Decision-Tree for Classification

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

datacamp

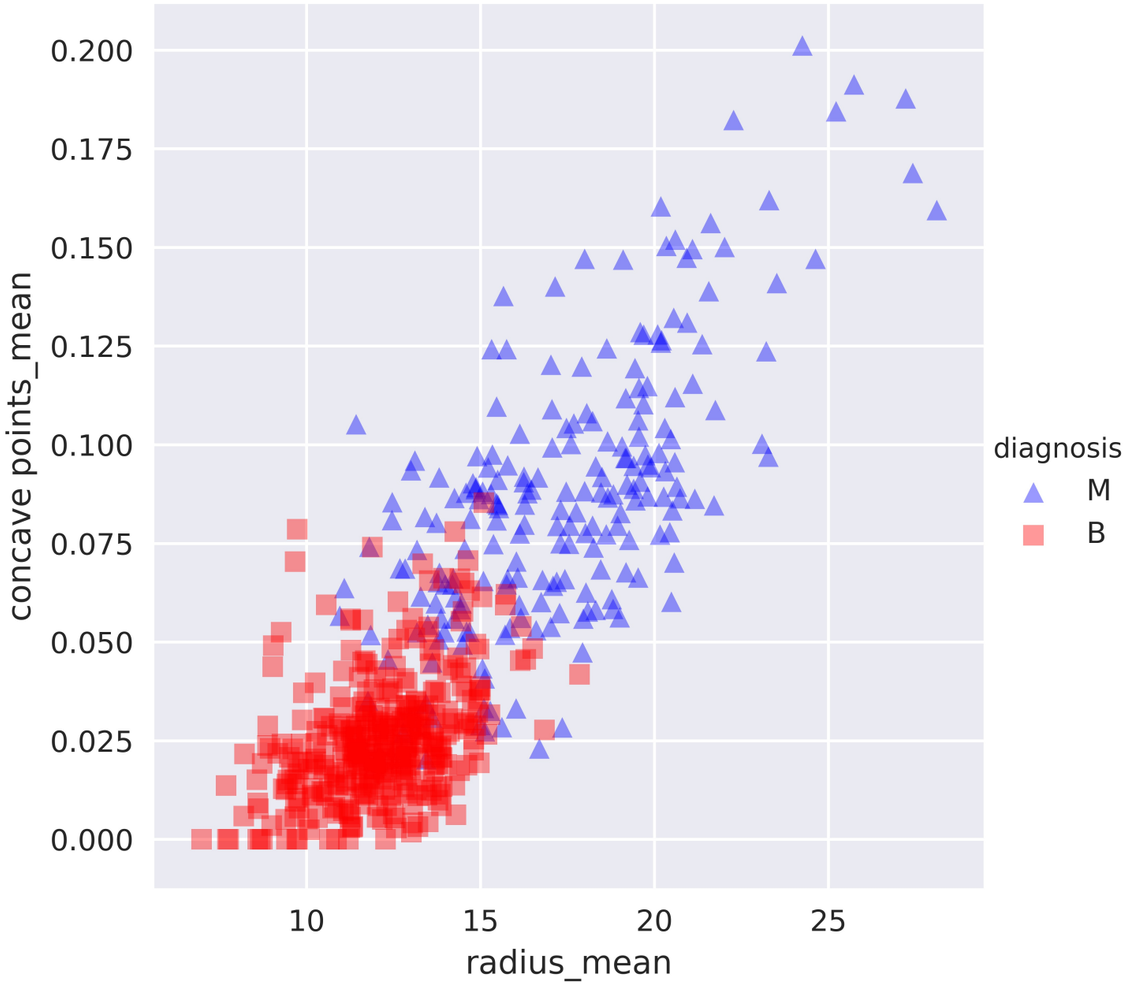
Course Overview

- **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

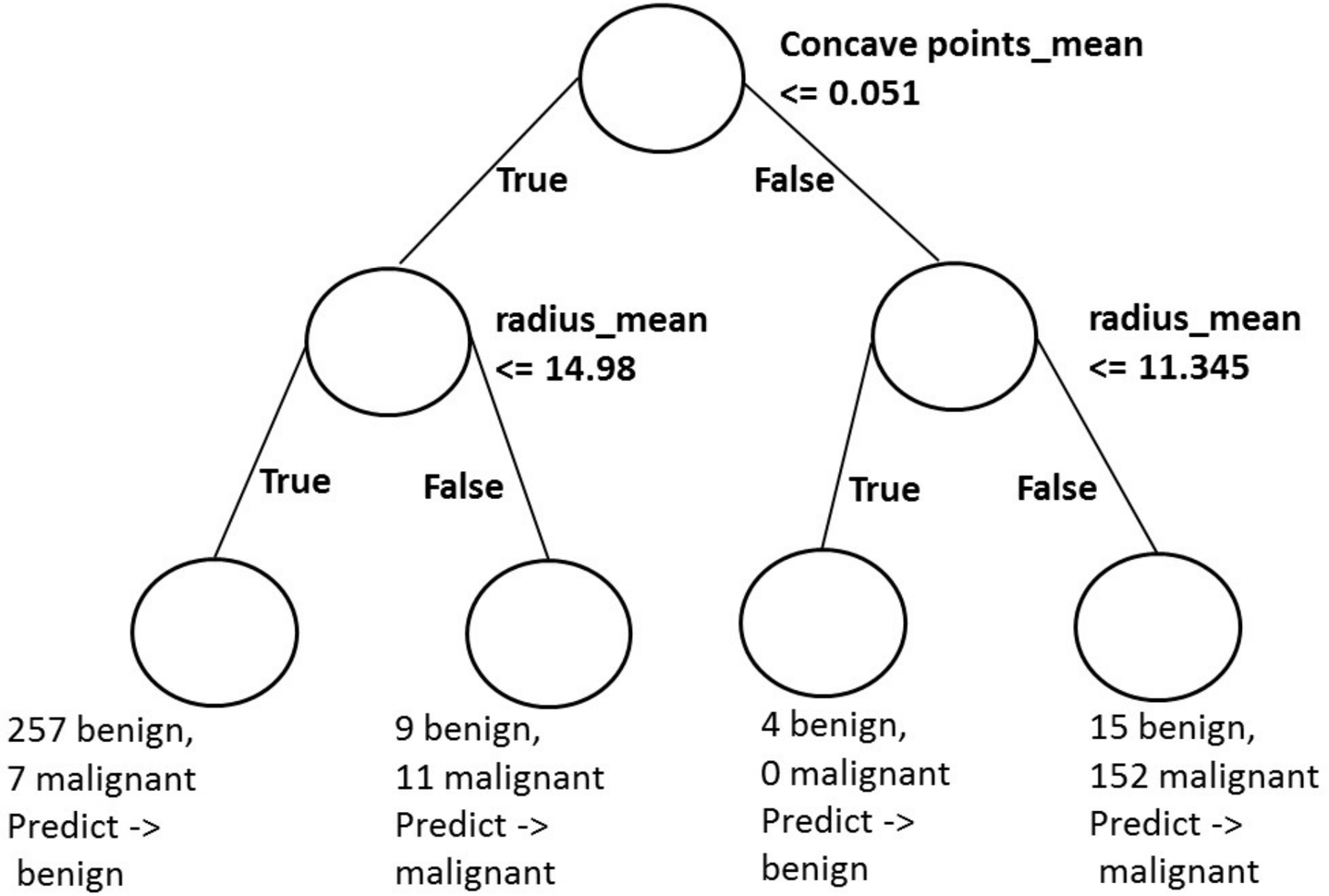
Classification-tree

- 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

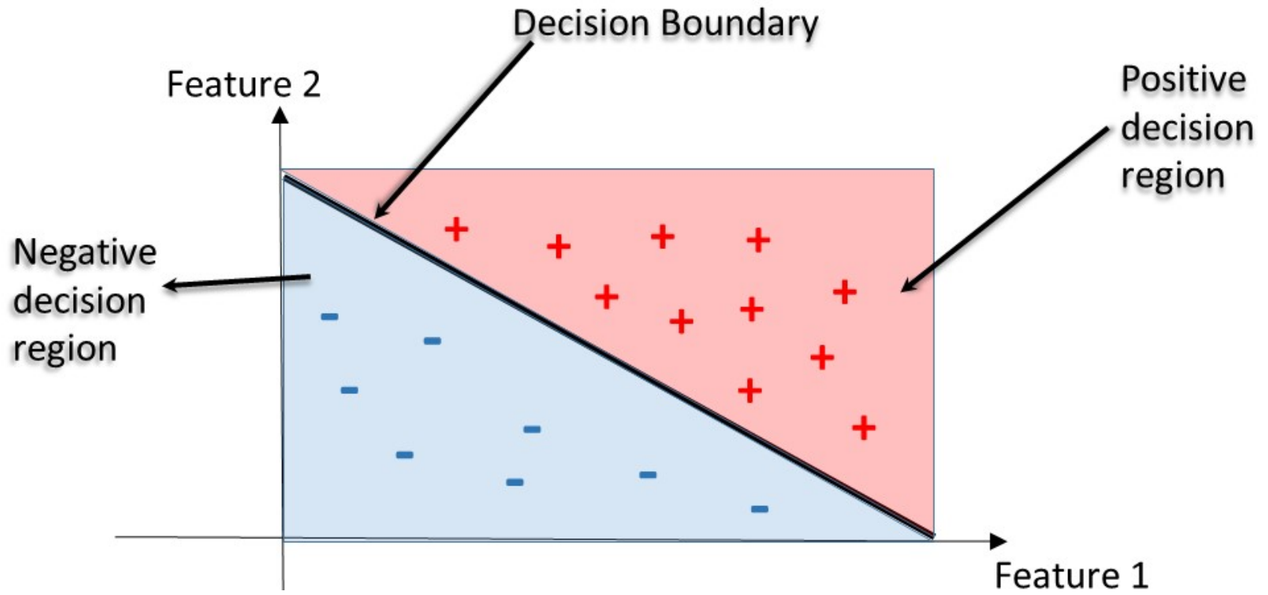
```
# Fit dt to the training set
dt.fit(X_train,y_train)


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

```
0.9035087719298245
```

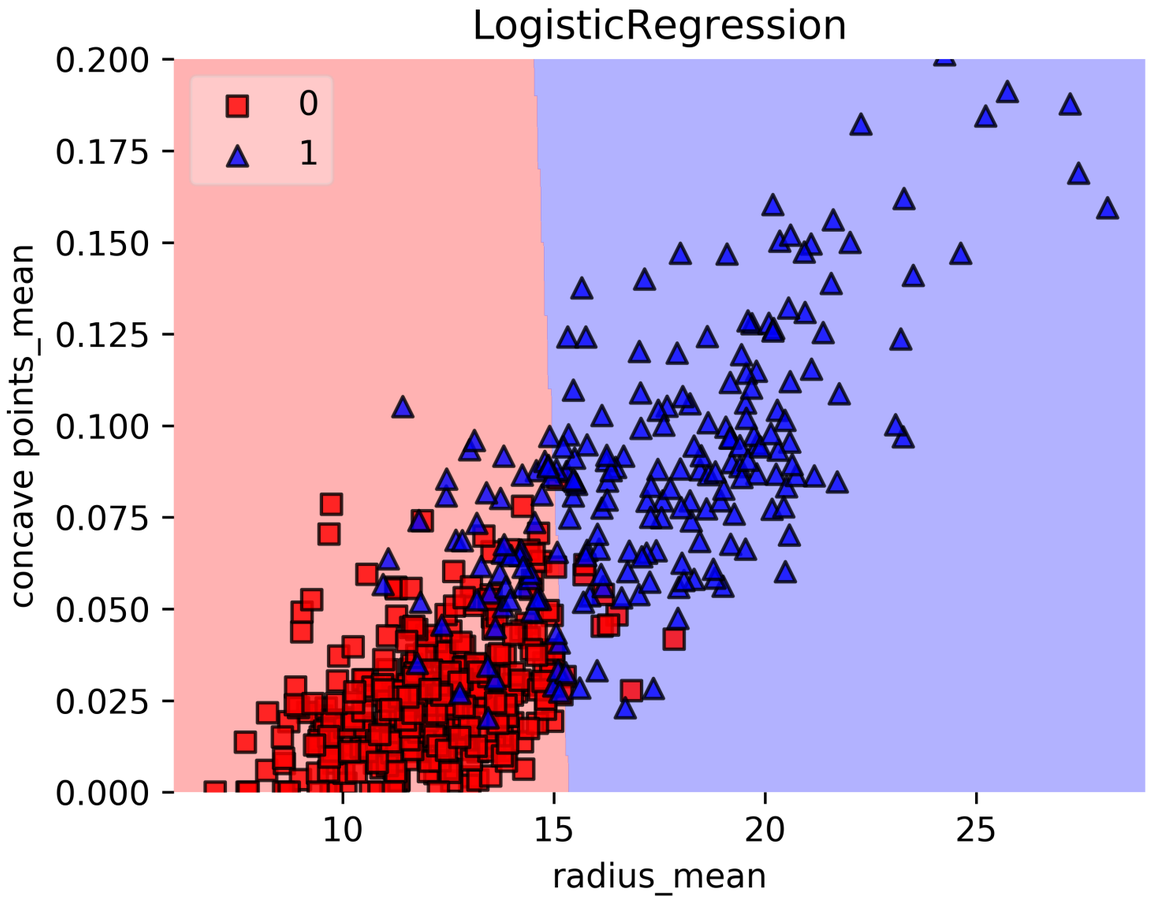
Decision Regions

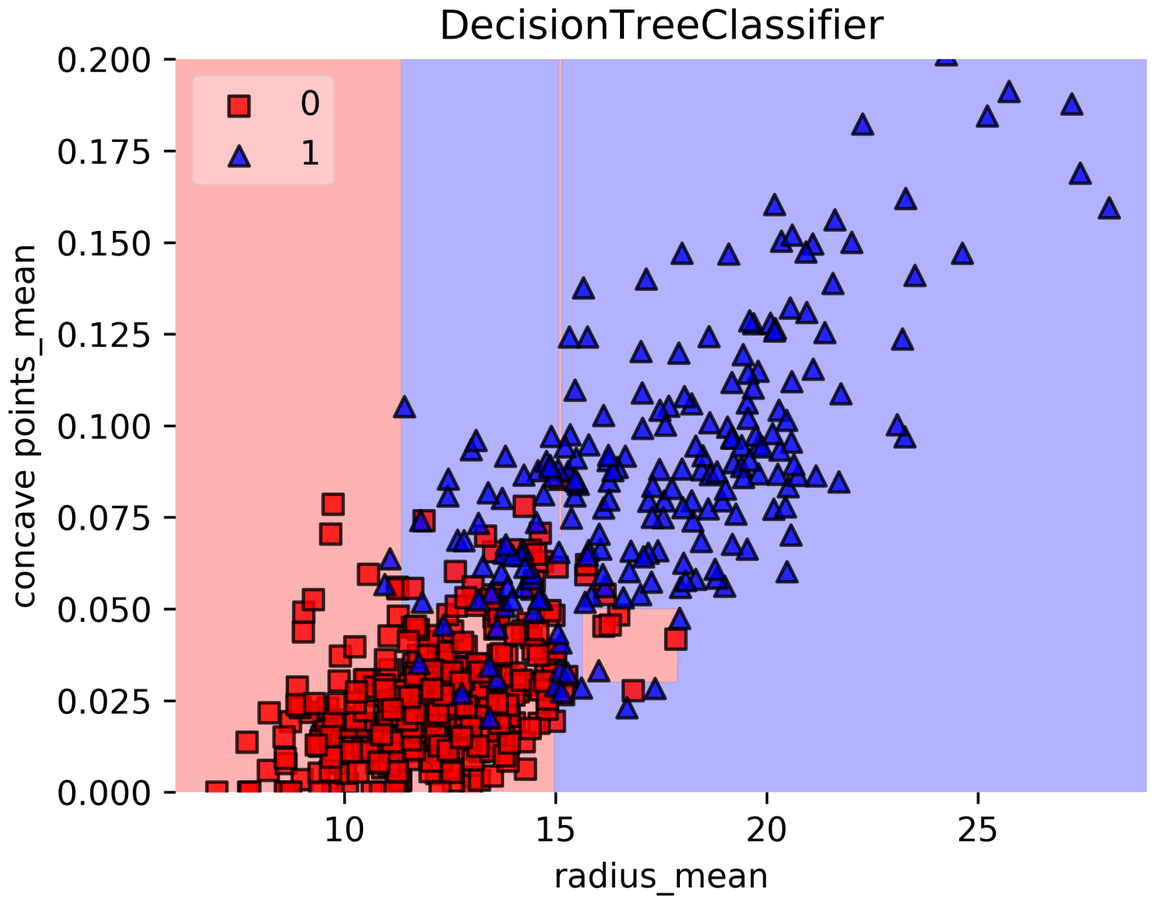
Decision region: region in the feature space where all instances are assigned to one class label.

Decision Boundary: surface separating different decision regions.



Decision Regions: CART vs. Linear Model





Classification-Tree Learning

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

datacamp

Building Blocks of a Decision-Tree

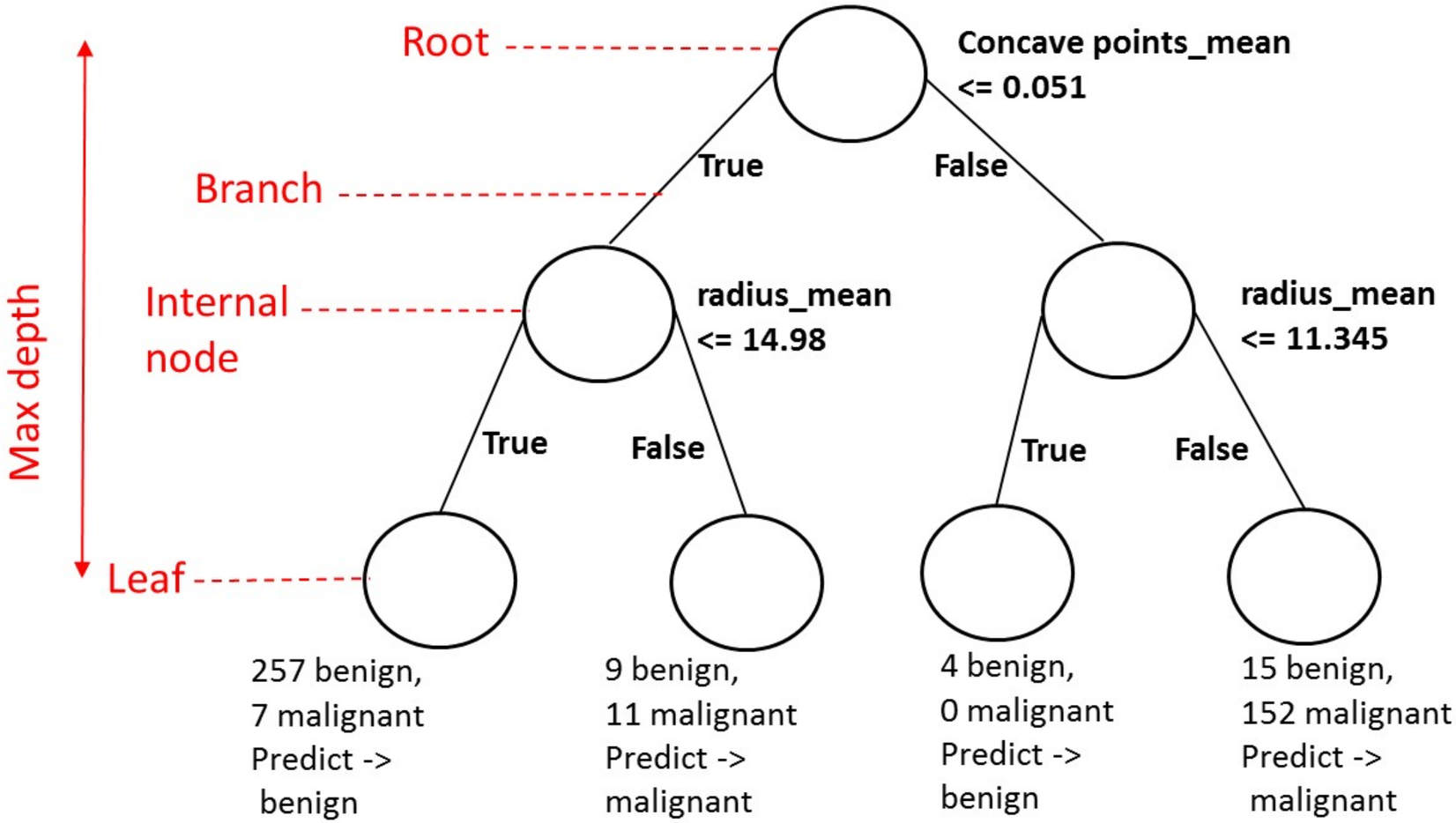
- **Decision-Tree**: data structure consisting of a hierarchy of nodes.
- **Node**: question or prediction.

Building Blocks of a Decision-Tree

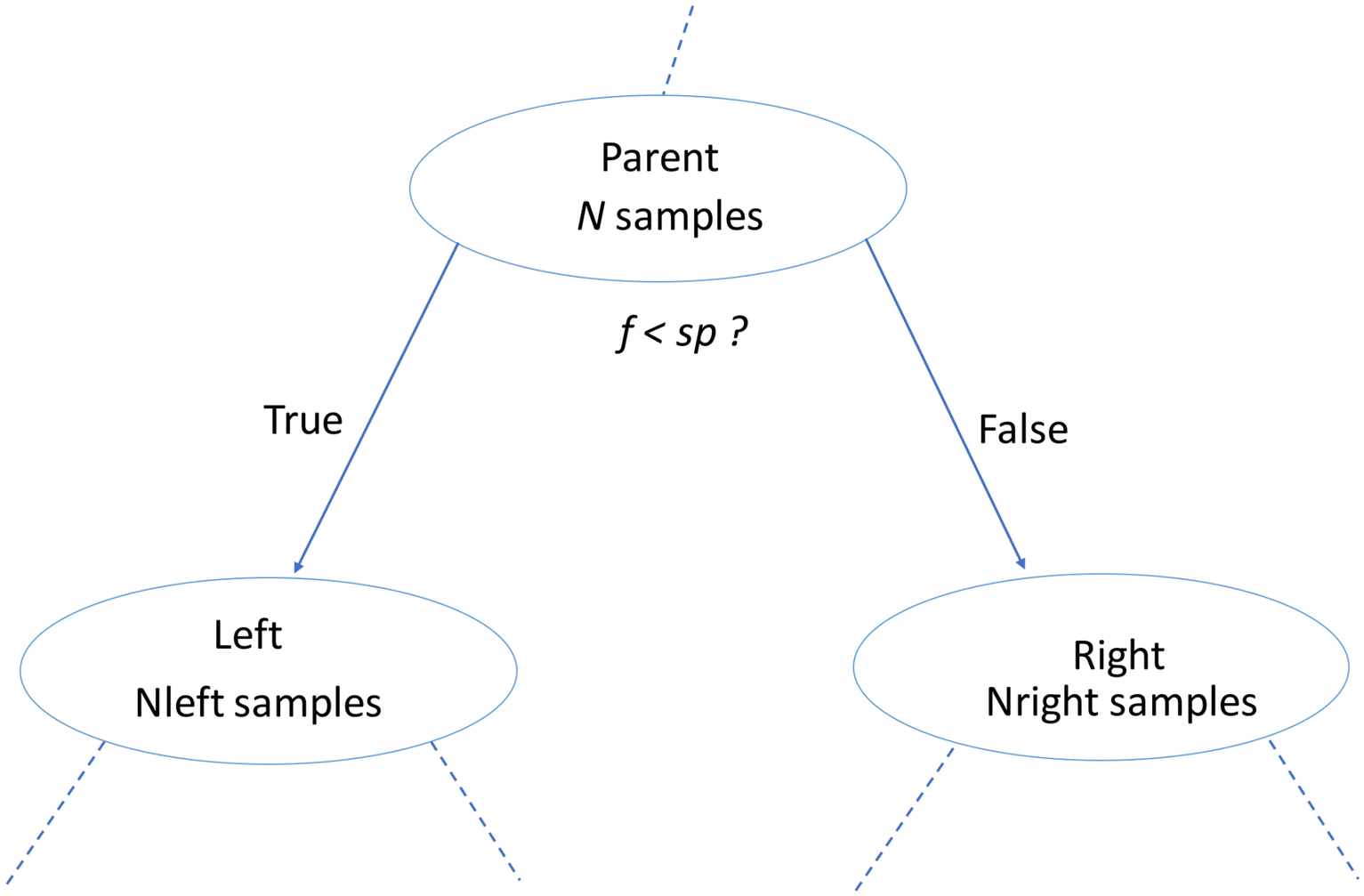
Three kinds of nodes:

- **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
```

Decision-Tree for Regression

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

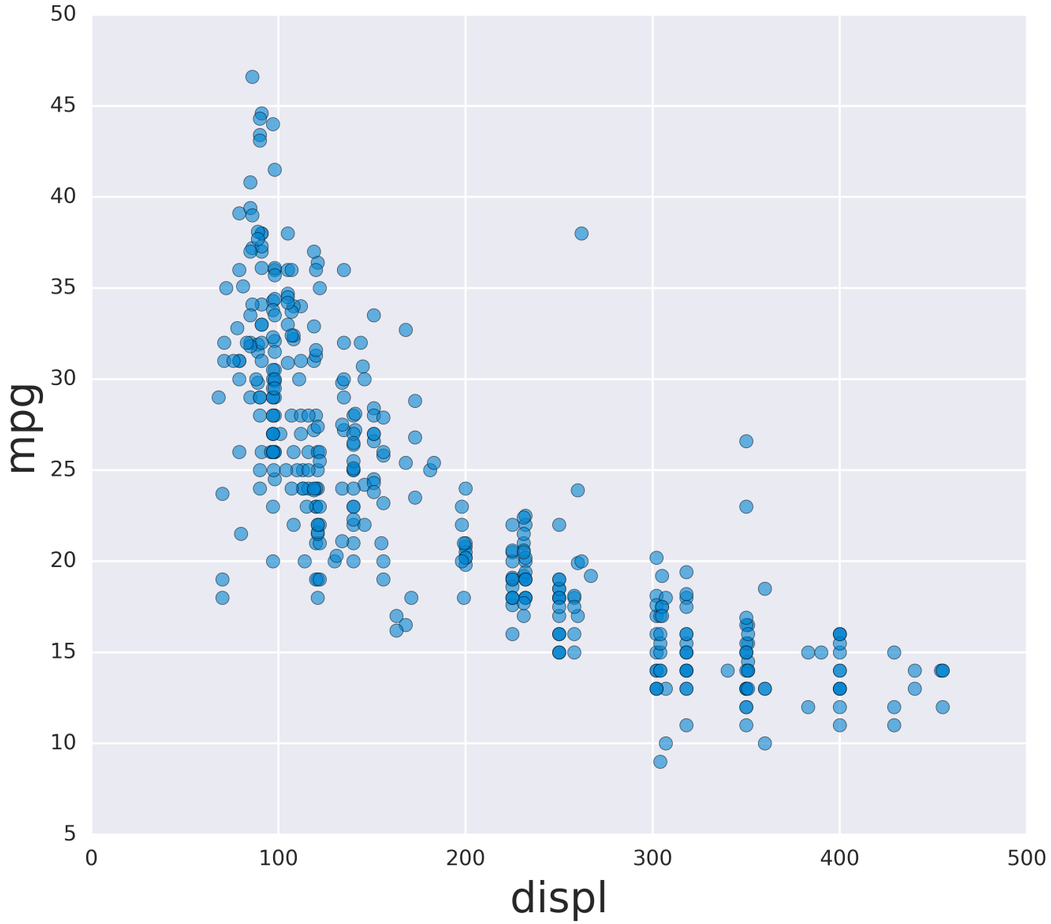
Elie Kawerk
Data Scientist

datacamp

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

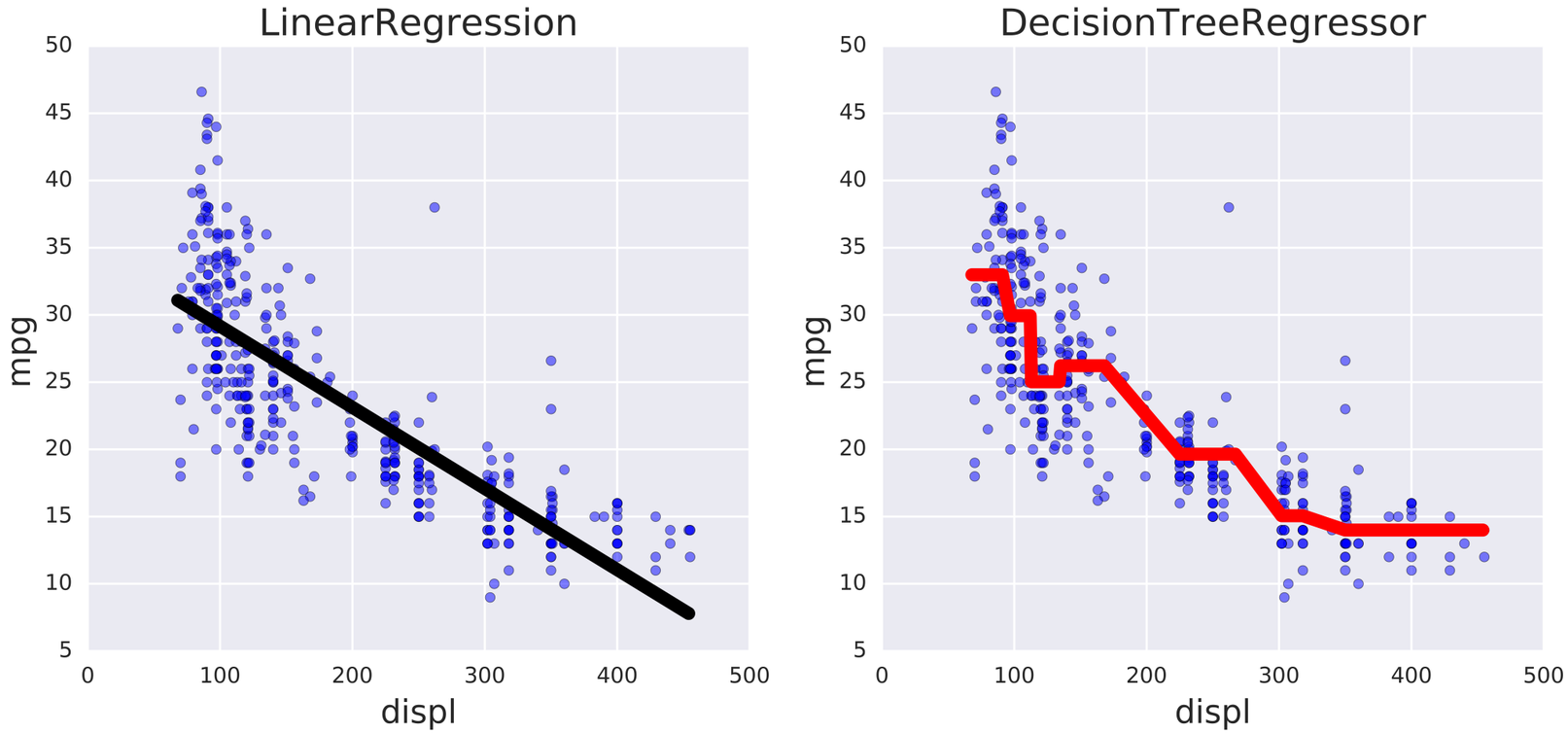
$$I(\text{node}) = \underbrace{\text{MSE(node)}}_{mean-squared-error} = \frac{1}{N_{node}} \sum_{i \in node} \left(y^{(i)} - \hat{y}_{node}\right)^2$$

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

Prediction

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

Linear Regression vs. Regression-Tree



Generalization Error

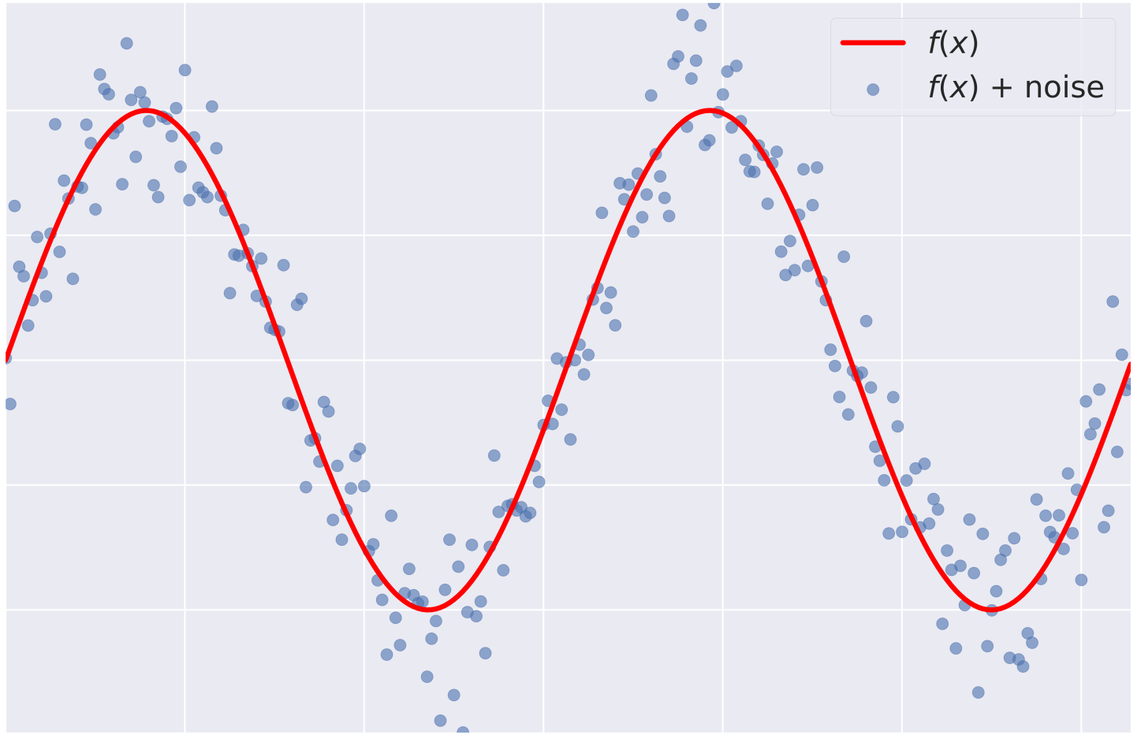
MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

datacamp

Supervised Learning - Under the Hood

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



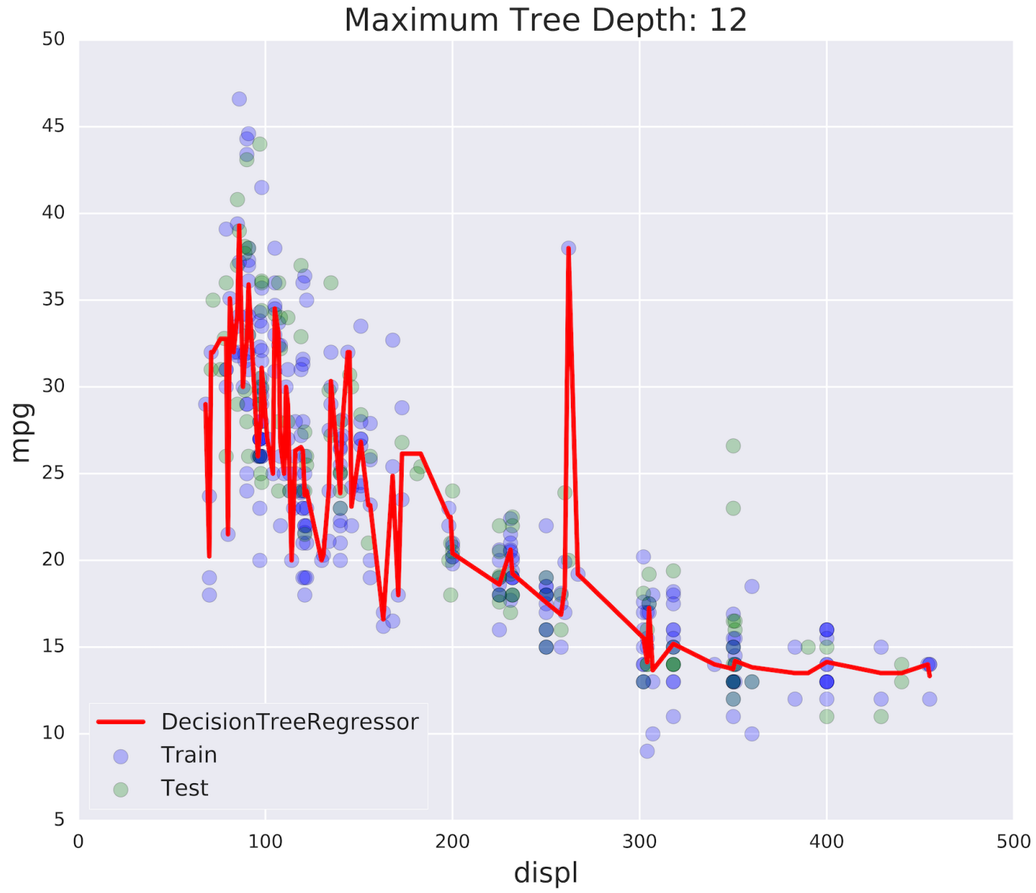
Goals of Supervised Learning

- 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.

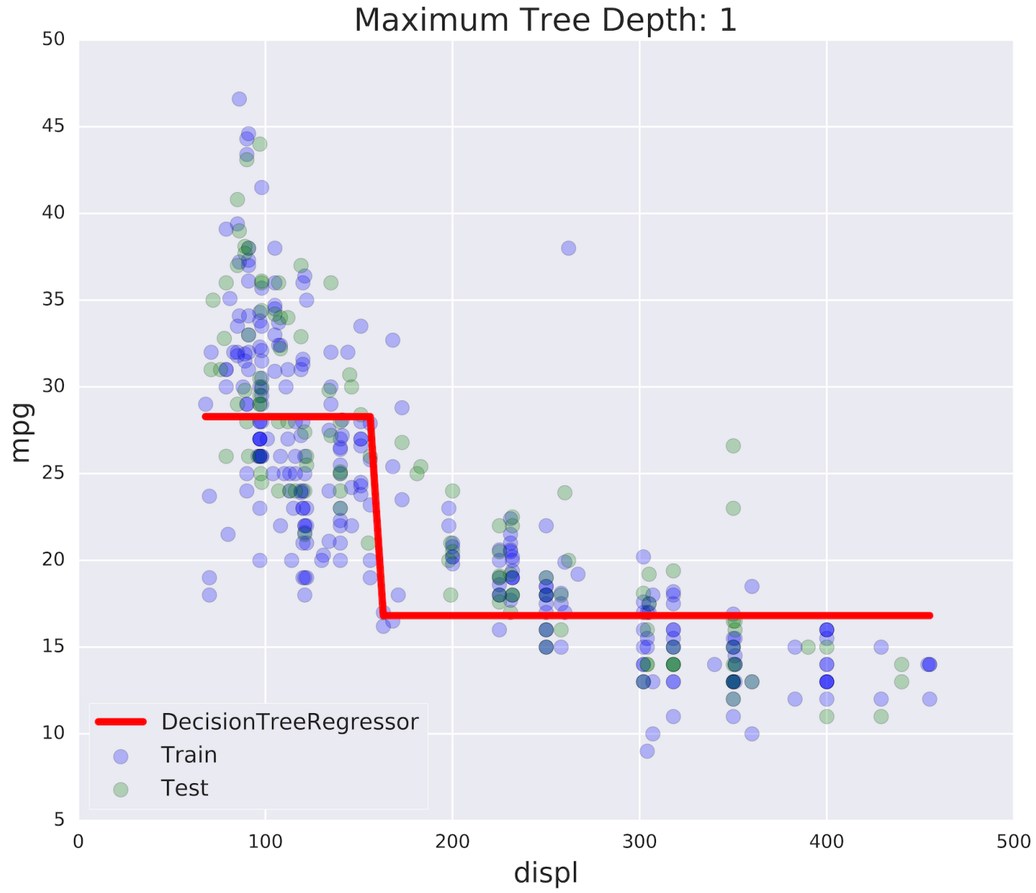
Difficulties in Approximating f

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

Overfitting



Underfitting

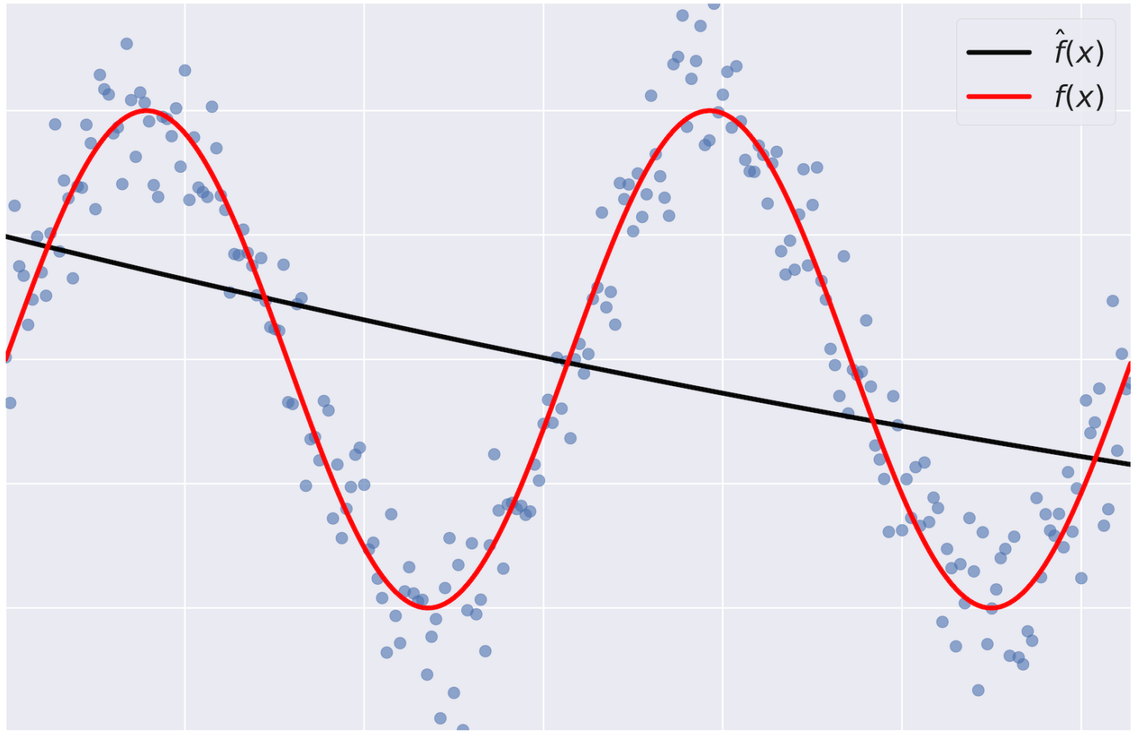


Generalization Error

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

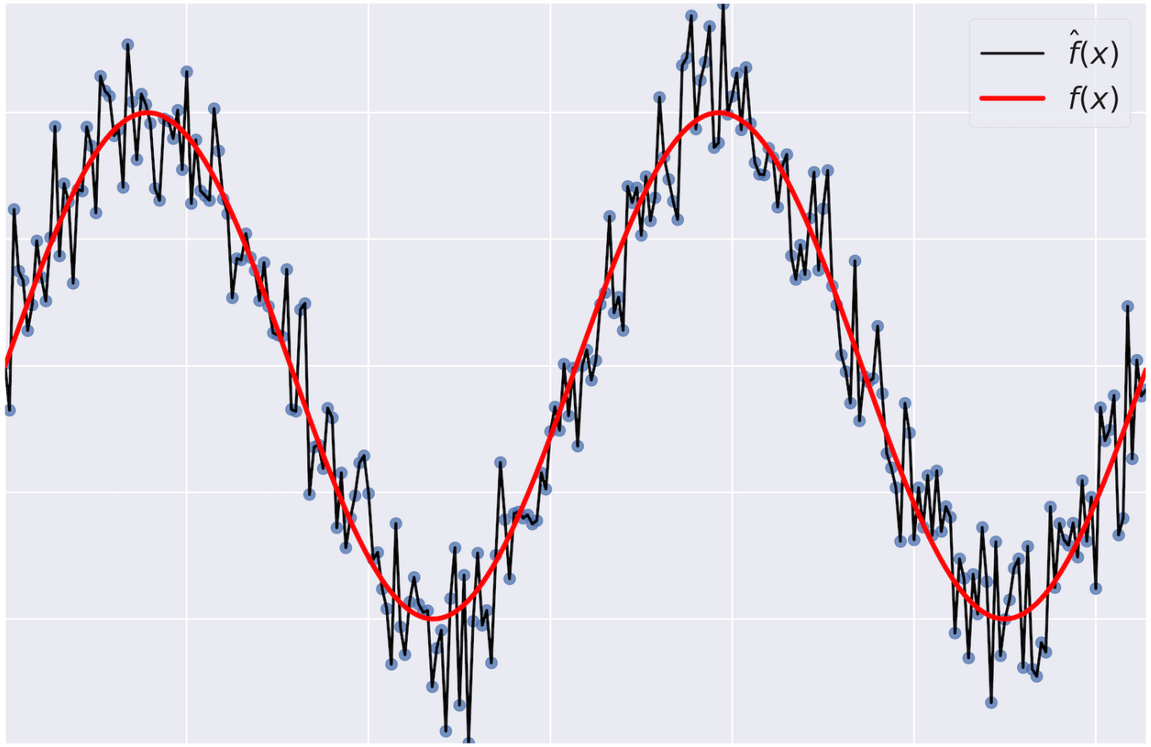
Bias

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



Variance

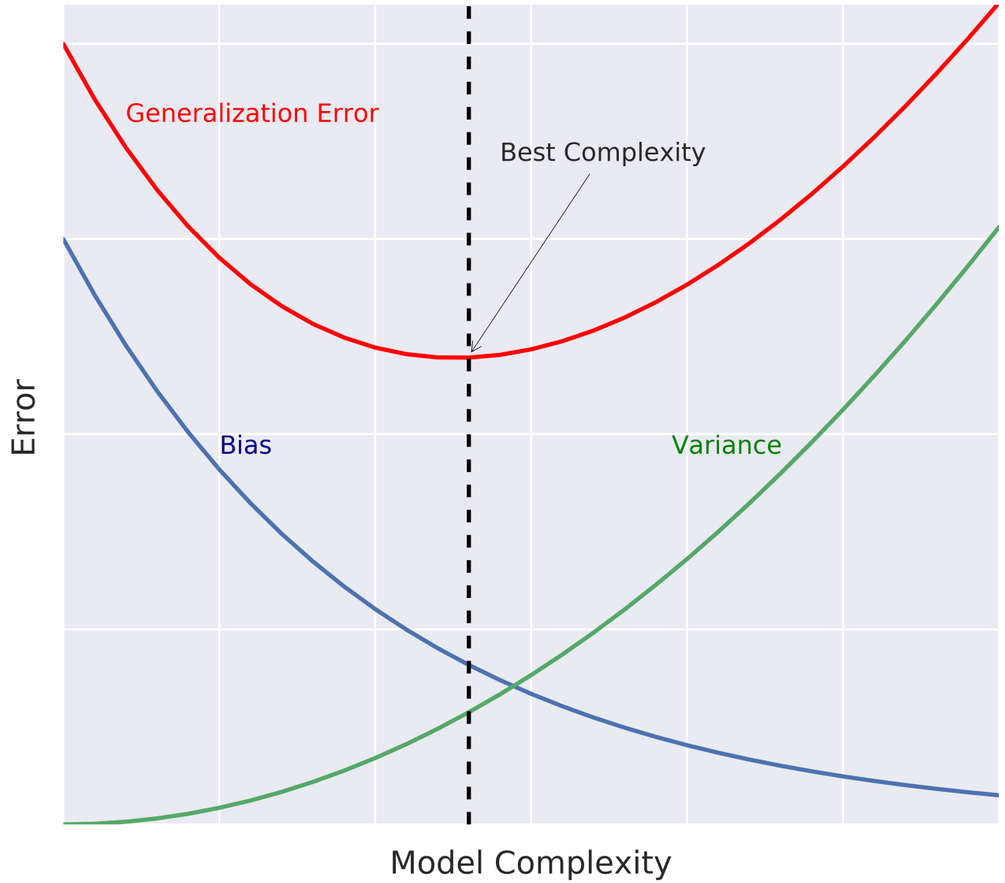
- **Variance**: tells you how much $\hat{f}$ is inconsistent over different training sets.



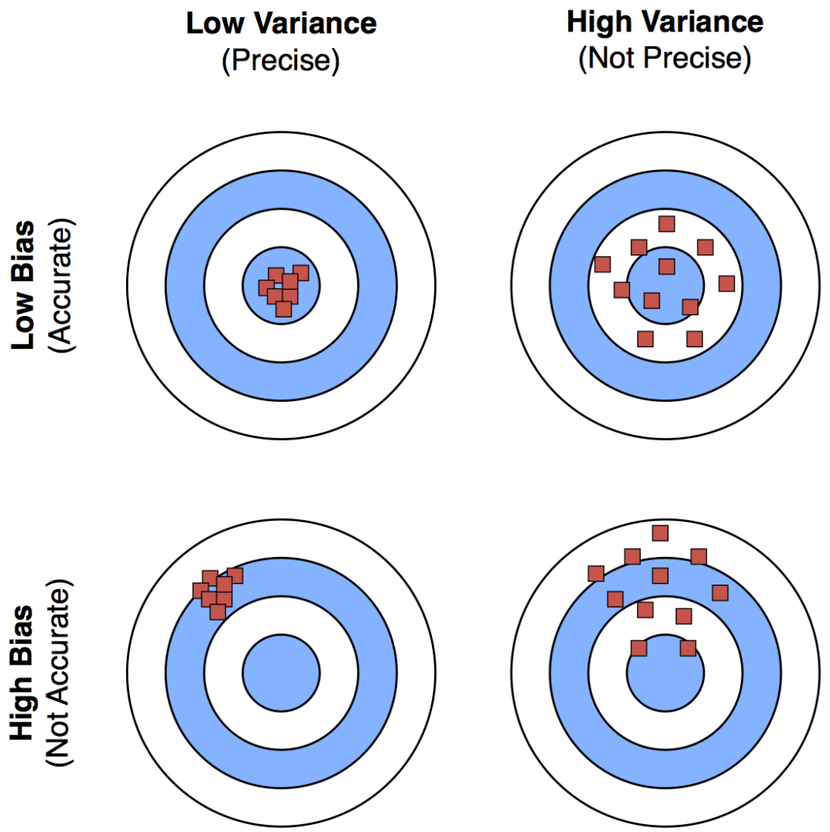
Model Complexity

- **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

datacamp

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

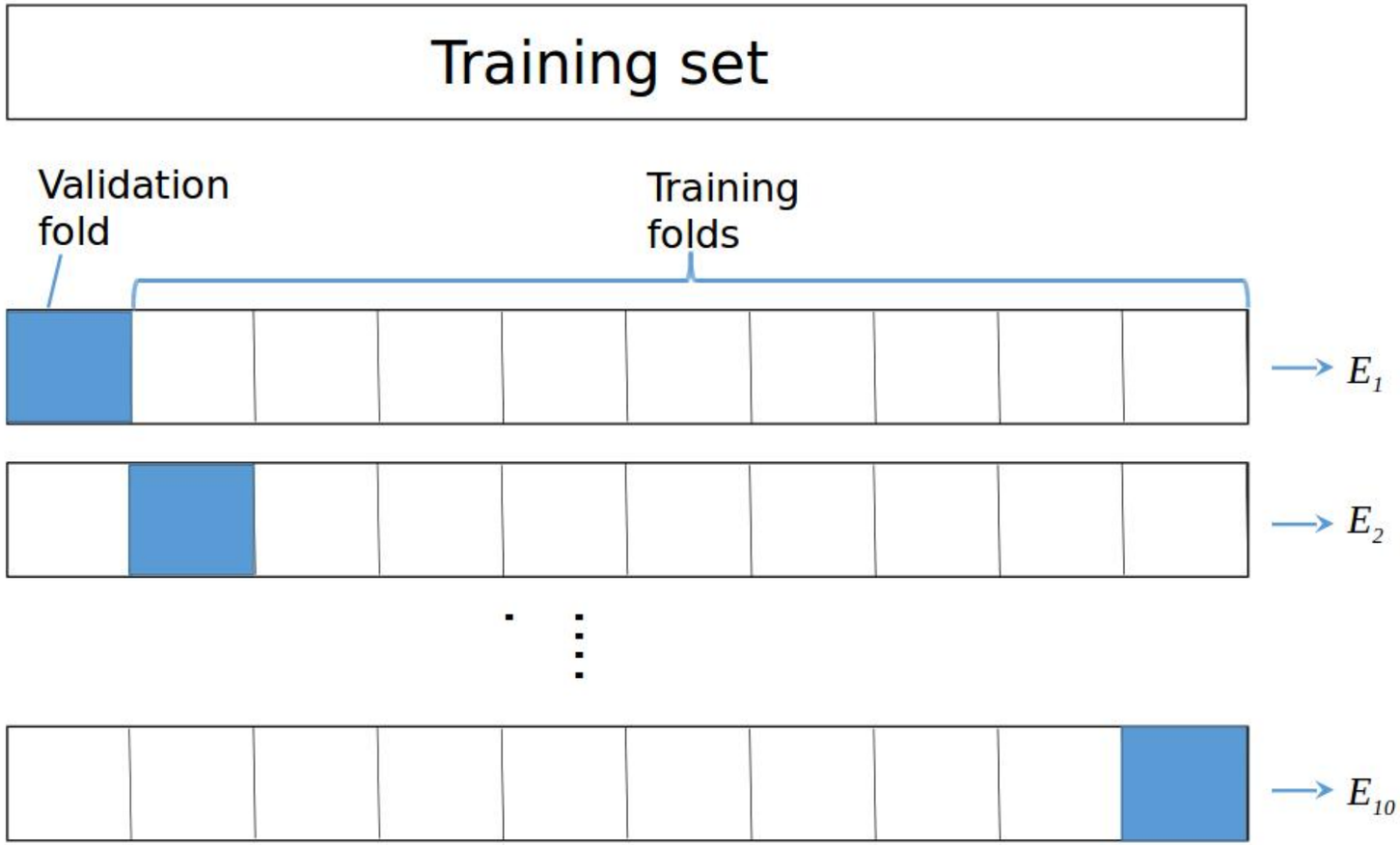
Solution:

- 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)
```

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
```

Ensemble Learning

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

datacamp

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, ...

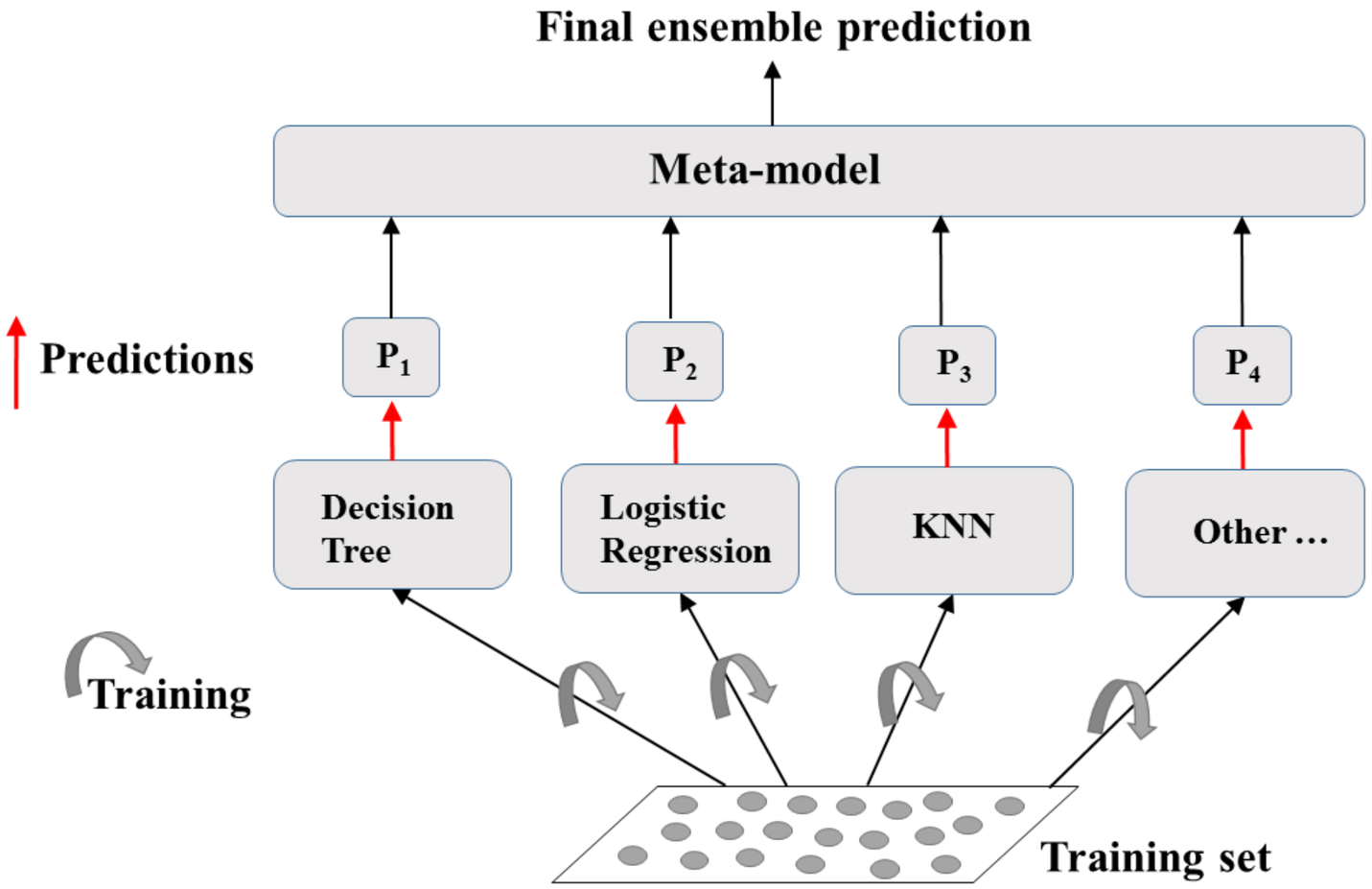
Limitations of CARTs

- 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.

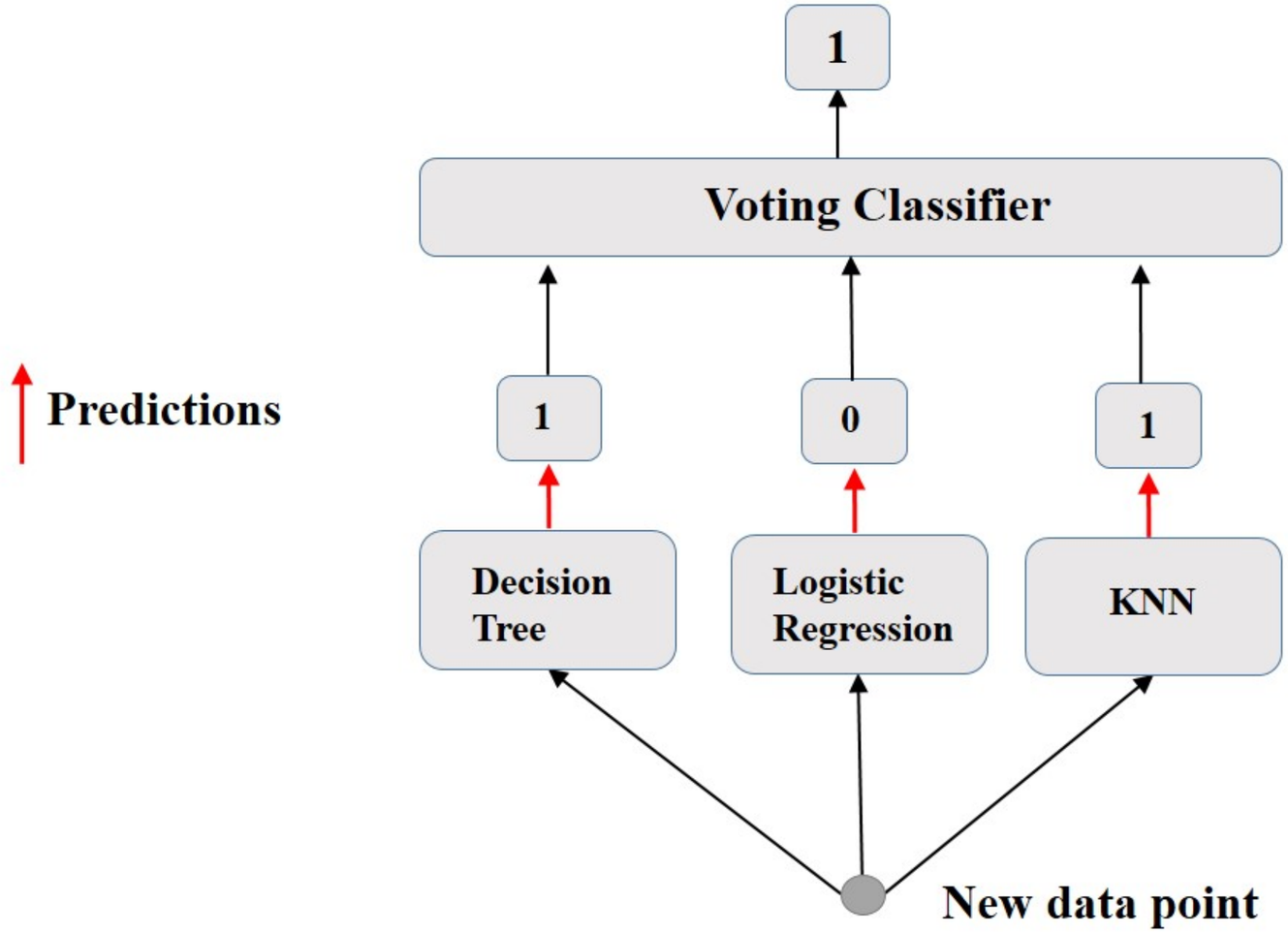
Ensemble Learning: A Visual Explanation



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


1
Voting Classifier
Predictions
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
```