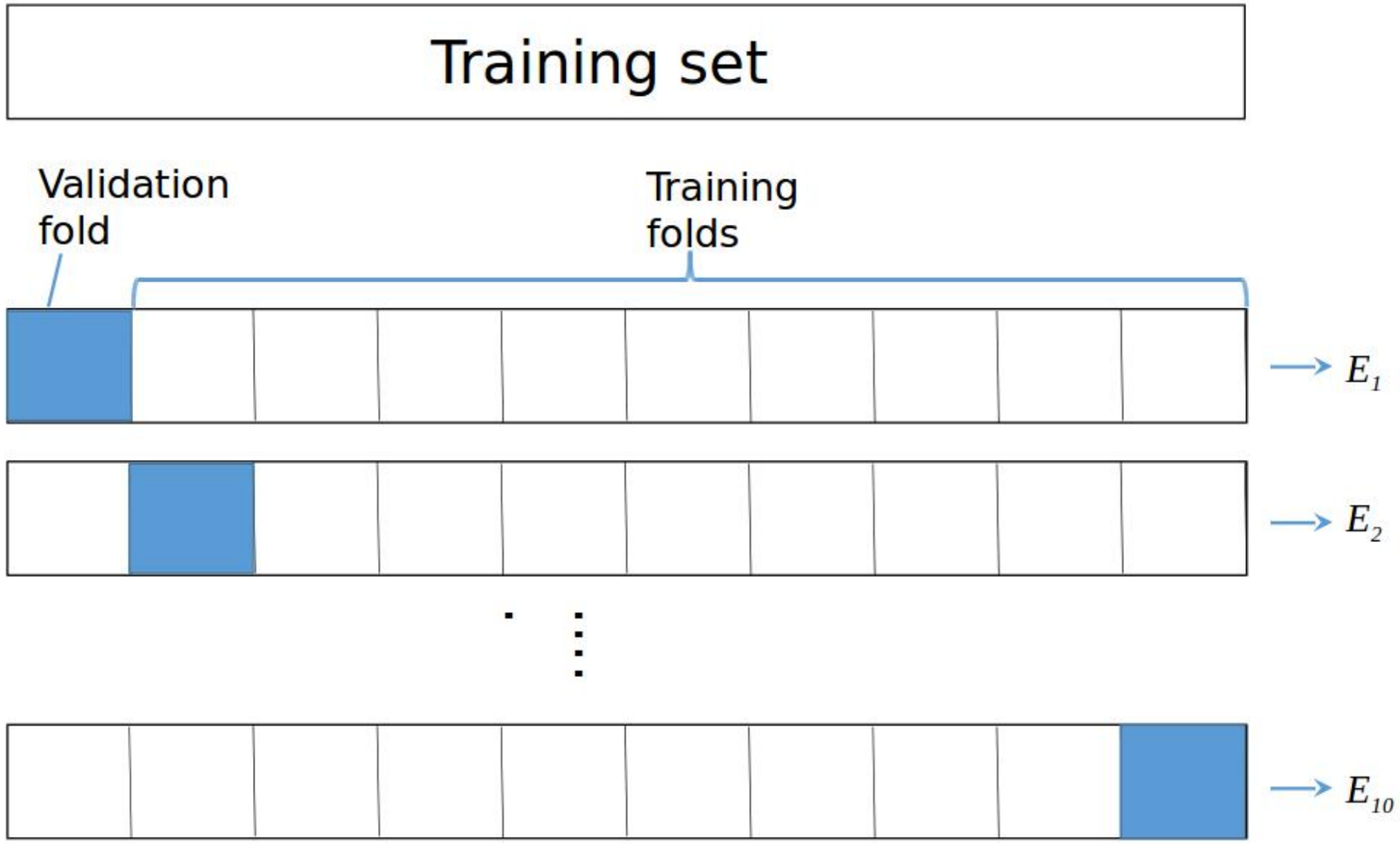
Solution:

- split the data to training and test sets,
- fit $\hat{f}$ to the training set,
- evaluate the error of $\hat{f}$ on the **unseen** test set.
- generalization error of $\hat{f} \approx$ test set error of $\hat{f}$.

Better Model Evaluation with Cross-Validation

- Test set should not be touched until we are confident about $\hat{f}$'s performance.
- Evaluating $\hat{f}$ on training set: biased estimate, $\hat{f}$ has already seen all training points.
- Solution $\longrightarrow$ Cross-Validation (CV):
  - K-Fold CV,
  - Hold-Out CV.

K-Fold CV



K-Fold CV

$$\text{CV error} = \frac{E_1 + \ldots + E_{10}}{10}$$

Diagnose Variance Problems

- If $\hat{f}$ suffers from **high variance**: CV error of $\hat{f}$ > training set error of $\hat{f}$.
  - $\hat{f}$ is said to overfit the training set. To remedy overfitting:
    - decrease model complexity,
    - for ex: decrease max depth, increase min samples per leaf, ...
    - gather more data, ..

Diagnose Bias Problems

- if $\hat{f}$ suffers from high bias: CV error of $\hat{f} \approx$ training set error of $\hat{f} >>$ desired error.
- $\hat{f}$ is said to underfit the training set. To remedy underfitting:
  - increase model complexity
  - for ex: increase max depth, decrease min samples per leaf, ...
  - gather more relevant features

K-Fold CV in sklearn on the Auto Dataset

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error as MSE
from sklearn.model_selection import cross_val_score

# Set seed for reproducibility
SEED = 123
# 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,
                                                    random_state=SEED)

# Instantiate decision tree regressor and assign it to 'dt'
dt = DecisionTreeRegressor(max_depth=4,
                           min_samples_leaf=0.14,
                           random_state=SEED)
```

K-Fold CV in sklearn on the Auto Dataset

```
# Evaluate the list of MSE ontained by 10-fold CV
# Set n_jobs to -1 in order to exploit all CPU cores in computation
MSE_CV = - cross_val_score(dt, X_train, y_train, cv= 10,
                           scoring='neg_mean_squared_error',
                           n_jobs = -1)

# Fit 'dt' to the training set
dt.fit(X_train, y_train)
# Predict the labels of training set
y_predict_train = dt.predict(X_train)
# Predict the labels of test set
y_predict_test = dt.predict(X_test)
```

```
# CV MSE
print('CV MSE: {:.2f}'.format(MSE_CV.mean()))
```

```
CV MSE: 20.51
```

```
# Training set MSE
print('Train MSE: {:.2f}'.format(MSE(y_train, y_predict_train)))
```

```
Train MSE: 15.30
```

```
# Test set MSE
print('Test MSE: {:.2f}'.format(MSE(y_test, y_predict_test)))
```

```
Test MSE: 20.92
```

Let's practice!

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Ensemble Learning

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

Advantages of CARTs

- Simple to understand.
- Simple to interpret.
- Easy to use.
- Flexibility: ability to describe non-linear dependencies.
- Preprocessing: no need to standardize or normalize features, ...

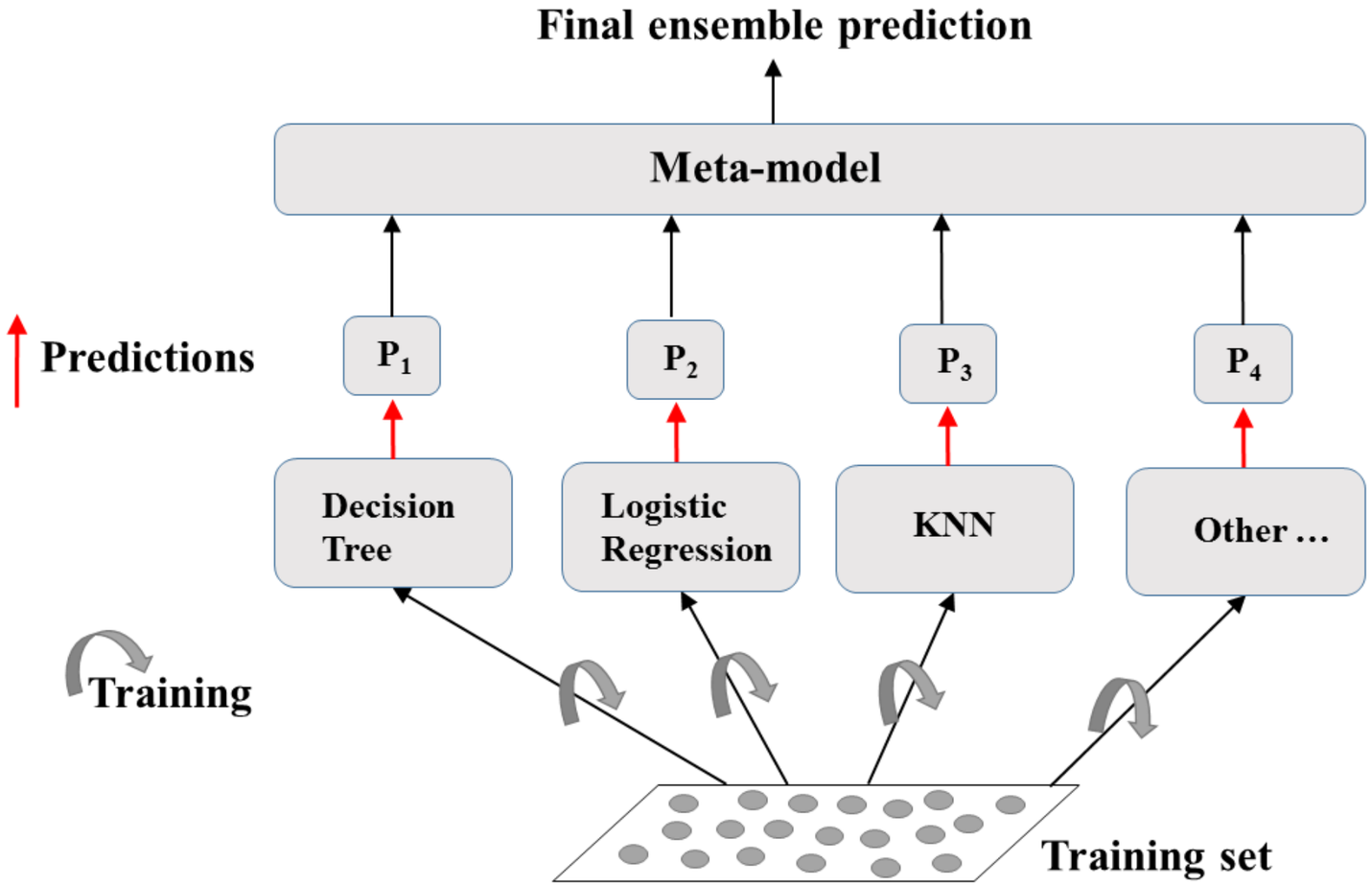
Limitations of CARTs

- Classification: can only produce orthogonal decision boundaries.
- Sensitive to small variations in the training set.
- High variance: unconstrained CARTs may overfit the training set.
- Solution: ensemble learning.

Ensemble Learning

- Train different models on the same dataset.
- Let each model make its predictions.
- Meta-model: aggregates predictions of individual models.
- Final prediction: more robust and less prone to errors.
- Best results: models are skillful in different ways.

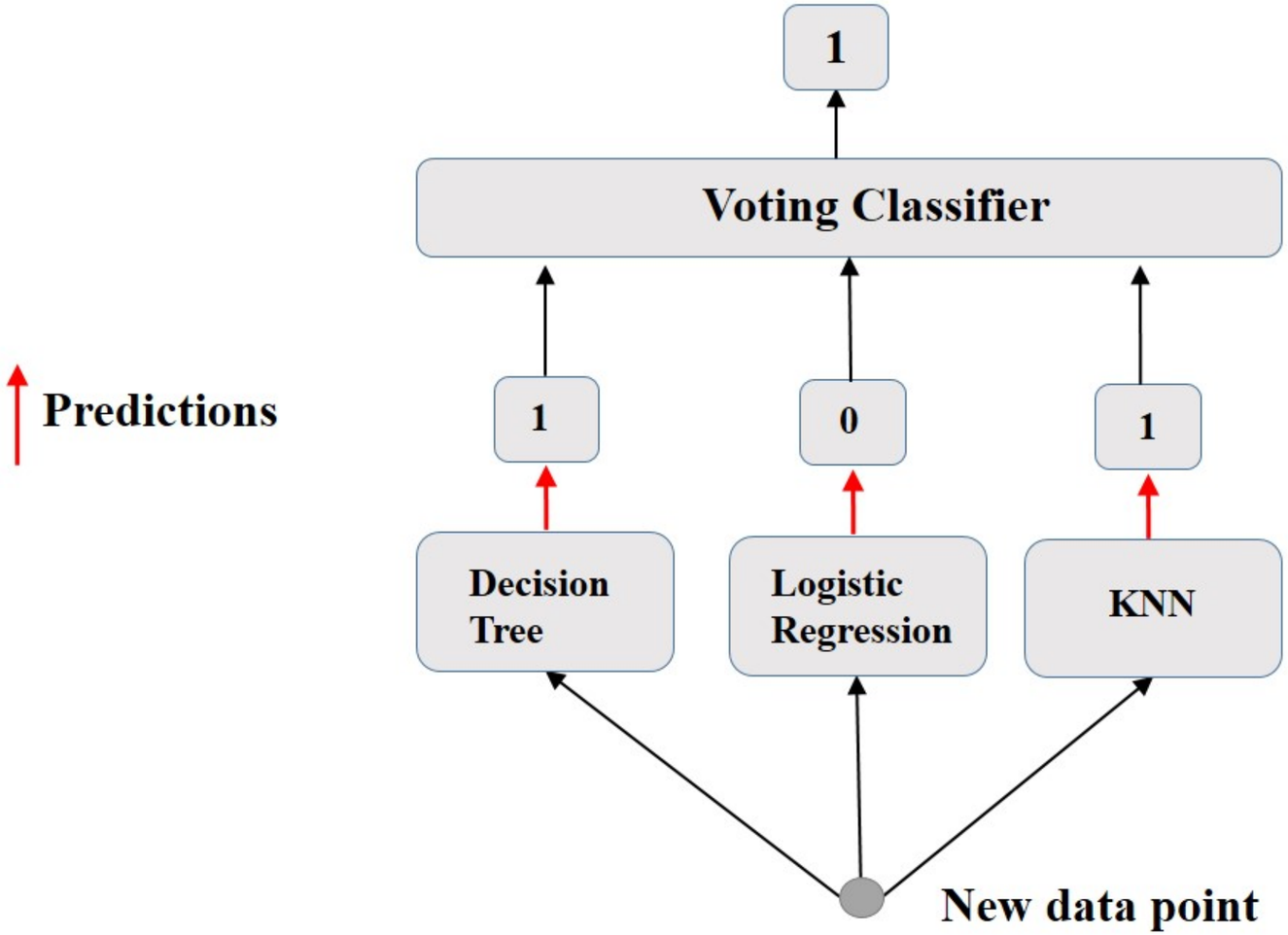
Ensemble Learning: A Visual Explanation



Ensemble Learning in Practice: Voting Classifier

- Binary classification task.
- N classifiers make predictions: $P_1, P_2, ..., P_N$ with P_i = 0 or 1.
- Meta-model prediction: hard voting.

Hard Voting


Predictions
1
Voting Classifier
1
0
1
Decision Tree
Logistic Regression
KNN
New data point

Voting Classifier in sklearn (Breast-Cancer dataset)

```
# Import functions to compute accuracy and split data
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# Import models, including VotingClassifier meta-model
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.ensemble import VotingClassifier

# Set seed for reproducibility
SEED = 1
```

Voting Classifier in sklearn (Breast-Cancer dataset)

```
# 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,
                                                    random_state= SEED)

# Instantiate individual classifiers
lr = LogisticRegression(random_state=SEED)
knn = KNN()
dt = DecisionTreeClassifier(random_state=SEED)

# Define a list called classifier that contains the tuples (classifier_name, classifier)
classifiers = [('Logistic Regression', lr),
               ('K Nearest Neighbours', knn),
               ('Classification Tree', dt)]
```

```
# Iterate over the defined list of tuples containing the classifiers
for clf_name, clf in classifiers:
    #fit clf to the training set
    clf.fit(X_train, y_train)

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

    # Evaluate the accuracy of clf on the test set
    print('{:s} : {:.3f}'.format(clf_name, accuracy_score(y_test, y_pred)))
```

```
Logistic Regression: 0.947
K Nearest Neighbours: 0.930
Classification Tree: 0.930
```

Voting Classifier in sklearn (Breast-Cancer dataset)

```
# Instantiate a VotingClassifier 'vc'
vc = VotingClassifier(estimators=classifiers)

# Fit 'vc' to the traing set and predict test set labels
vc.fit(X_train, y_train)
y_pred = vc.predict(X_test)

# Evaluate the test-set accuracy of 'vc'
print('Voting Classifier: {.3f}'.format(accuracy_score(y_test, y_pred)))
```

```
Voting Classifier: 0.953
```

Let's practice!

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Bagging

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

Ensemble Methods

Voting Classifier

- same training set,
- $\neq$ algorithms.

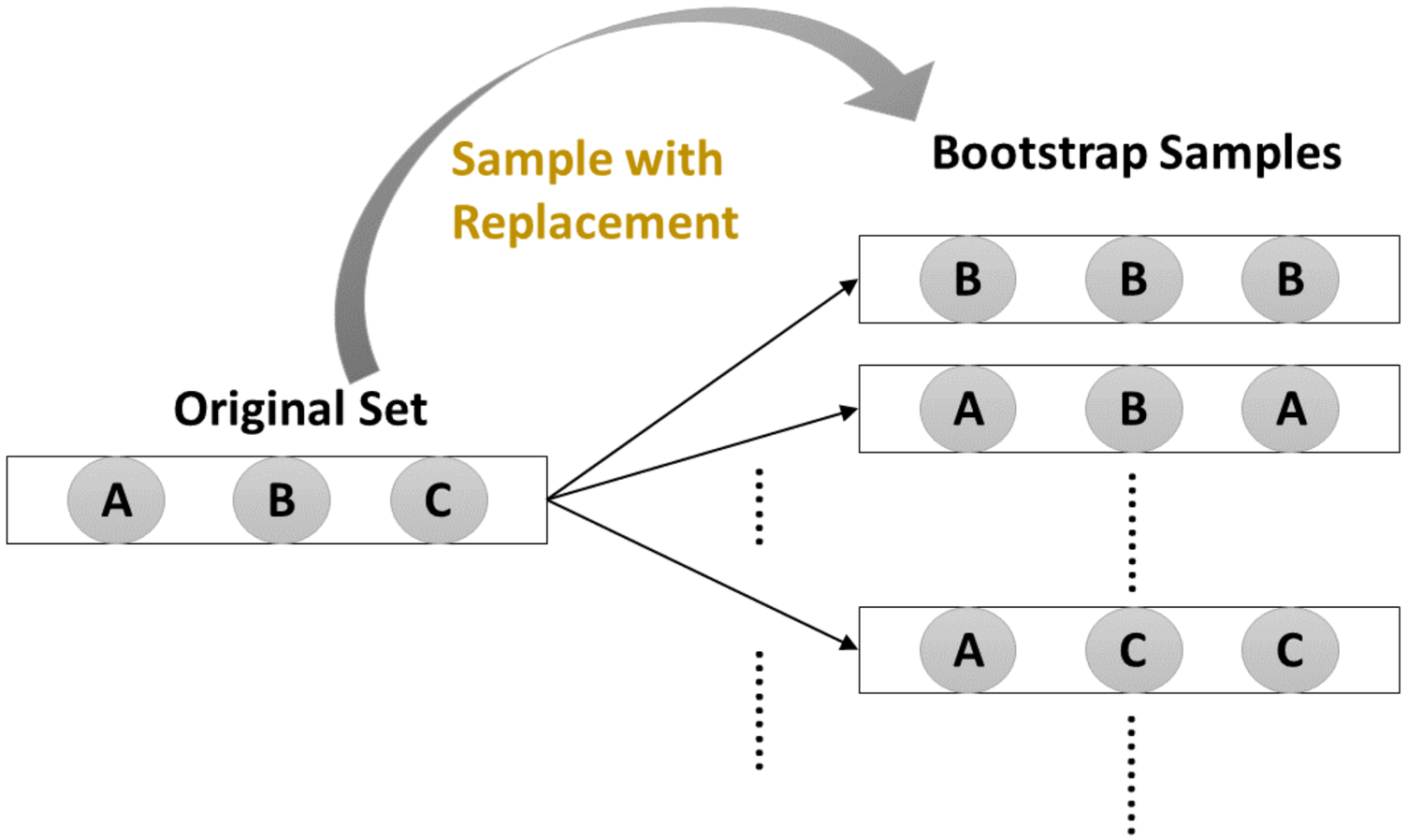
Bagging

- one algorithm,
- $\neq$ subsets of the training set.

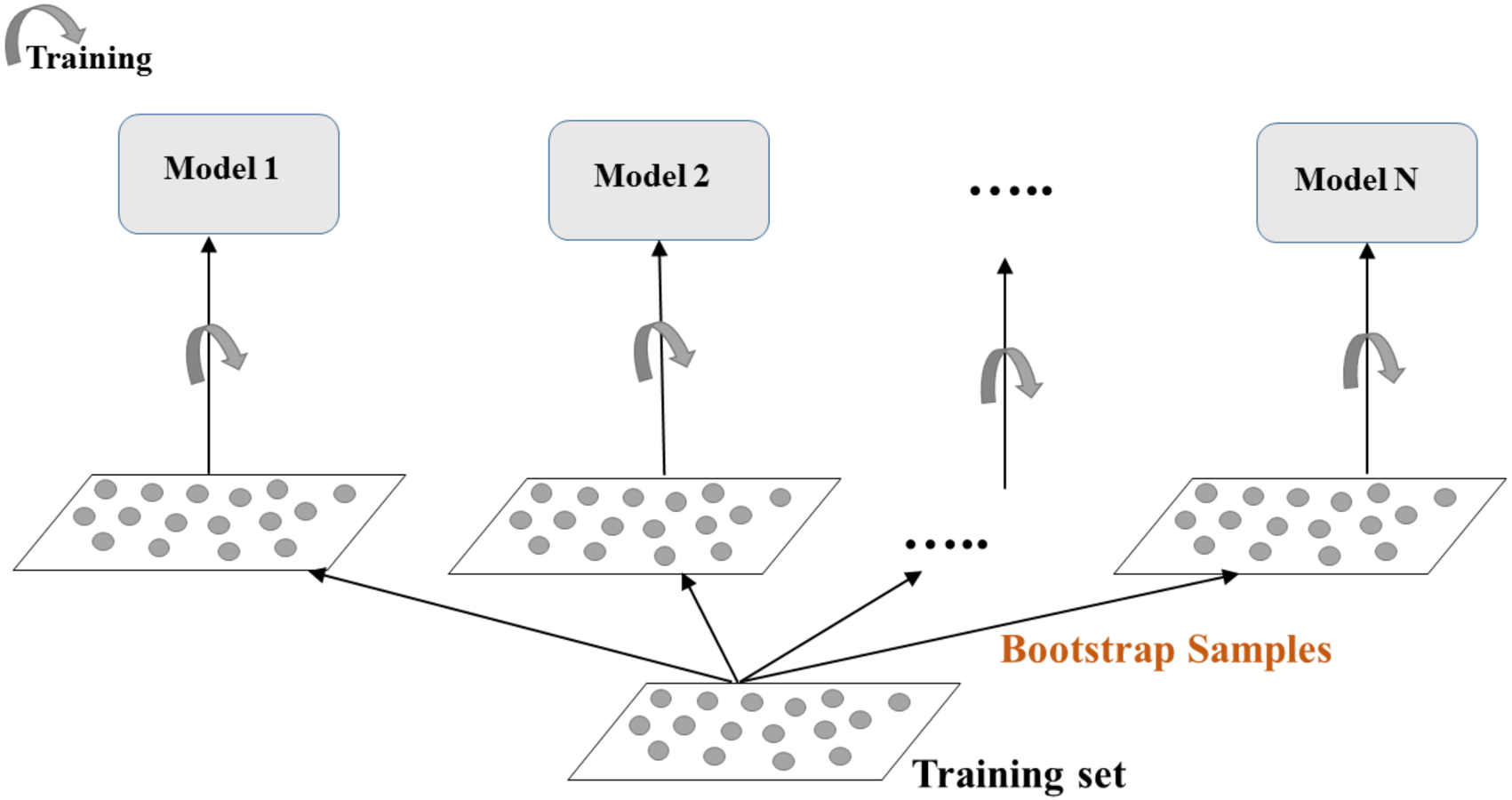
Bagging

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

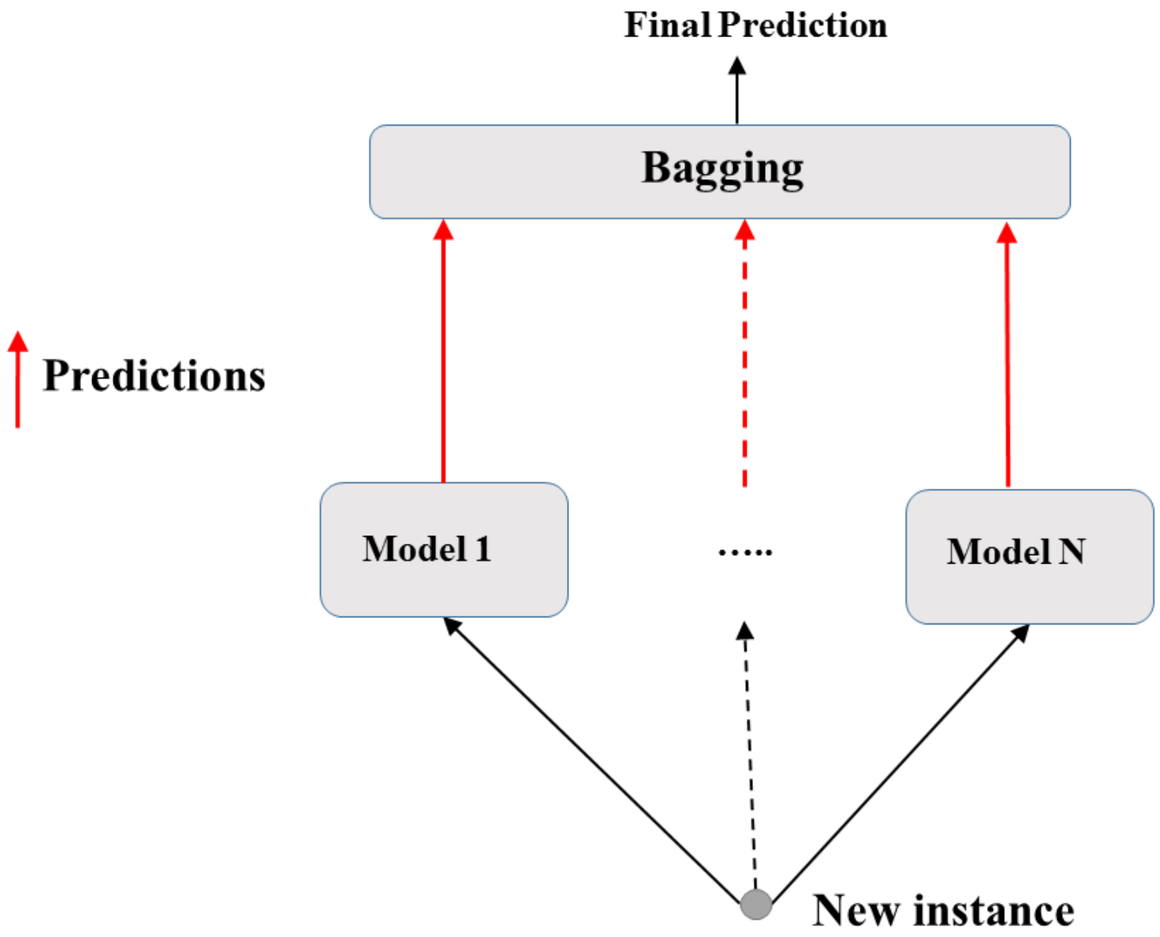
Bootstrap


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)

```
# 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)
```

```
# 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
```

Let's practice!

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Out Of Bag Evaluation

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

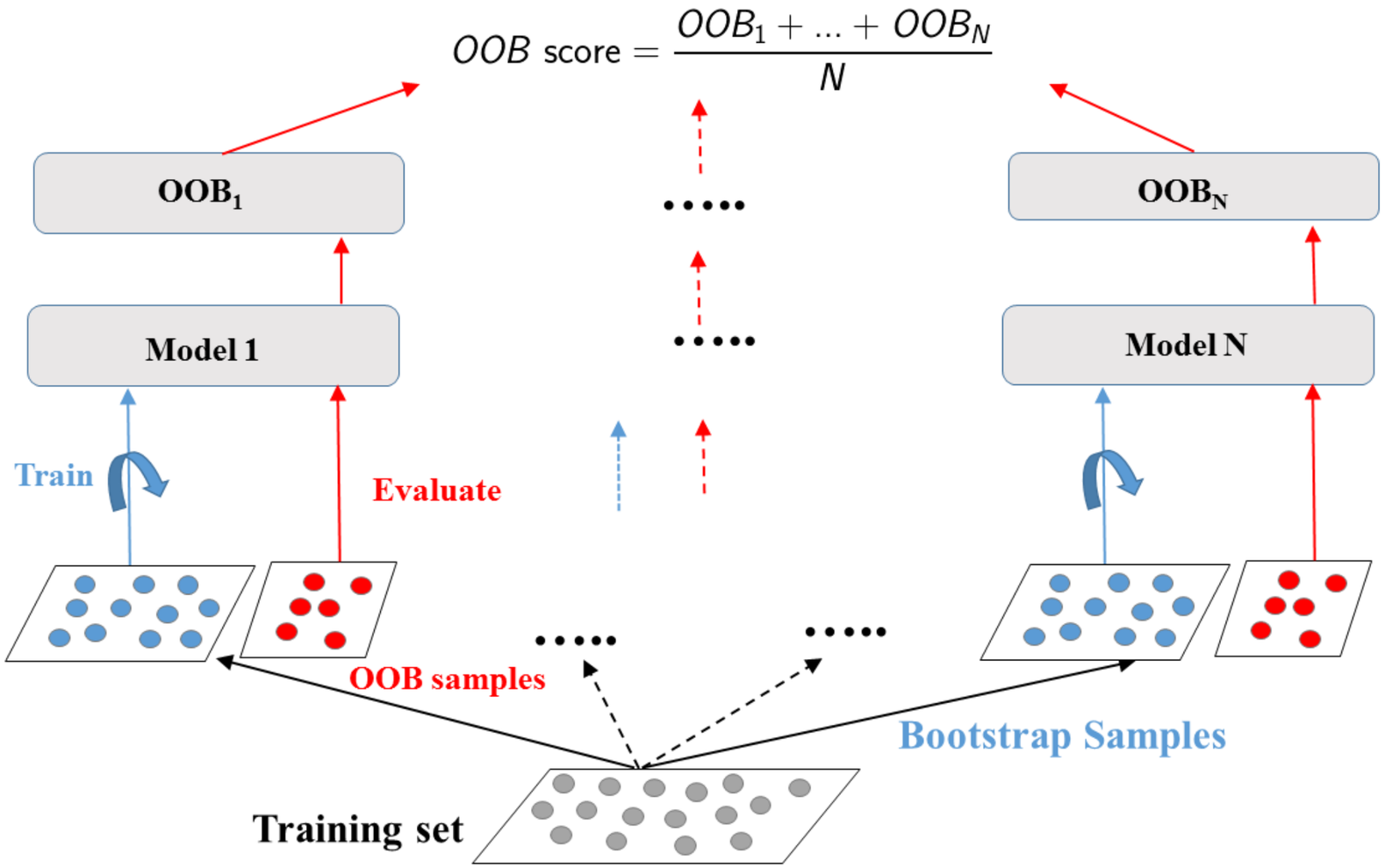
Bagging

- 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 traing set
bc.fit(X_train, y_train)

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

```
# 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
```

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

```
OOB accuracy: 0.925
```

Let's practice!

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Random Forests

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

Bagging

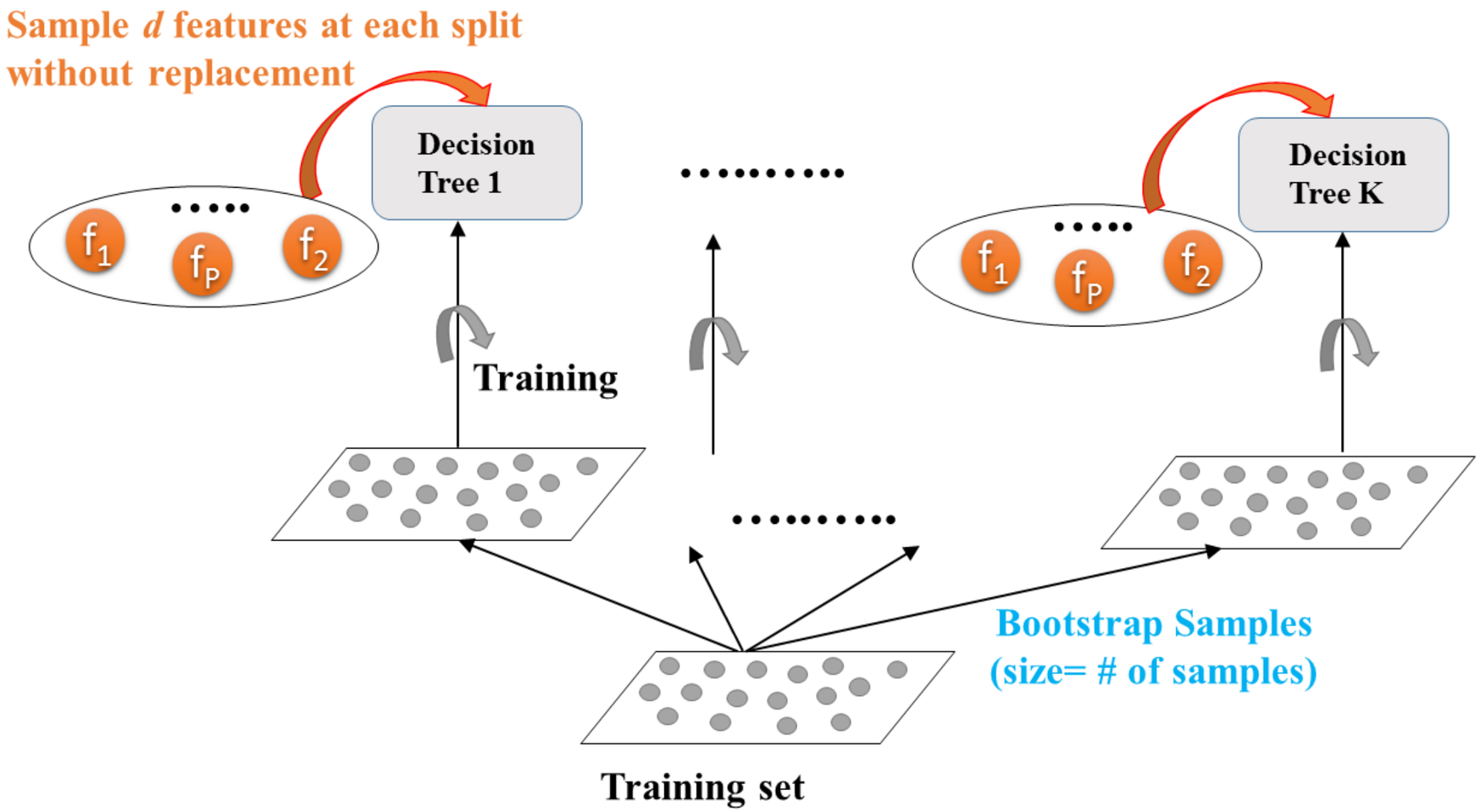
- Base estimator: Decision Tree, Logistic Regression, Neural Net, ...
- Each estimator is trained on a distinct bootstrap sample of the training set
- Estimators use all features for training and prediction

Further Diversity with Random Forests

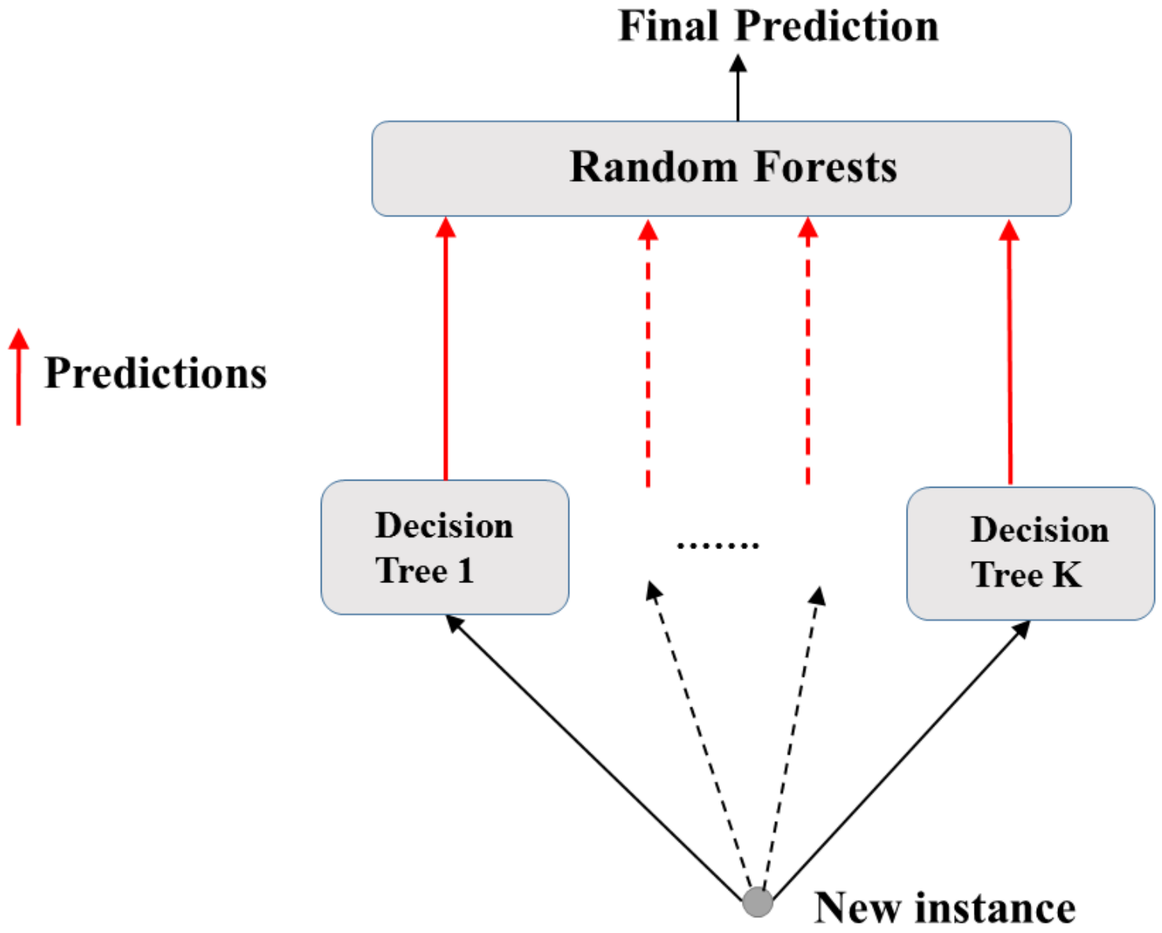
- Base estimator: Decision Tree
- Each estimator is trained on a different bootstrap sample having the same size as the training set
- RF introduces further randomization in the training of individual trees
- d features are sampled at each node without replacement

($d < \text{total number of features}$)

Random Forests: Training



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)
```

```
# 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)
```

```
# 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.

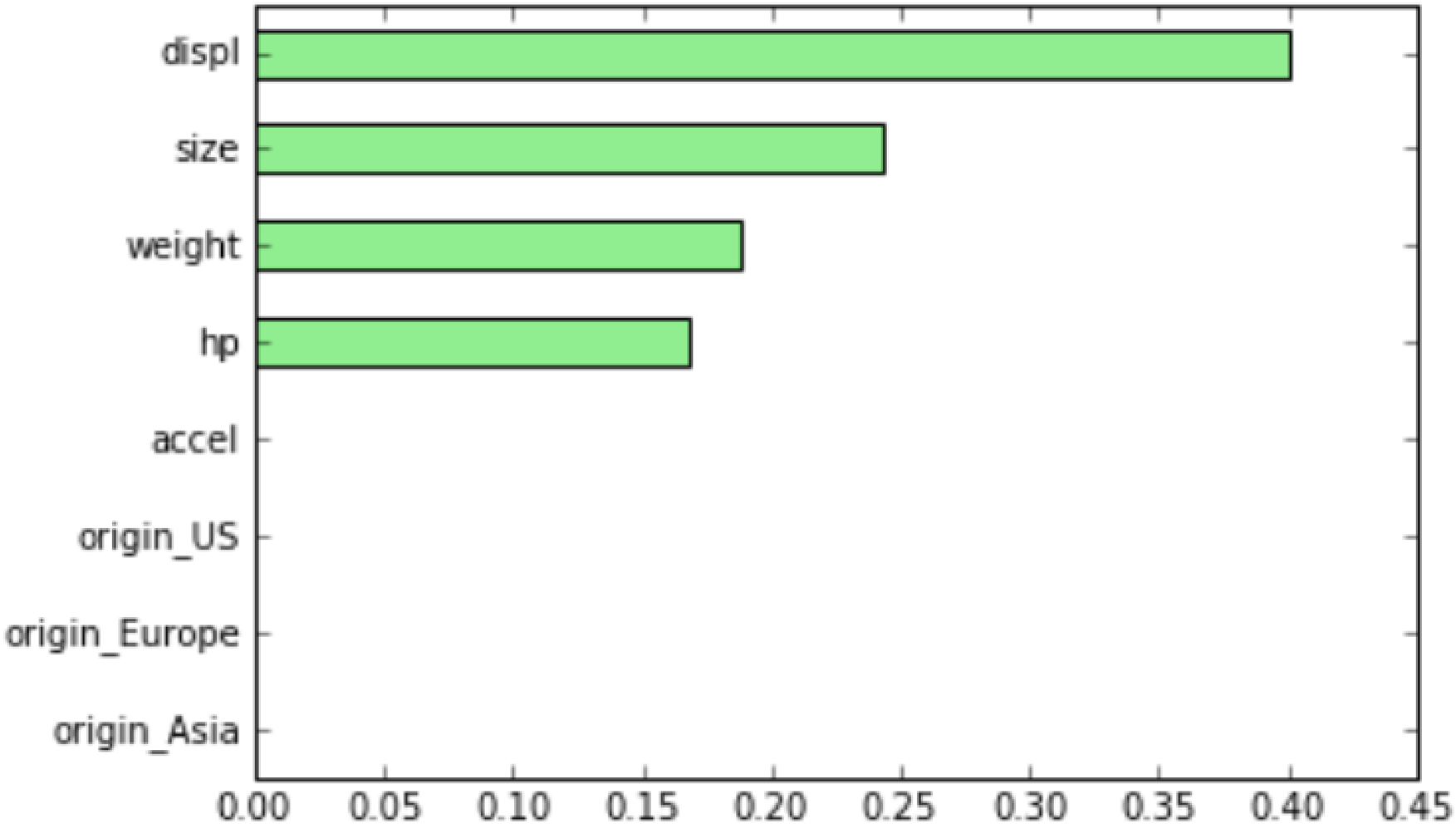
In `sklearn` :

- how much the tree nodes use a particular feature (weighted average) to reduce impurity
- accessed using the attribute `feature_importance_`

Feature Importance in sklearn

```
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



Let's practice!

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

AdaBoost

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

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).

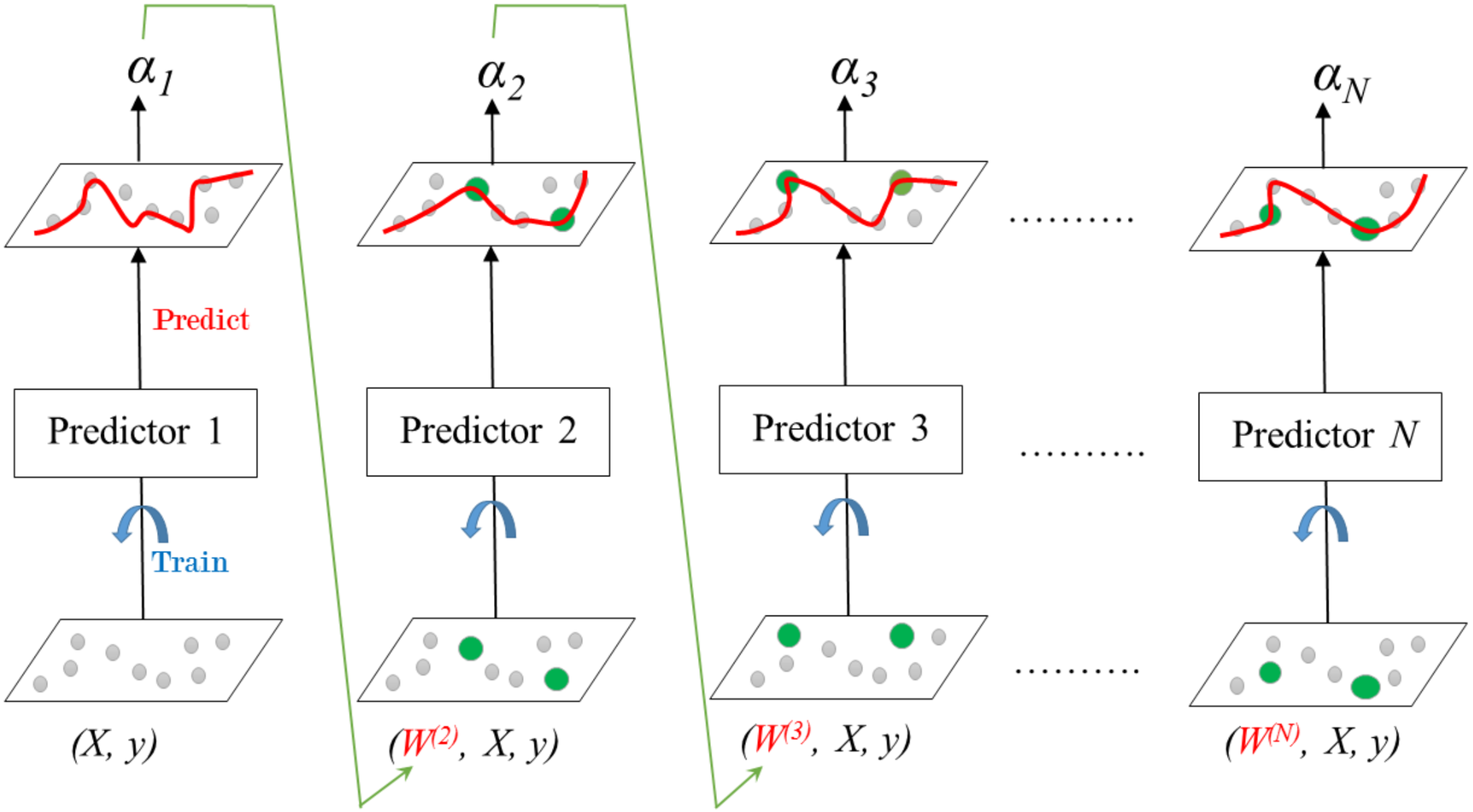
Boosting

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

Adaboost

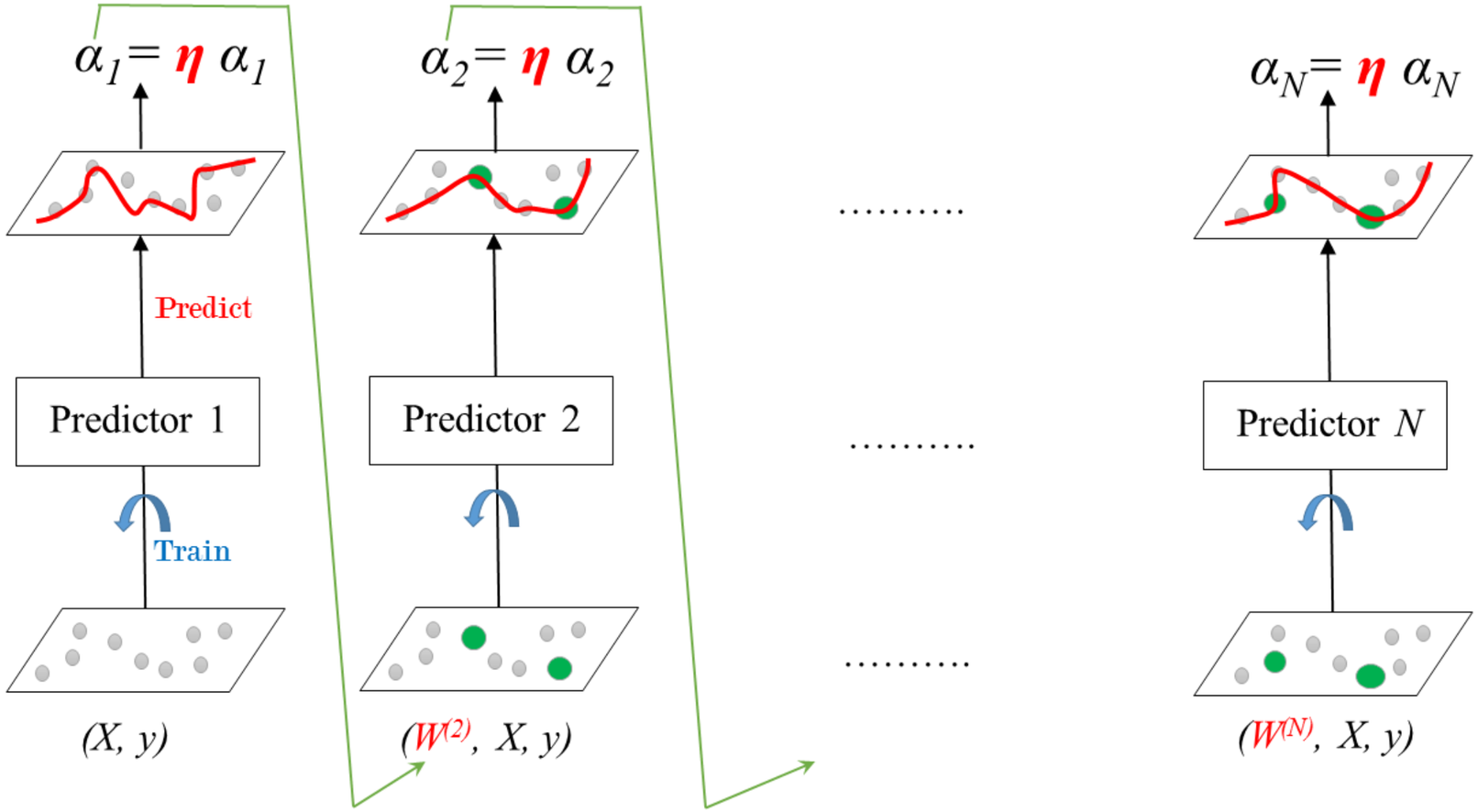
- Stands for **Ada**ptive **Boost**ing.
- Each predictor pays more attention to the instances wrongly predicted by its predecessor.
- Achieved by changing the weights of training instances.
- Each predictor is assigned a coefficient α.
- α depends on the predictor's training error.

AdaBoost: Training



Learning Rate

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



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
```

Let's practice!

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Gradient Boosting (GB)

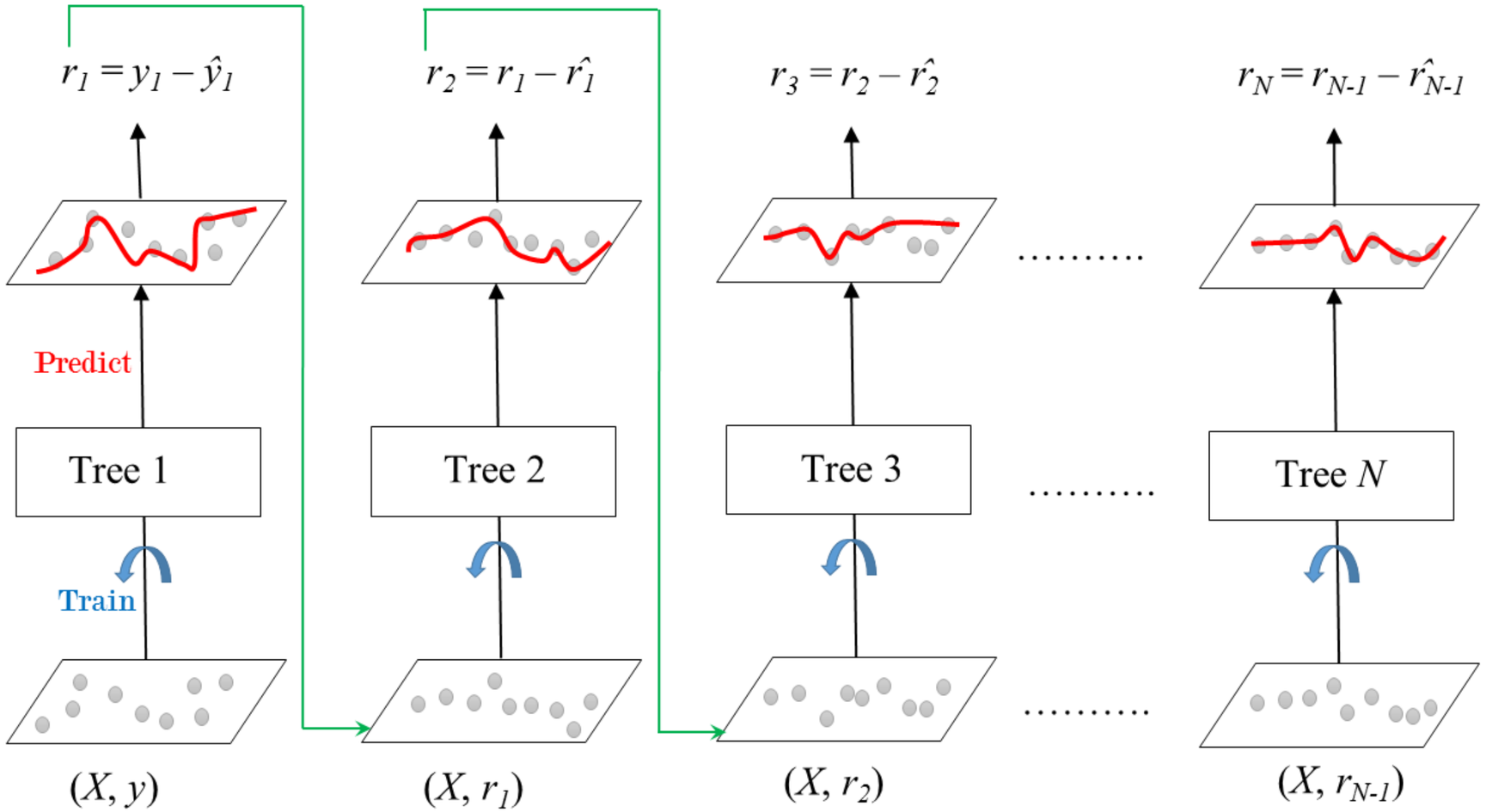
MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

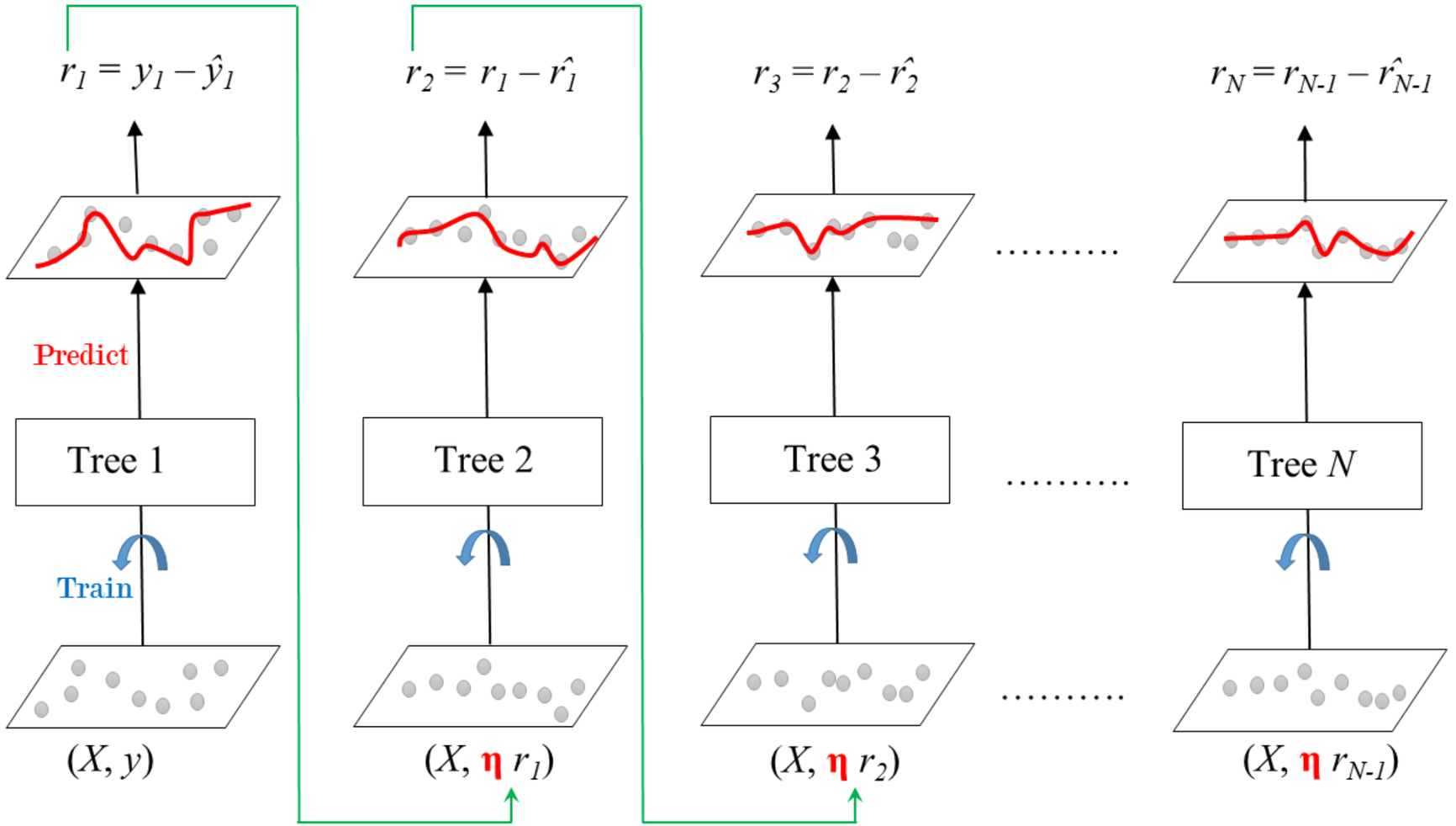
Gradient Boosted Trees

- 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 for Regression: Training



Shrinkage


$r_1 = y_1 - \hat{y}_1$
$r_2 = r_1 - \hat{r_1}$
$r_3 = r_2 - \hat{r_2}$
$r_N = r_{N-1} - \hat{r}_{N-1}$
Predict
Tree 1
Tree 2
Tree 3
Tree N
Train
(X, y)
$(X, \eta\ r_1)$
$(X, \eta\ r_2)$
$(X, \eta\ r_{N-1})$

Gradient Boosted Trees: Prediction

- Regression:
  - $y_{pred} = y_1 + \eta r_1 + ... + \eta r_N$
  - 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
```

Let's practice!

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Stochastic Gradient Boosting (SGB)

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

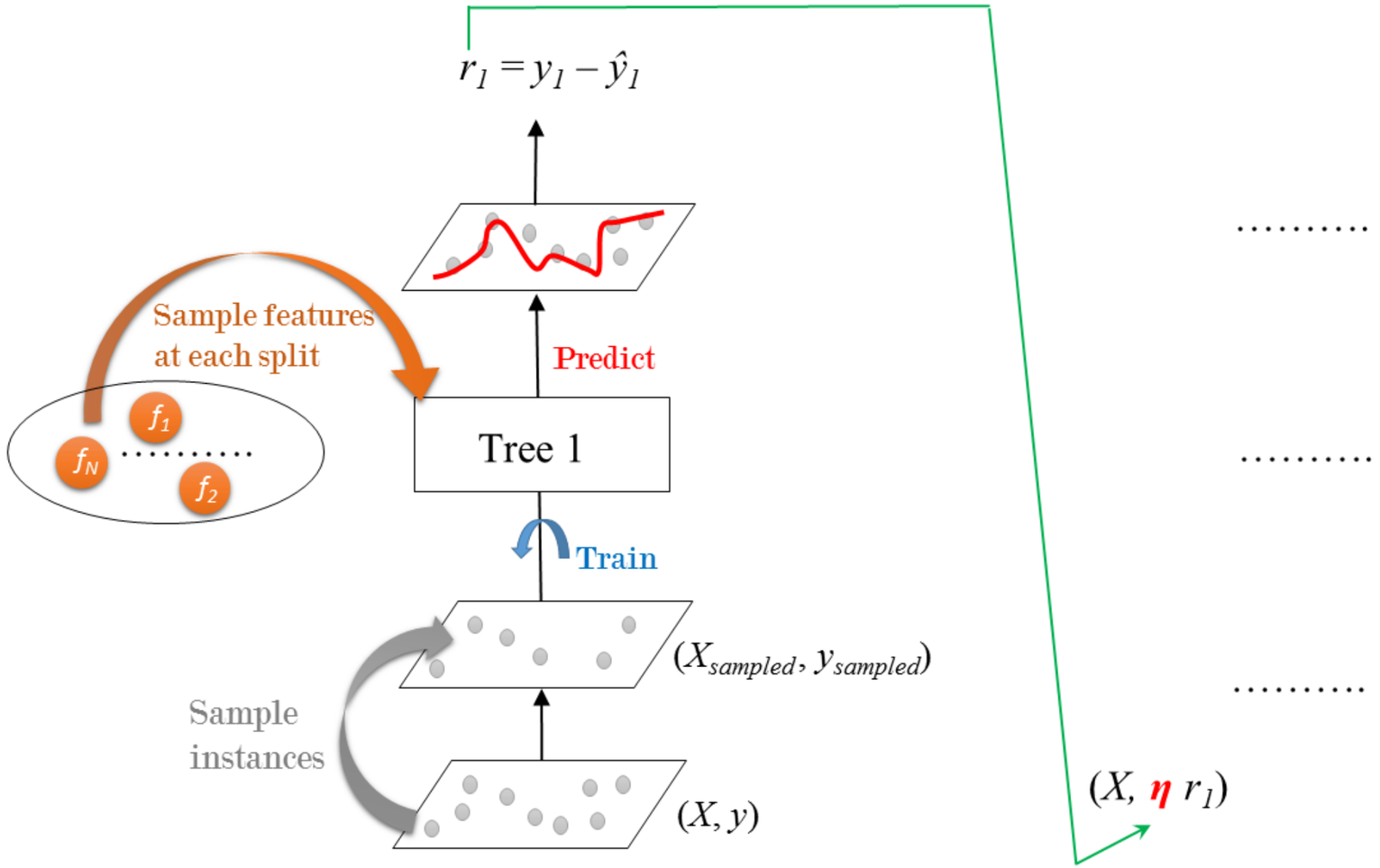
Gradient Boosting: Cons

- 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



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)
```

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
```

Let's practice!

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Tuning a CART's hyperparameters

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

Hyperparameters

Machine learning model:

- **parameters**: learned from data
  - CART example: split-point of a node, split-feature of a node, ...
- **hyperparameters**: not learned from data, set prior to training
  - CART example: `max_depth` , `min_samples_leaf` , splitting criterion ...

What is hyperparameter tuning?

- **Problem**: search for a set of optimal hyperparameters for a learning algorithm.
- **Solution**: find a set of optimal hyperparameters that results in an optimal model.
- **Optimal model**: yields an optimal **score**.
- **Score**: in sklearn defaults to accuracy (classification) and R^2 (regression).
- Cross validation is used to estimate the generalization performance.

Why tune hyperparameters?

- In `sklearn` , a model's default hyperparameters are not optimal for all problems.
- Hyperparameters should be tuned to obtain the best model performance.

Approaches to hyperparameter tuning

- Grid Search
- Random Search
- Bayesian Optimization
- Genetic Algorithms
- ....

Grid search cross validation

- Manually set a grid of discrete hyperparameter values.
- Set a metric for scoring model performance.
- Search exhaustively through the grid.
- For each set of hyperparameters, evaluate each model's CV score.
- The optimal hyperparameters are those of the model achieving the best CV score.

Grid search cross validation: example

- Hyperparameters grids:
  - `max_depth` = {2,3,4},
  - `min_samples_leaf` = {0.05, 0.1}
- hyperparameter space = { (2,0.05) , (2,0.1) , (3,0.05), ... }
- CV scores = { $score_{(2,0.05)}$, ... }
- optimal hyperparameters = set of hyperparameters corresponding to the best CV score.

Inspecting the hyperparameters of a CART in sklearn

```
# Import DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier

# Set seed to 1 for reproducibility
SEED = 1

# Instantiate a DecisionTreeClassifier 'dt'
dt = DecisionTreeClassifier(random_state=SEED)
```

Inspecting the hyperparameters of a CART in sklearn

```
# Print out 'dt's hyperparameters
print(dt.get_params())
```

```
{'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'presort': False,
 'random_state': 1,
 'splitter': 'best'}
```

```
# Import GridSearchCV
from sklearn.model_selection import GridSearchCV

# Define the grid of hyperparameters 'params_dt'
params_dt = {
             'max_depth': [3, 4,5, 6],
             'min_samples_leaf': [0.04, 0.06, 0.08],
             'max_features': [0.2, 0.4,0.6, 0.8]
            }

# Instantiate a 10-fold CV grid search object 'grid_dt'
grid_dt = GridSearchCV(estimator=dt,
                       param_grid=params_dt,
                       scoring='accuracy',
                       cv=10,
                       n_jobs=-1)

# Fit 'grid_dt' to the training data
grid_dt.fit(X_train, y_train)
```

Extracting the best hyperparameters

```
# Extract best hyperparameters from 'grid_dt'
best_hyperparams = grid_dt.best_params_
print('Best hyerparameters:\n', best_hyperparams)
```

```
Best hyerparameters:
  {'max_depth': 3, 'max_features': 0.4, 'min_samples_leaf': 0.06}
```

```
# Extract best CV score from 'grid_dt'
best_CV_score = grid_dt.best_score_
print('Best CV accuracy'.format(best_CV_score))
```

```
Best CV accuracy: 0.938
```

Extracting the best estimator

```
# Extract best model from 'grid_dt'
best_model = grid_dt.best_estimator_


# Evaluate test set accuracy
test_acc = best_model.score(X_test,y_test)

# Print test set accuracy
print("Test set accuracy of best model: {:.3f}".format(test_acc))
```

```
Test set accuracy of best model: 0.947
```

Let's practice!

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Tuning an RF's Hyperparameters

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

Random Forests Hyperparameters

- CART hyperparameters
- number of estimators
- bootstrap
- ....

Tuning is expensive

Hyperparameter tuning:

- computationally expensive,
- sometimes leads to very slight improvement,

Weight the impact of tuning on the whole project.

Inspecting RF Hyperparameters in sklearn

```
# Import RandomForestRegressor
from sklearn.ensemble import RandomForestRegressor

# Set seed for reproducibility
SEED = 1

# Instantiate a random forests regressor 'rf'
rf = RandomForestRegressor(random_state= SEED)
```

```
# Inspect rf' s hyperparameters
rf.get_params()
```

```
{'bootstrap': True,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 10,
 'n_jobs': -1,
 'oob_score': False,
 'random_state': 1,
 'verbose': 0,
 'warm_start': False}
```

```
# Basic imports
from sklearn.metrics import mean_squared_error as MSE
from sklearn.model_selection import GridSearchCV

# Define a grid of hyperparameter 'params_rf'
params_rf = {
              'n_estimators': [300, 400, 500],
              'max_depth': [4, 6, 8],
              'min_samples_leaf': [0.1, 0.2],
              'max_features': ['log2', 'sqrt']
             }

# Instantiate 'grid_rf'
grid_rf = GridSearchCV(estimator=rf,
                       param_grid=params_rf,
                       cv=3,
                       scoring='neg_mean_squared_error',
                       verbose=1,
                       n_jobs=-1)
```

Searching for the best hyperparameters

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

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   10.0s
[Parallel(n_jobs=-1)]: Done 108 out of 108 | elapsed:   24.3s finished
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=4,
           max_features='log2', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=0.1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=400, n_jobs=1,
           oob_score=False, random_state=1, verbose=0, warm_start=False)
```

Extracting the best hyperparameters

```
# Extract best hyperparameters from 'grid_rf'
best_hyperparams = grid_rf.best_params_

print('Best hyerparameters:\n', best_hyperparams)
```

```
Best hyerparameters:
        {'max_depth': 4,
         'max_features': 'log2',
         'min_samples_leaf': 0.1,
         'n_estimators': 400}
```

Evaluating the best model performance

```
# Extract best model from 'grid_rf'
best_model = grid_rf.best_estimator_
# Predict the test set labels
y_pred = best_model.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 of rf: {:.2f}'.format(rmse_test))
```

```
Test set RMSE of rf: 3.89
```

Let's practice!

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Congratulations!

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

How far you have come

- **Chapter 1:** Decision-Tree Learning
- **Chapter 2:** Generalization Error, Cross-Validation, Ensembling
- **Chapter 3:** Bagging and Random Forests
- **Chapter 4:** AdaBoost and Gradient-Boosting
- **Chapter 5:** Model Tuning

Thank you!

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON