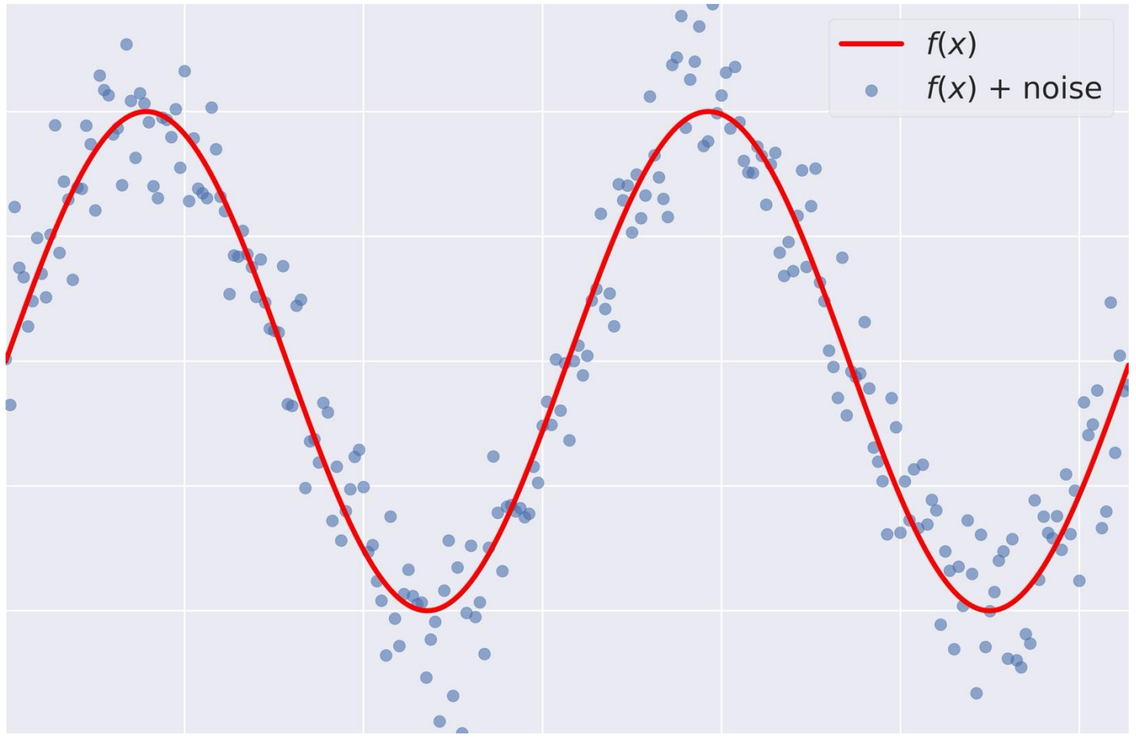
Generalization Error

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Supervised Learning - Under the Hood

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



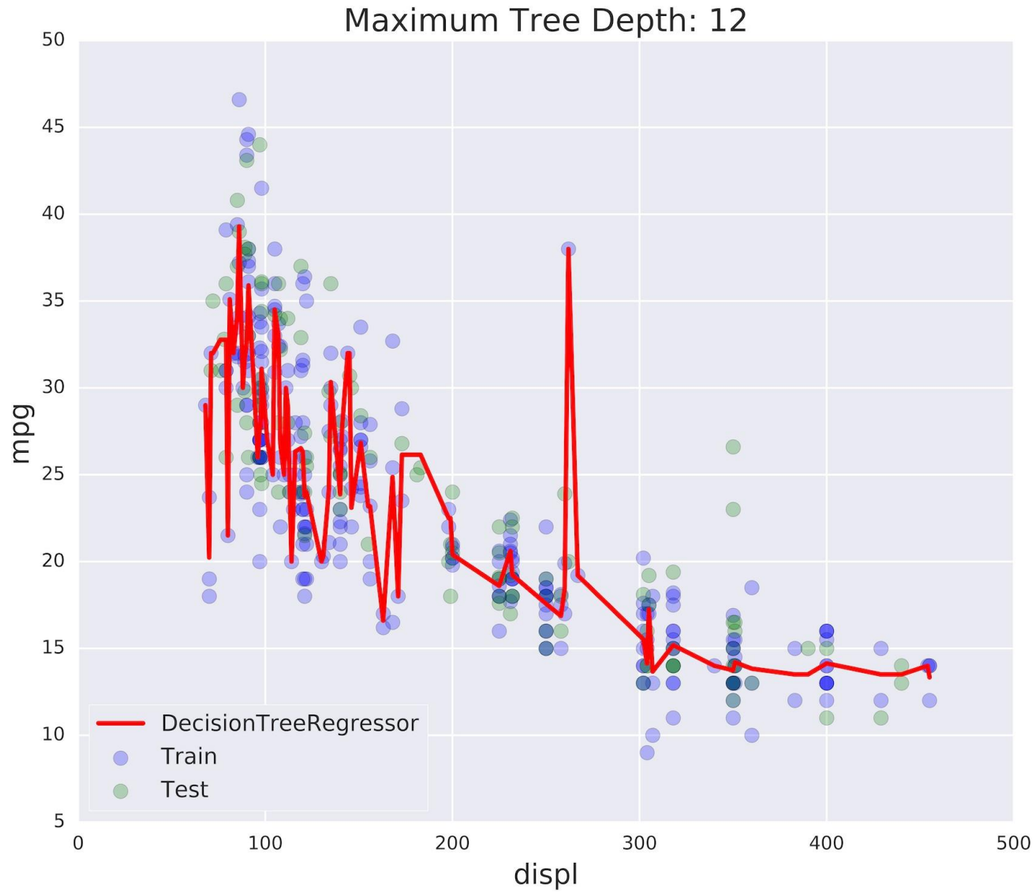
Goals of Supervised Learning

- 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 achieve a low predictive error on unseen datasets.

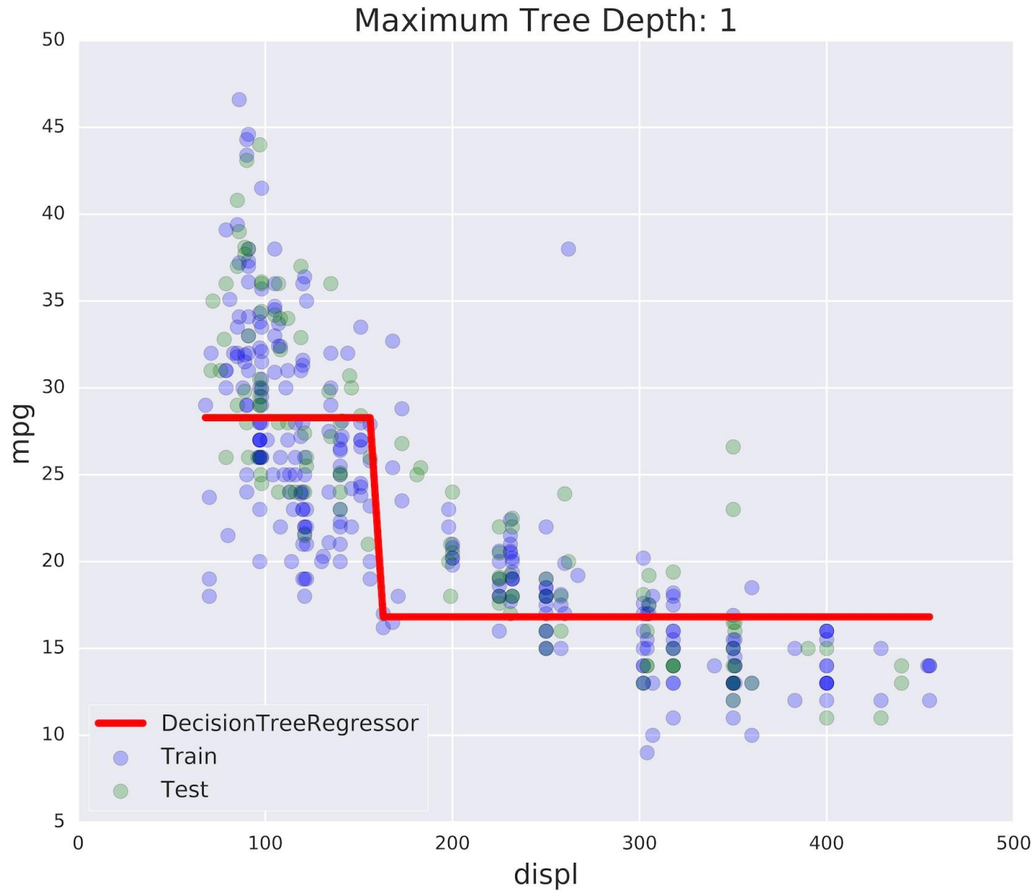
Difficulties in Approximating f

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

Overfitting



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 $\hat{f} = bias^2 + variance + \text{irreducible error}$

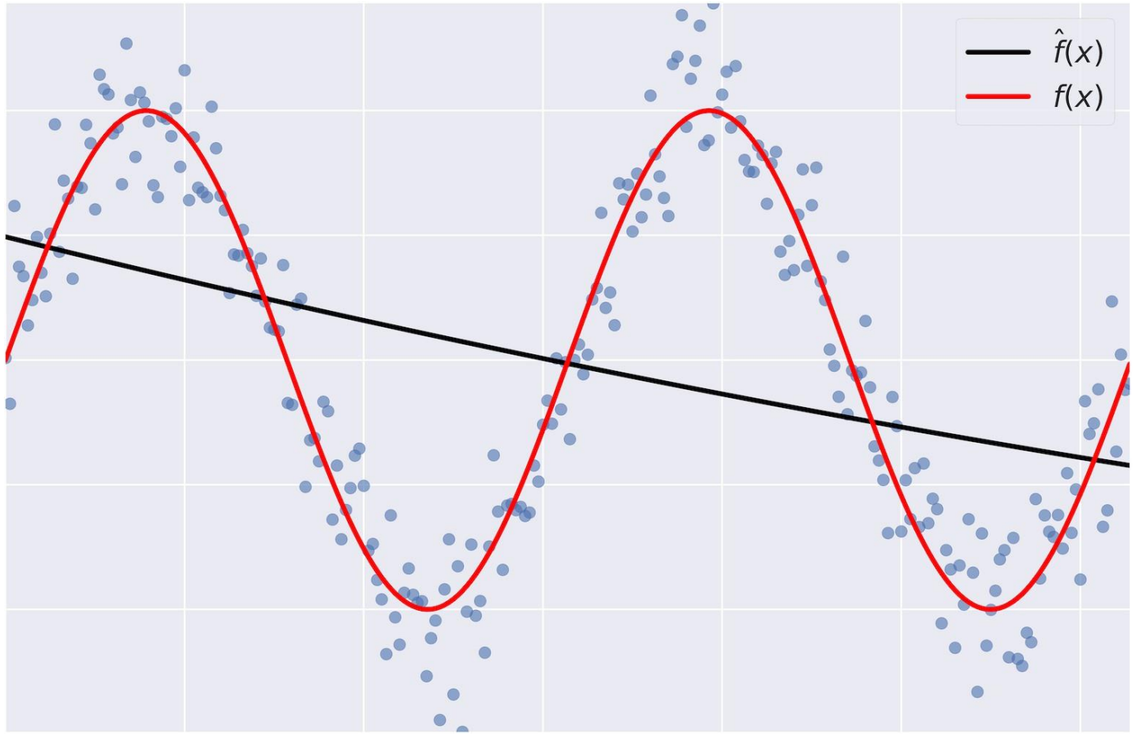
Bias measures the error introduced by approximating a real-world problem (which may be complex) with a simplified model.
High Bias occurs when a model is too simple to capture the underlying structure of the data (e.g., linear regression applied to non-linear data). Results in **underfitting**, where the model performs poorly on both the training and test datasets.
Low Bias means the model is flexible enough to capture the true patterns in the data.

Variance measures the sensitivity of a model to small changes in the training data. A high variance model captures noise in the data along with the underlying patterns.
High Variance occurs when the model is overly complex and too closely fits the training data, including its noise. Results in **overfitting**, where the model performs well on the training data but poorly on unseen data.
Low Variance means the model is stable and consistent across different training datasets.

Irreducible error is the error caused by inherent noise in the data, such as measurement errors or random variability in the process being modeled.
This error cannot be eliminated, regardless of the model used.
Comes from factors outside the model's control, such as unmeasured variables, data inconsistencies, or inherent randomness.

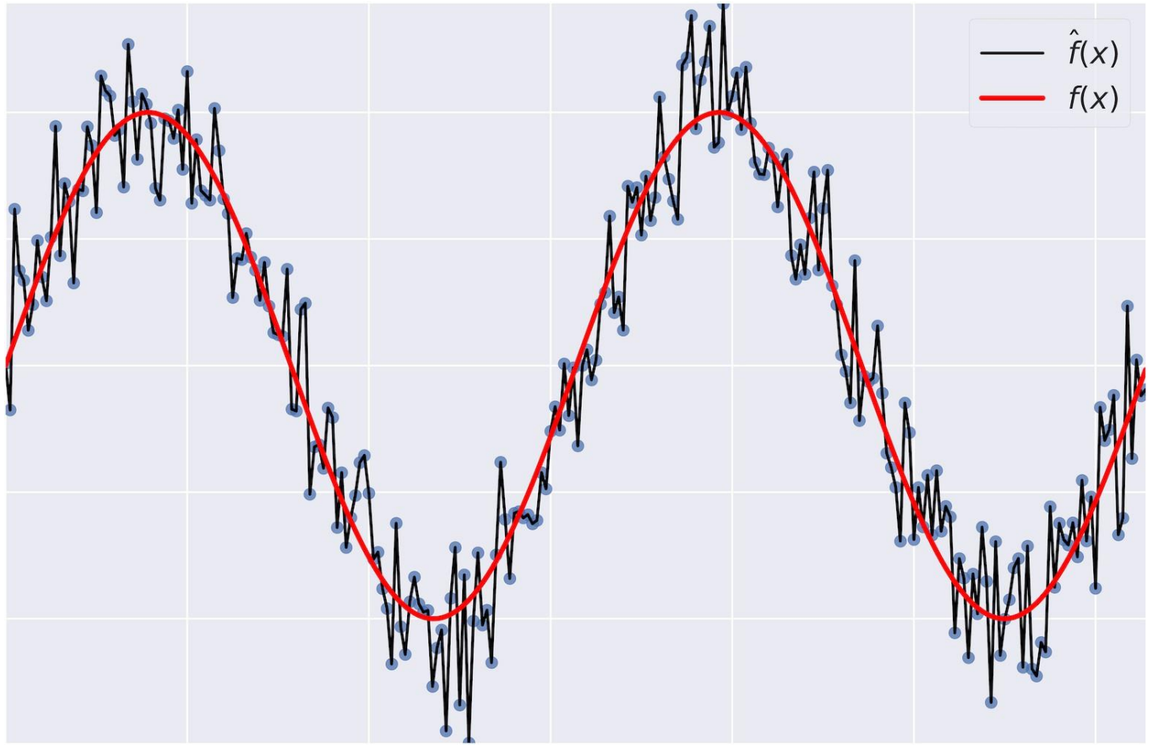
Bias

- **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.



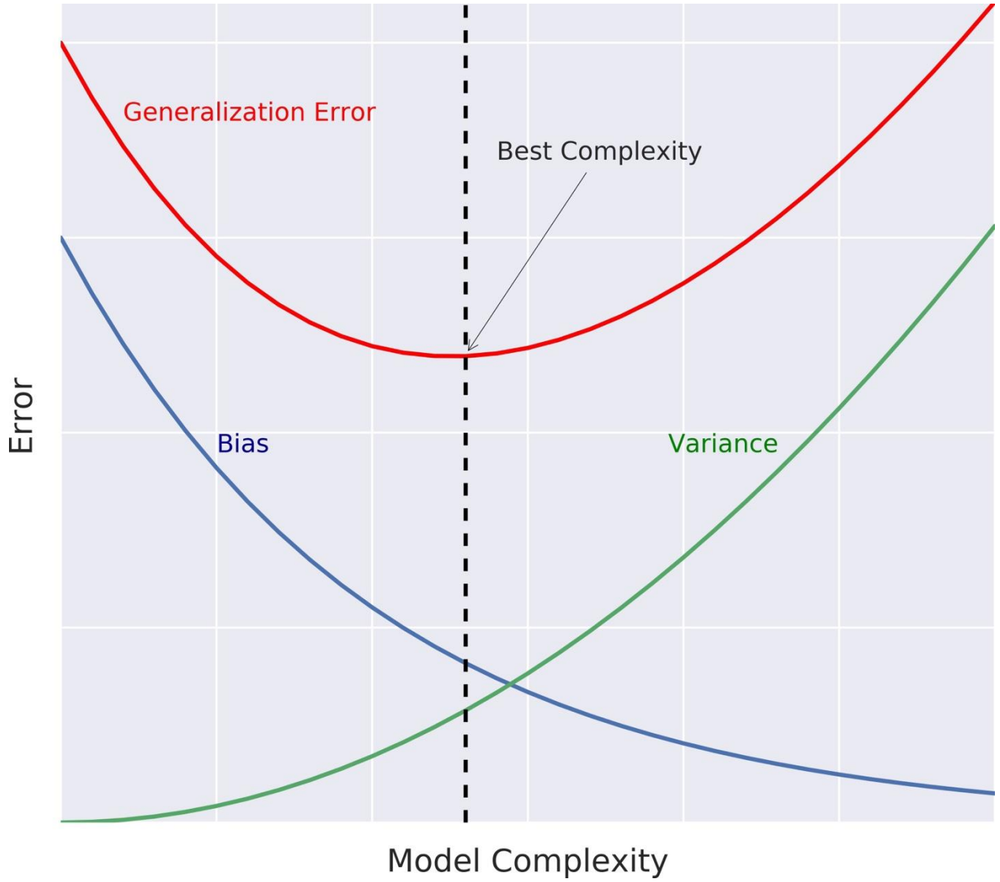
Model Complexity

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

Bias-Variance Tradeoff

The **Bias-Variance Tradeoff** is a fundamental concept in supervised learning that describes the tradeoff between **bias** and **variance**, which are two sources of error in a predictive model.

The tradeoff occurs because reducing one often increases the other, and finding the right balance between bias and variance is critical for building a model that generalizes well to unseen data.

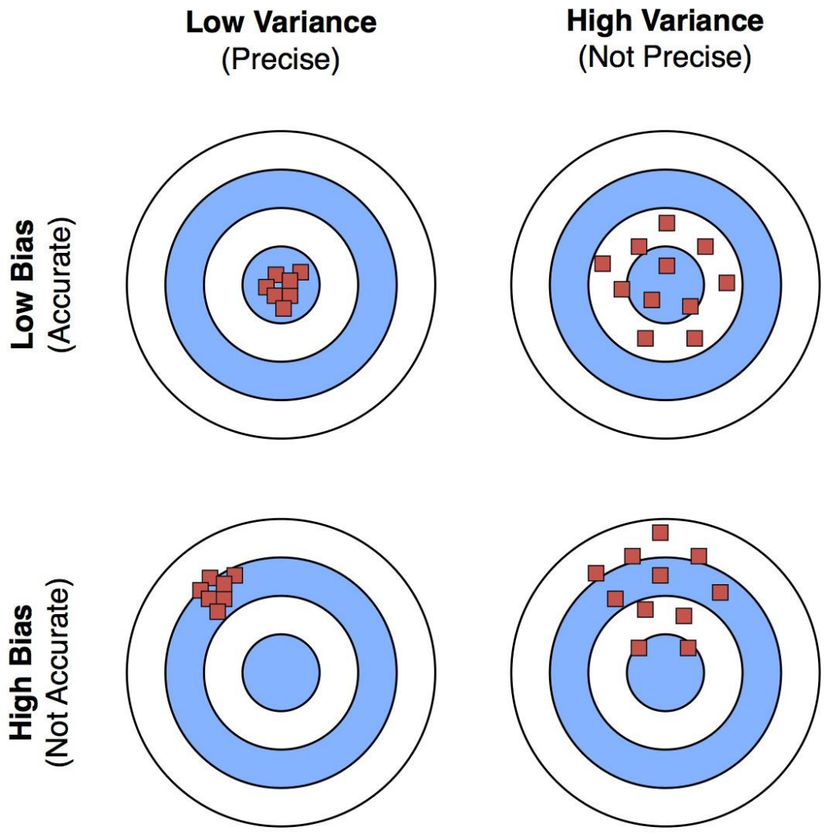


High Bias, Low Variance: A simple model (e.g., linear regression) has high bias because it cannot capture complex patterns, but it is stable and consistent across different datasets (low variance).
Results in **underfitting**.

Low Bias, High Variance: A complex model (e.g., deep neural network) has low bias because it can capture intricate patterns, but it is highly sensitive to the training data, leading to inconsistent predictions (high variance).
Results in **overfitting**.

Balanced Model: The ideal model minimizes both bias and variance, achieving the lowest possible total error while generalizing well to new data.

Bias-Variance Tradeoff: A Visual Explanation



Diagnosing Bias and Variance Problems

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

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)
```

cross_val_score expects that a higher score corresponds to better model performance.
However, Mean Squared Error (MSE) is a **loss function**, where **lower values are better**.
To make MSE compatible with this convention, scikit-learn negates the MSE values during scoring, transforming it into **-MSE**, so higher values indicate better performance.

```
# 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

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.

What Does "Orthogonal Decision Boundaries" Mean?

1.Orthogonal:

1. In the context of CART, "orthogonal" means that splits (decision boundaries) are aligned with the axes of the feature space.
2. Each split divides the data by comparing a feature to a threshold, creating hyperplanes that are perpendicular to the feature axis.

2.Single-Feature-Based Splits:

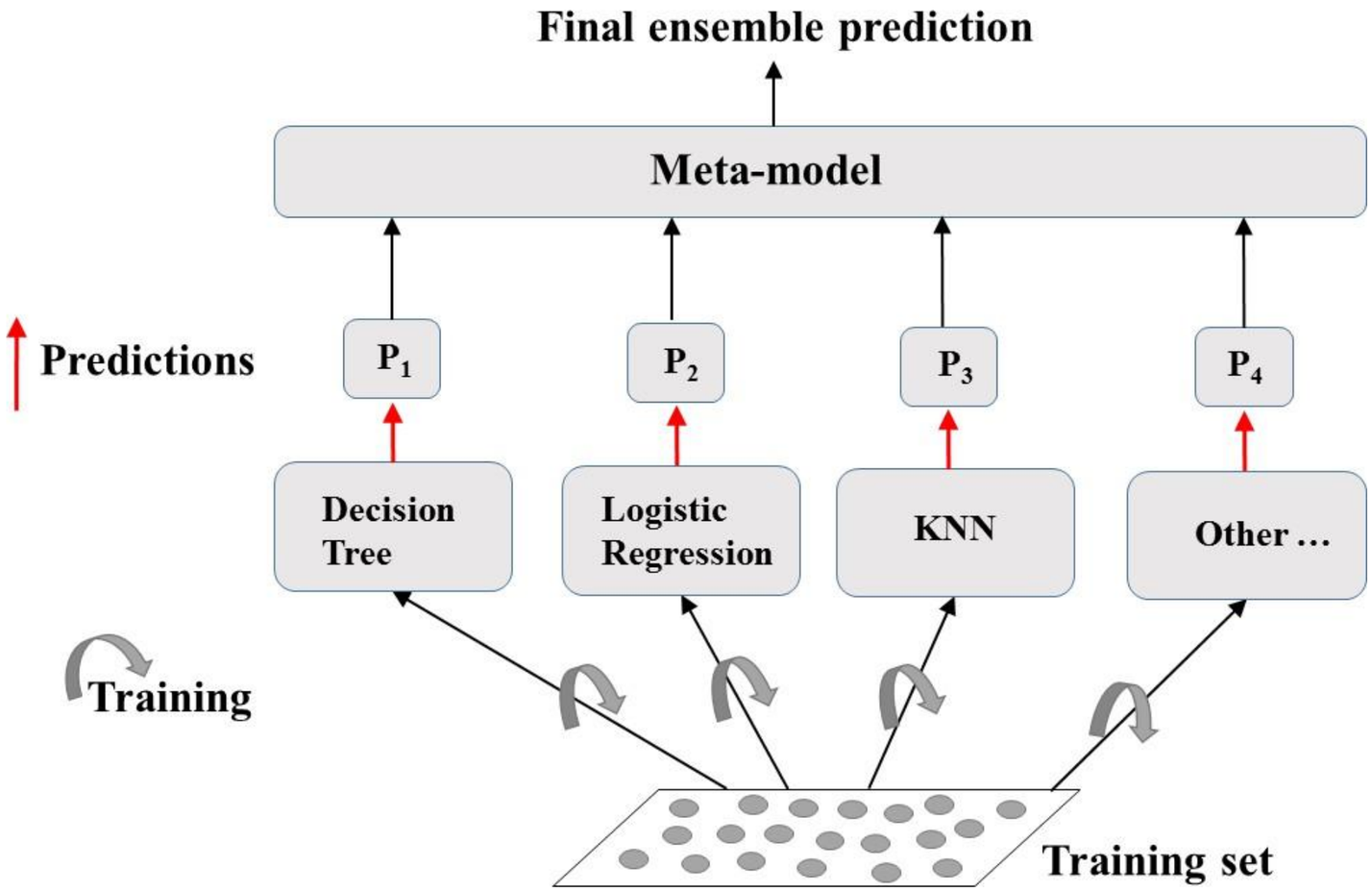
1. CART considers one feature at a time for splitting.
2. The resulting decision regions are rectangular (in 2D) or hyper-rectangular (in higher dimensions).

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 is a machine learning technique where multiple models (often referred to as "weak learners" or "base models") are combined to create a more powerful and robust model, called an **ensemble**. The goal is to improve performance by leveraging the strengths of individual models while minimizing their weaknesses.

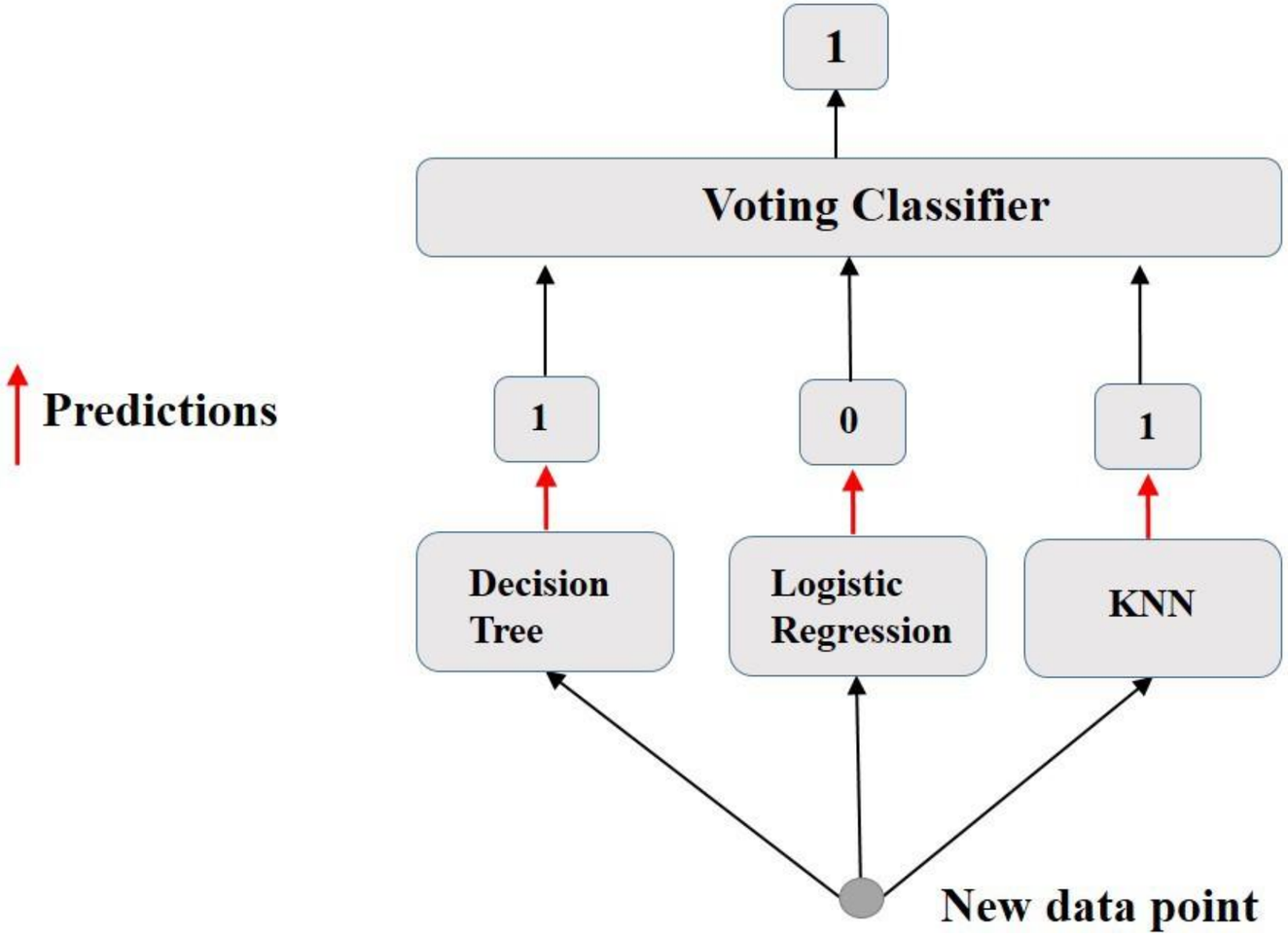
Ensemble Learning: A Visual Explanation



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
```