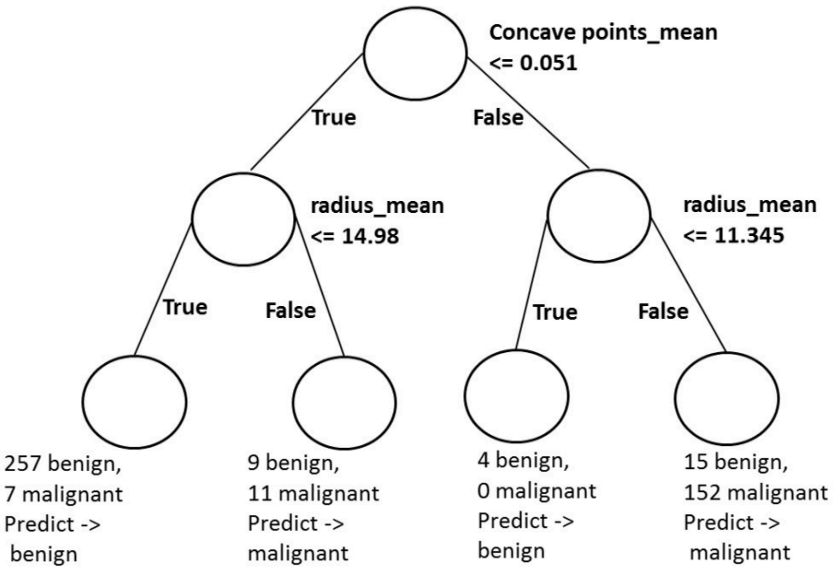
Decision tree for classification

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, ..)

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 the 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

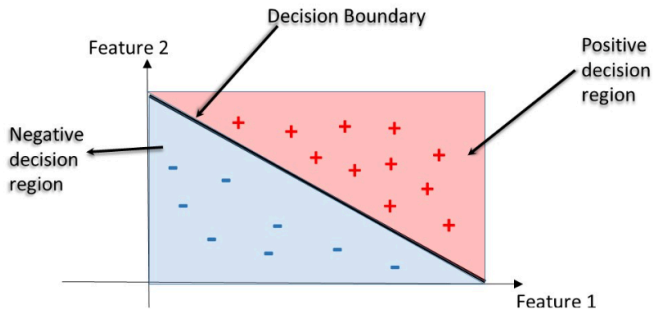
```
# Fit dt to the training set
dt.fit(X_train,y_train)

# Predict the test set labels
y_pred = dt.predict(X_test)
# Evaluate the test-set accuracy
accuracy_score(y_test, y_pred)
```

```
0.90350877192982459
```

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.



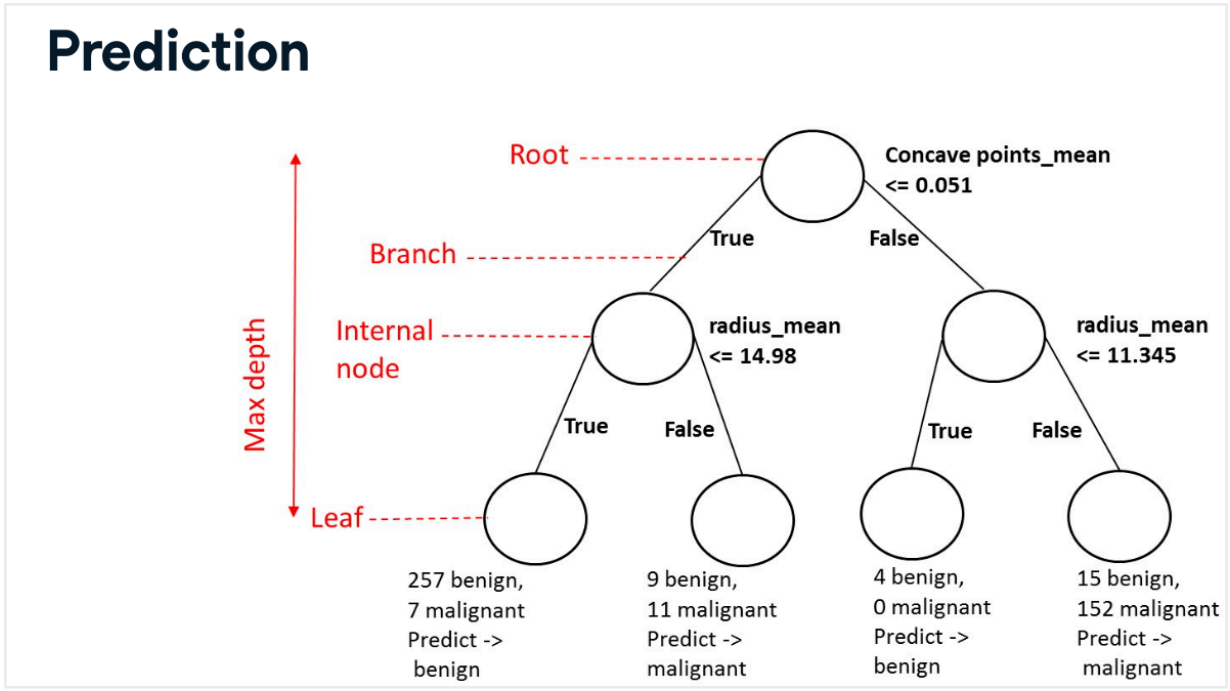
Building Blocks of a Decision-Tree

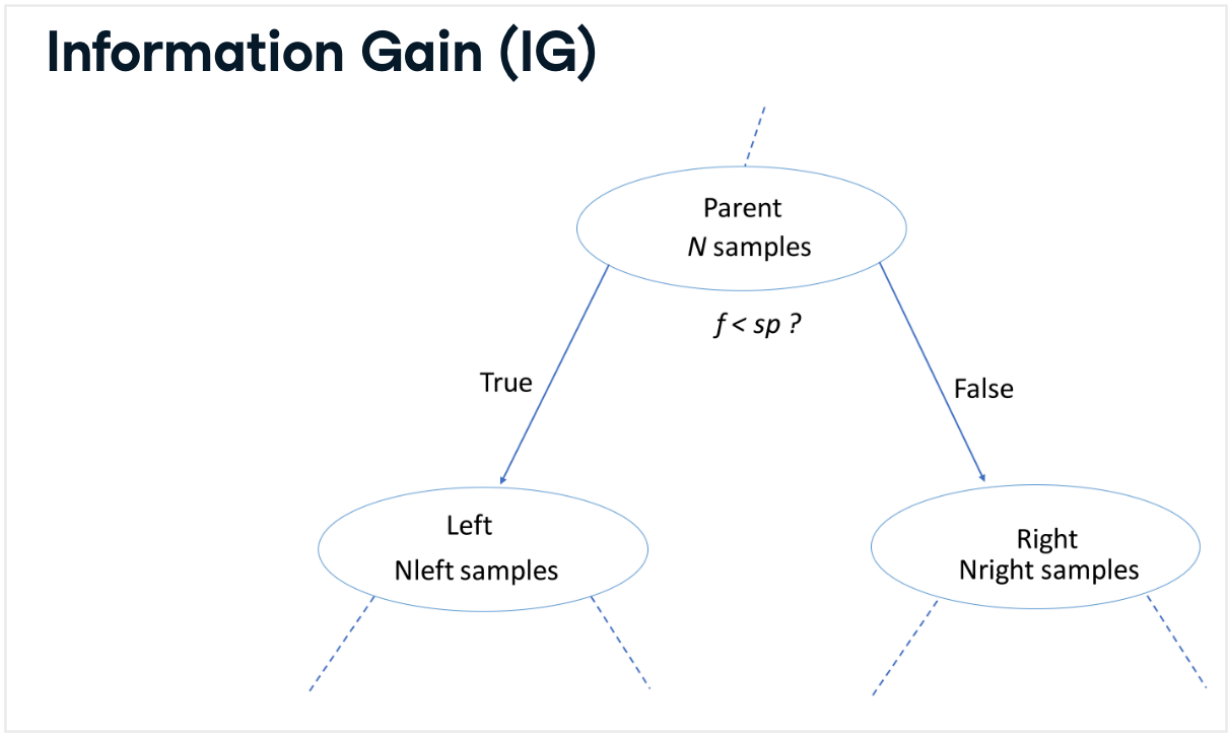
- **Decision-Tree**: data structure consisting of a hierarchy of nodes.
- **Node**: question or prediction.

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*.





sp - Split Point
The tree tries to maximize the information gain when a split is made.
Splitting an internal node always involves maximizing information gain!

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)
```

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

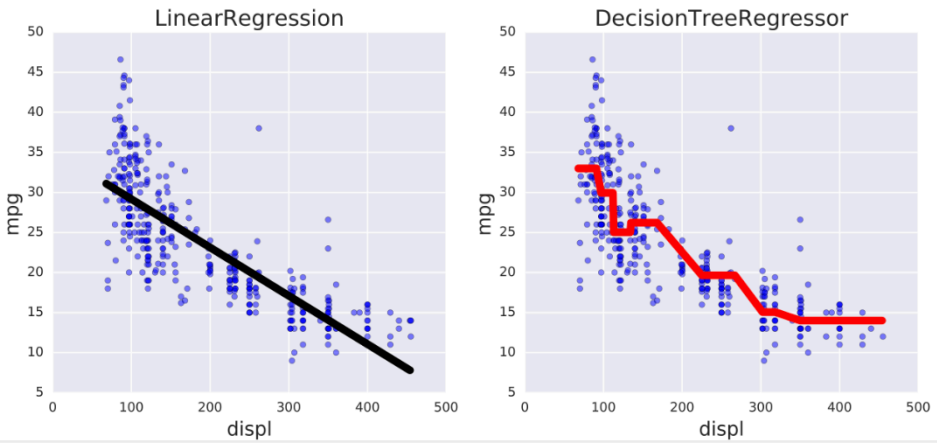
$$I(\text{node}) = \underbrace{\text{MSE(node)}}_{mean-squared-error} = \frac{1}{N_{node}} \sum_{i \in node} \left(y^{(i)} - \hat{y}_{node}\right)^2$$

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

Prediction

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

Linear Regression vs. Regression-Tree



The Bias-Variance Tradeoff

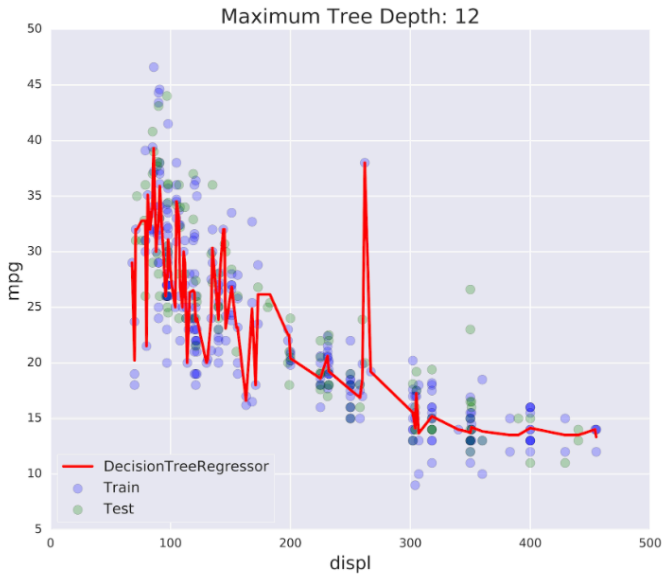
Goals of Supervised Learning

- 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 achieve a low predictive error on unseen datasets.

Difficulties in Approximating f

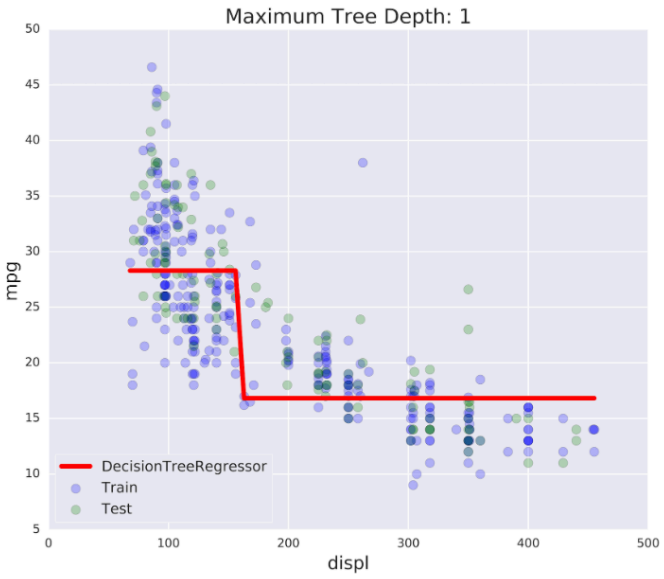
- **Overfitting**: $\hat{f}(x)$ fits the training set noise.
- **Underfitting**: $\hat{f}$ is not flexible enough to approximate f.

Overfitting



- low training set error and high test set error - Overfitting
- high training and test set error - Undercutting

Underfitting



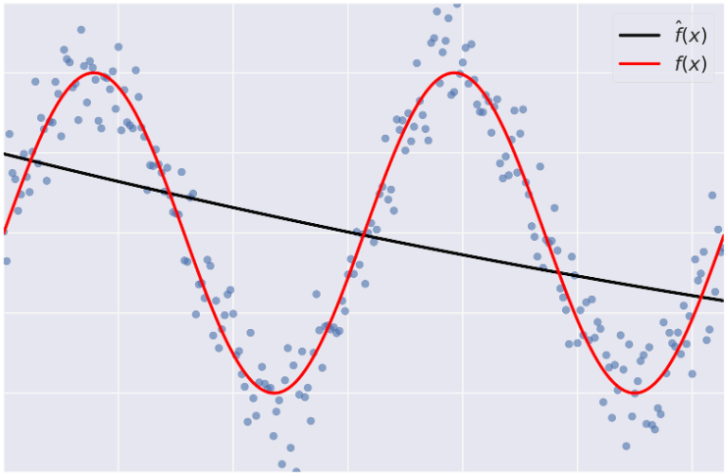
Generalization Error

Generalization Error

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

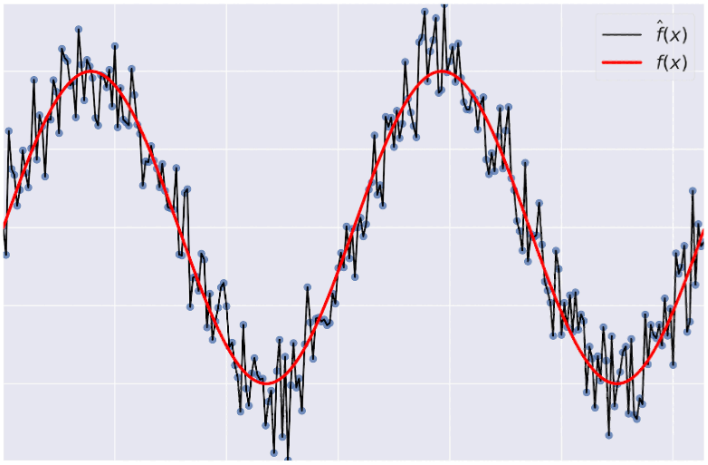
Bias

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



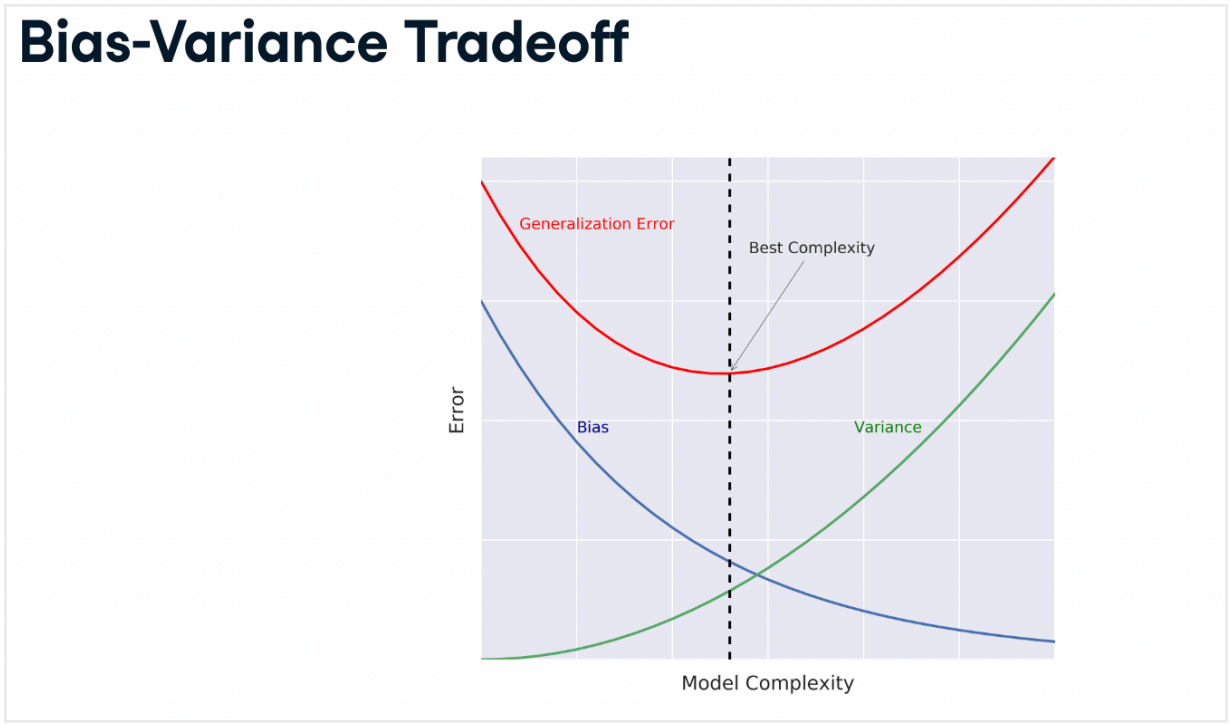
Variance

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

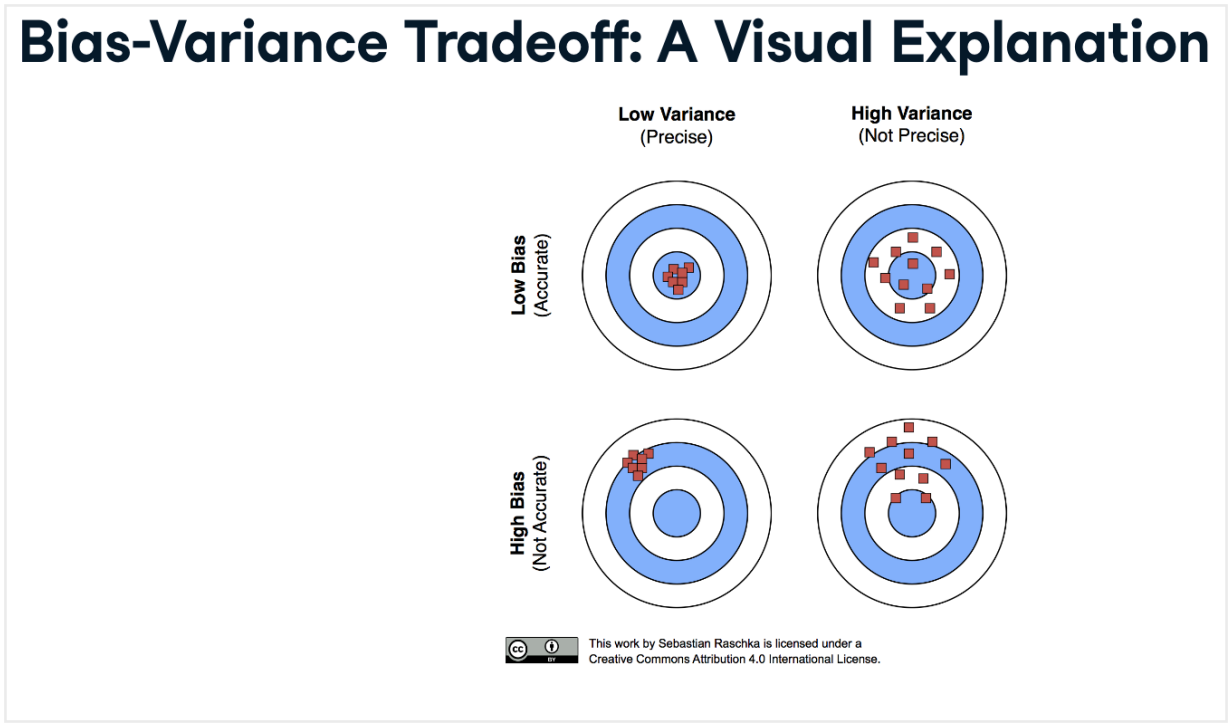


Model Complexity

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



When model complexity increases the bias(how much is f^ different from f) decreases and variance (how well f^ generalizes on unseen dataset) increases. Our goal is to find a model complexity on which the variance and bias both are low.



Diagnose bias and variance problems

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

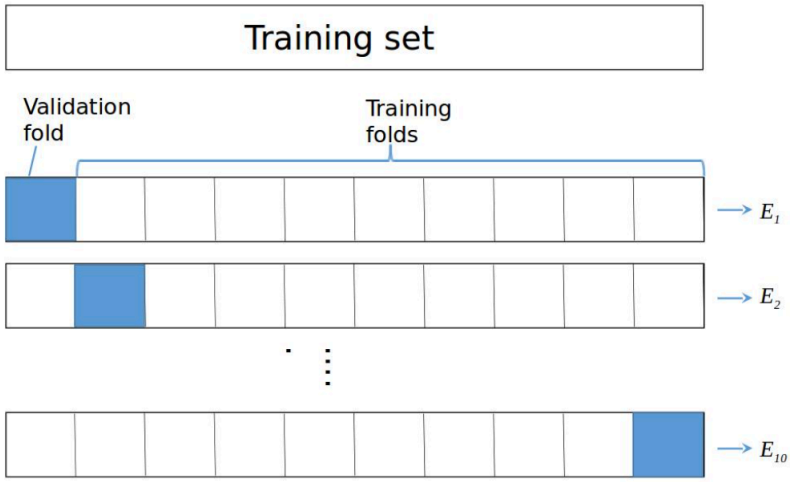
Solution:

- 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 $\rightarrow$ 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)
```

```
# 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
```

Ensemble Learning

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

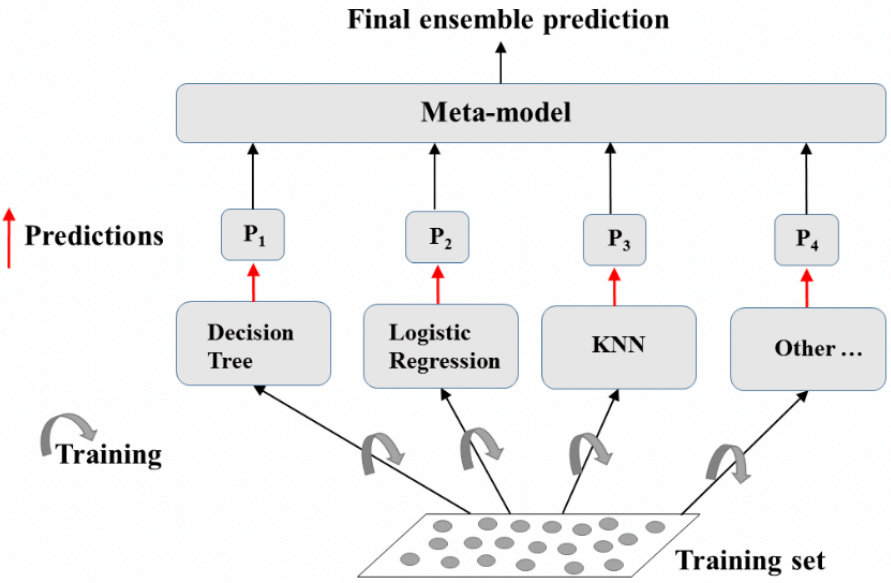
- Classification: can only produce orthogonal decision boundaries.
- Sensitive to small variations in the training set.
- High variance: unconstrained CARTs may overfit the training set.
- Solution: ensemble learning.

CART - Classification and Regression Trees

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.

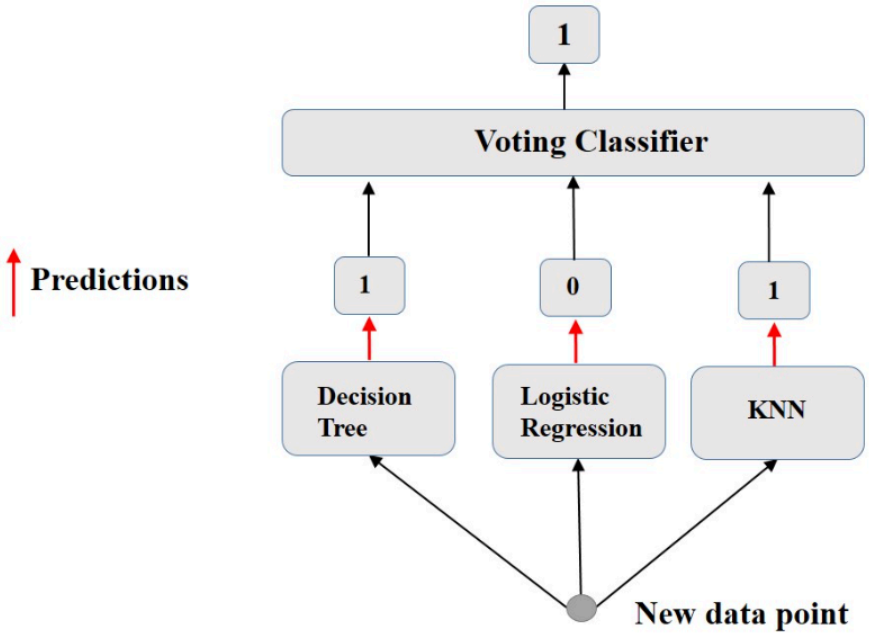
Ensemble Learning: A Visual Explanation



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



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)]
```

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
```

Bagging

Ensemble Methods

Voting Classifier

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

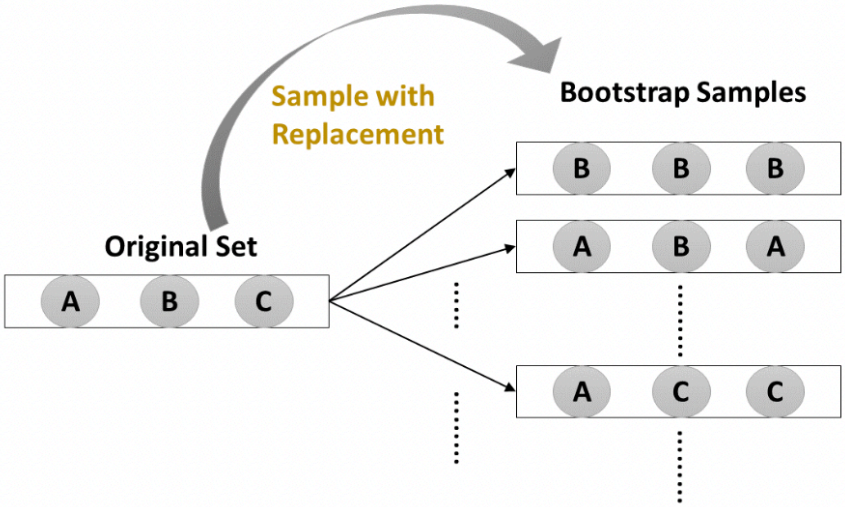
Bagging

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

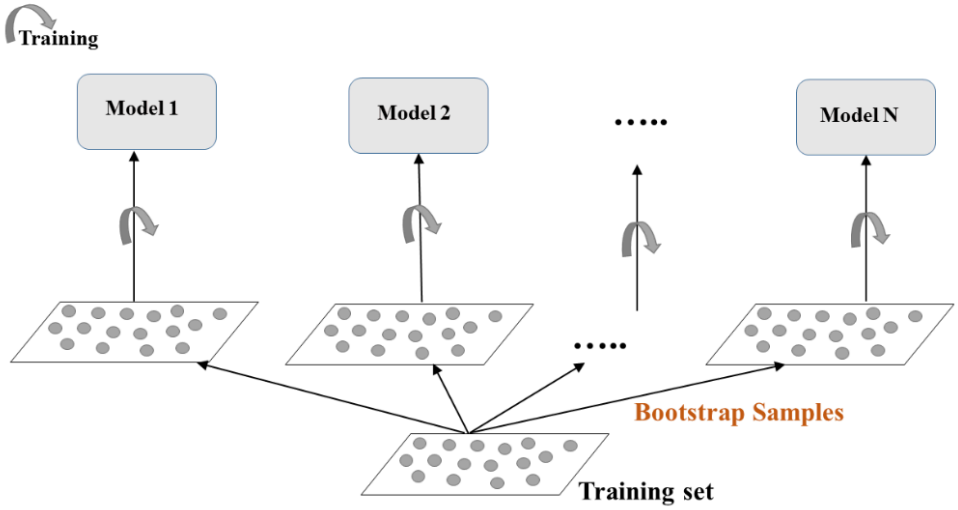
Bagging

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

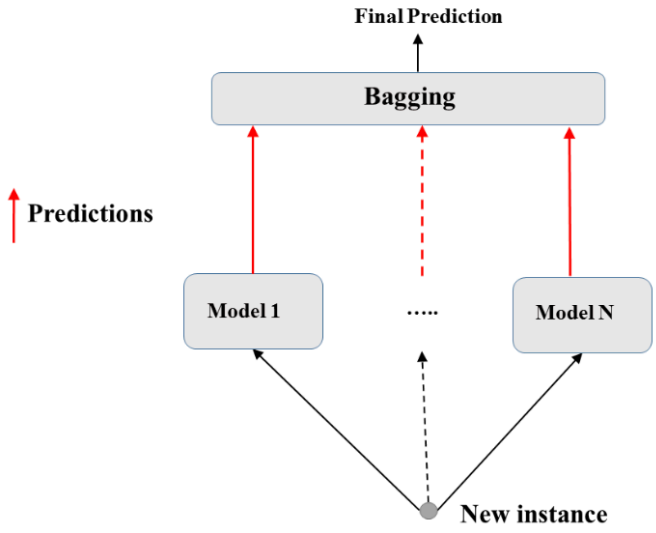
Bootstrap



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)
```

bc = BaggingClassifier(base_estimator = decree, n_estimators = 300, n_jobs = -1)
Here, n_estimators - Number of decision trees to be used
n_jobs = -1, use the maximum cpu cores for computation.

```
# 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
```

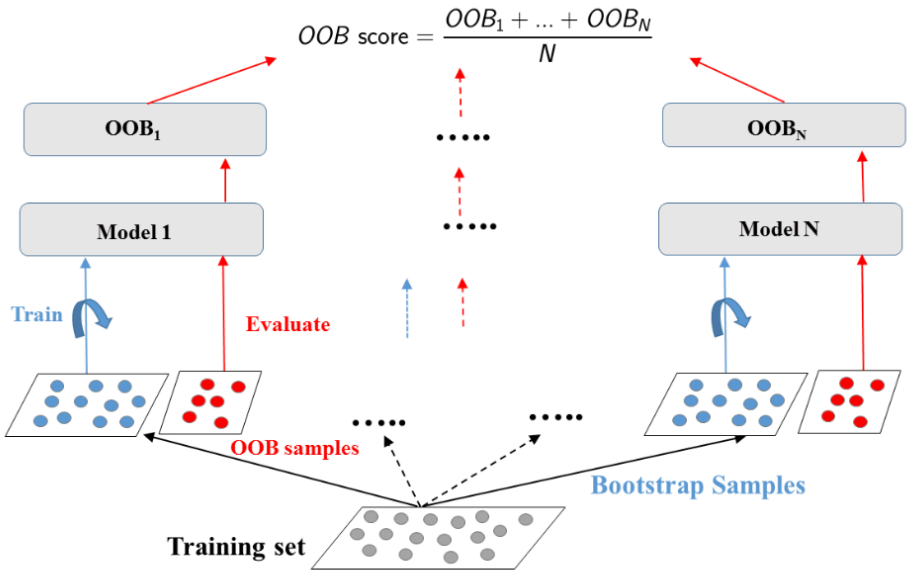
Bagging

- 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)
```

```
# 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
```

Random Forests (RF)

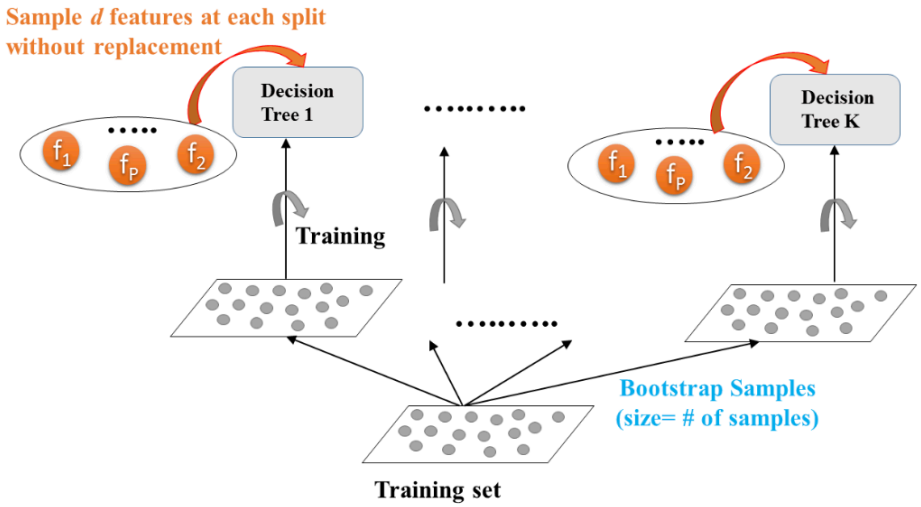
Bagging

- 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

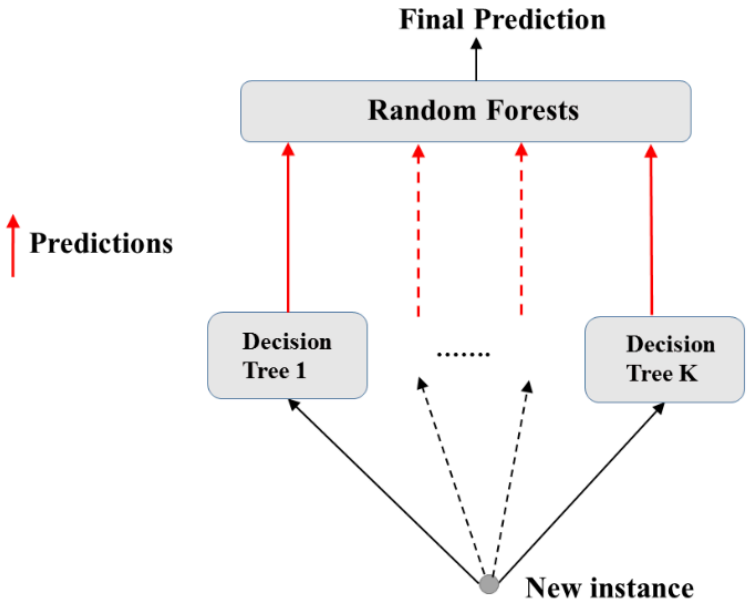
Further Diversity with Random Forests

- 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
  ($d <$ 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