SE ZG568 / SS ZG568,
Applied Machine Learning
Lecture No. 14 [04- May-2025]

Topic of Discussion

Bias-Variance Trade Off

Definition

Bias: Bias refers to the **error introduced by approximating a real-world problem**, which may be very complex, **by a simplified model**.

A high-bias model is **too simple** to capture the underlying patterns in the data. It makes **strong assumptions**, leading to **systematic errors**.

Example: Trying to fit a straight line to data that clearly follows a curve — the model **underfits**.

Consequences of High Bias:

- Poor performance on training **and** test data
- Fails to capture relevant patterns
- **Underfitting**

Definition

Variance: Variance refers to the model's **sensitivity to fluctuations in the training data**.

Intuition:

A high-variance model is **too complex** and tries to fit every tiny variation in the training data — even noise.

Example:

Fitting a 10th-degree polynomial to just a few data points — the model **overfits**.

Consequences of High Variance:

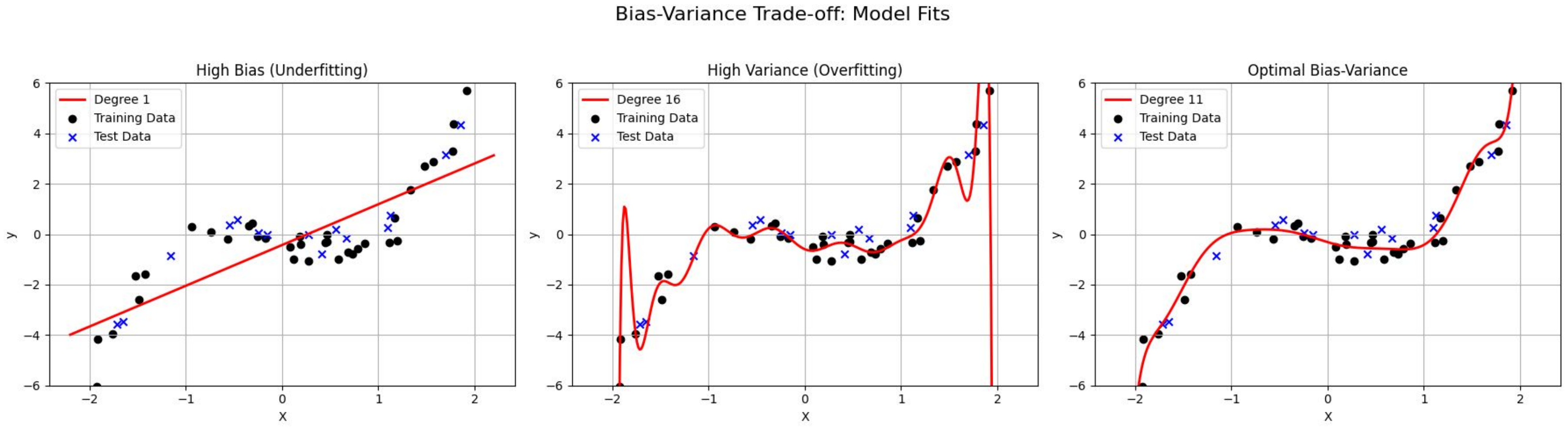
- Excellent performance on training data
- Poor performance on test data
- **Overfitting**

Property	High Bias	High Variance
Model Complexity	Low (too simple)	High (too complex)
Training Error	High	Low
Test Error	High	High (because of overfitting)
Problem	Underfitting	Overfitting

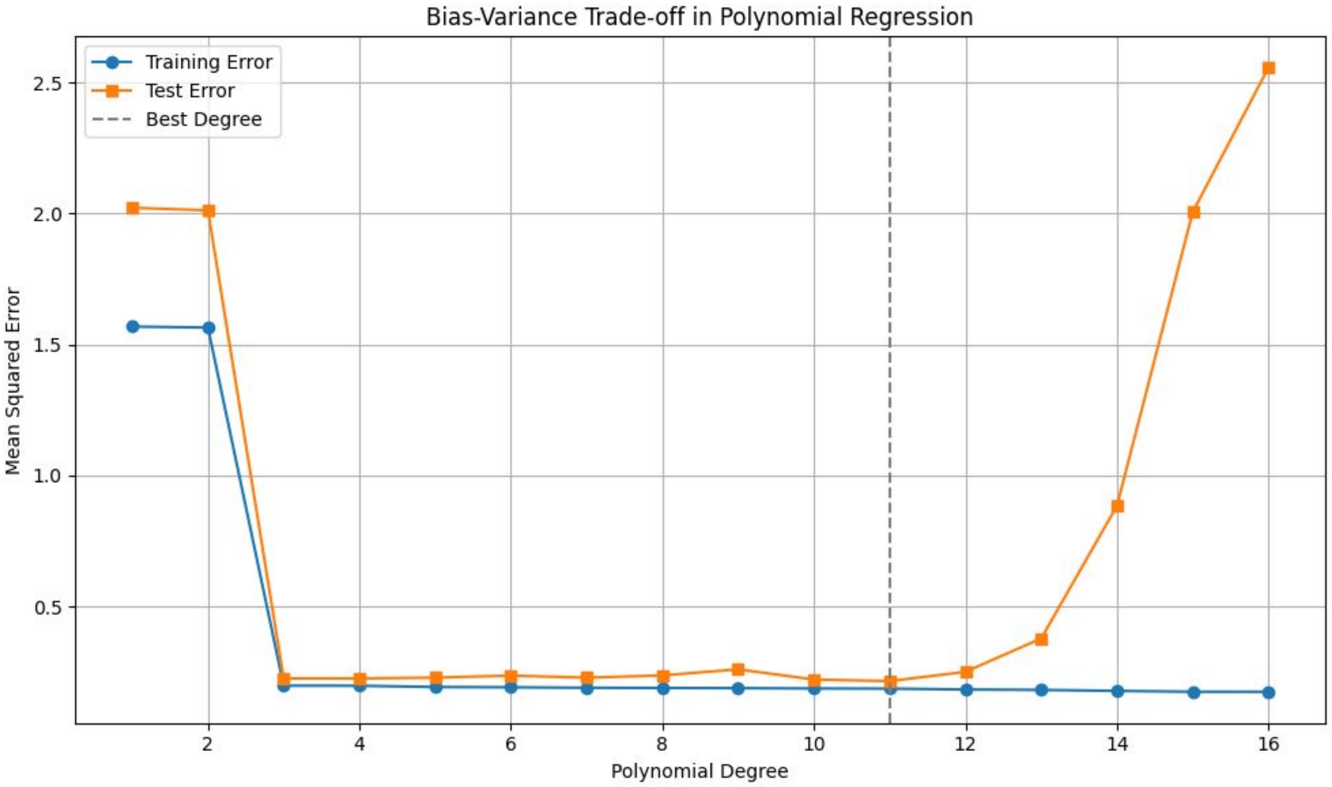
Goal:

Find a model with the **right balance** — low enough bias to capture the signal, and low enough variance to generalize well.

Bias Variance Trade Off



MSE vs Degree of Polynomial



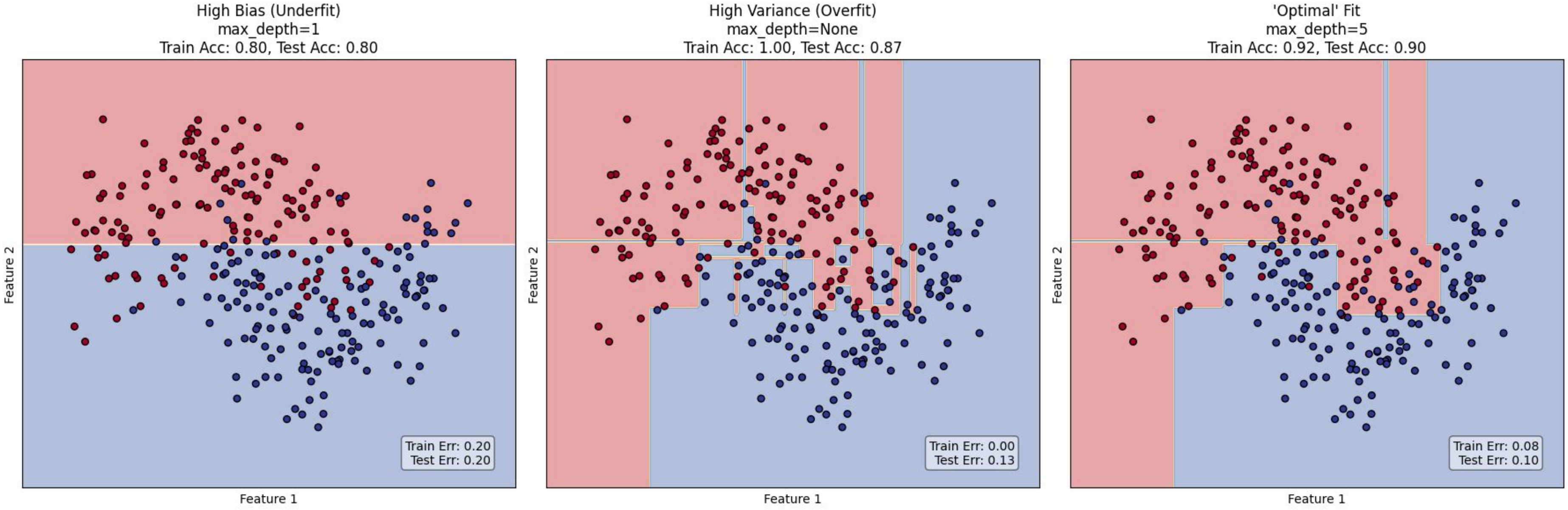
Principled Way of Doing ML

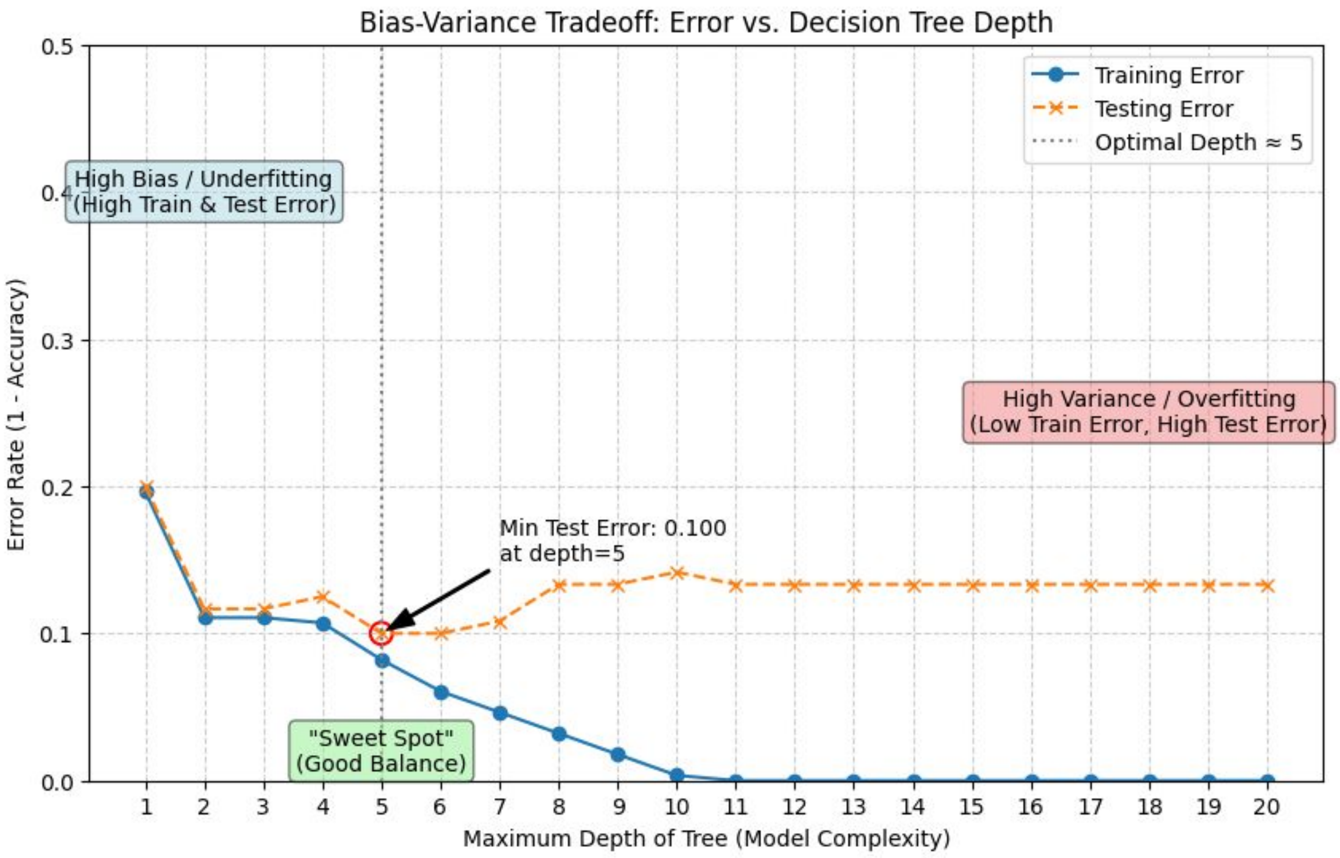
1. Get Data
   - Encode data: Choose a good representation
   - Preprocessing: feature selection, dimensionality reduction
   - random split (Train-Test), fix random seed for reproducibility
2. Generate ***Hypothesis Class***: space of possible solutions ($\mathcal{H}$)
   - make suitable model assumptions (application domain specific)
   - $y = h(x)$ where $h \in \mathcal{H}$ (space of all solutions)
3. **Characterize Loss function: *L(guess, actual)***
4. **Finalize the algorithm – find the best hyperparameters by doing crossvalidation on the training dataset**
5. Run algorithm: Retrain the entire Training data using the best hyperparameter found during cross validation.
6. Validate Result: Testing on unseen data
   - - performance evaluation metrics (Linear Regression: R^2)
   - Classification: Confusion Matrix (Acc., Prec., Recall, Macro F1-score), RoC curves, AuC

Bias-Variance for Decision Tree

Tree Depth	Bias	Variance	Over/Underfit
Small (e.g., 1–3)	High	Low	Underfitting
Medium (e.g., 4–6)	Moderate	Moderate	Good Trade-off
Large (unpruned)	Low	High	Overfitting

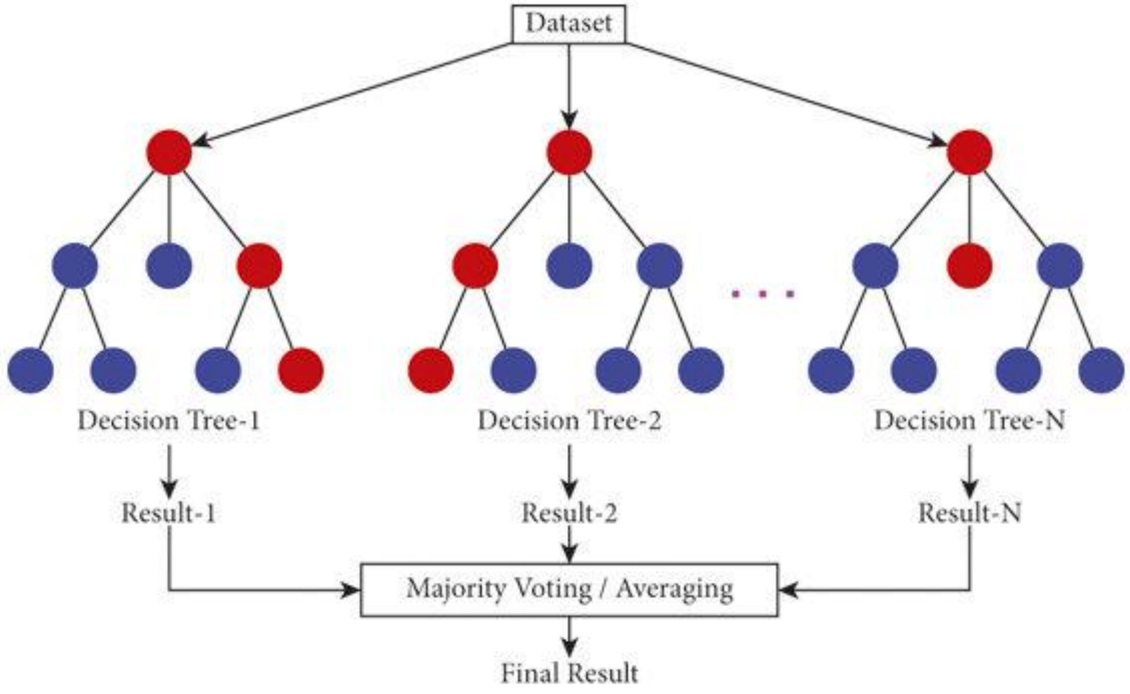
Decision Tree: Bias Variance Trade Off



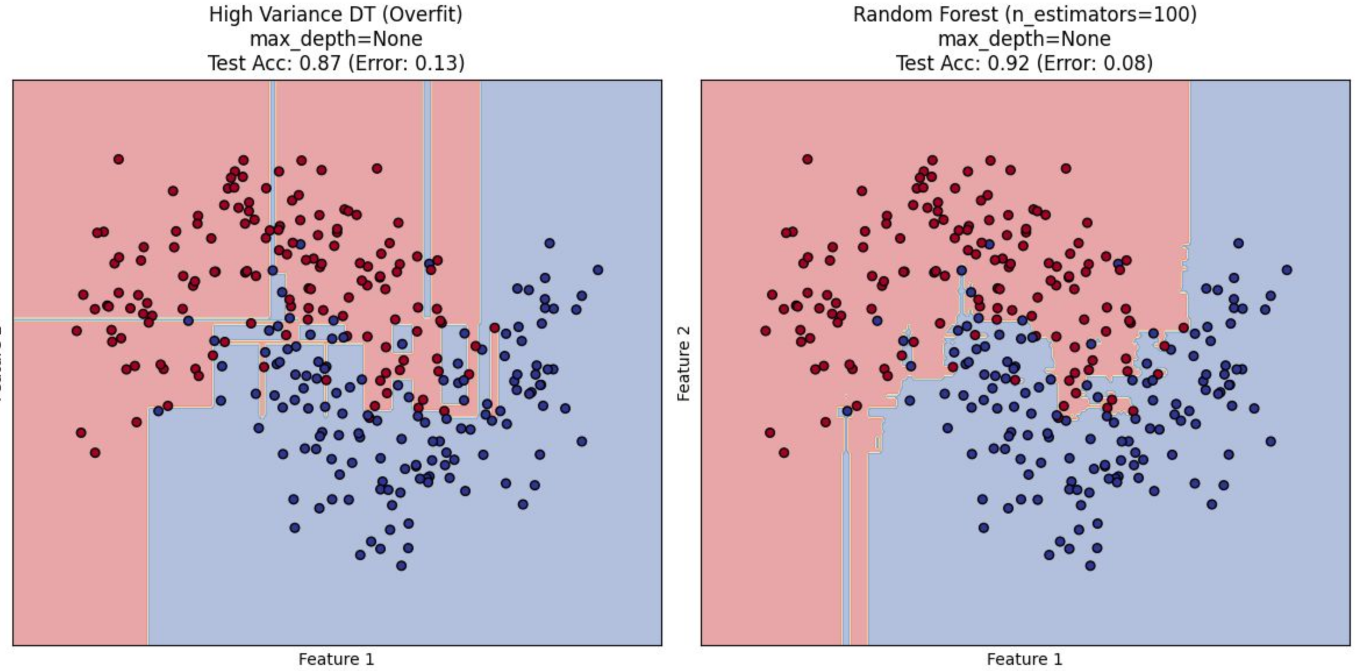


Bias-Variance Tradeoff: Error vs. Decision Tree Depth
Training Error
Testing Error
Optimal Depth ≈ 5
High Bias / Underfitting
(High Train & Test Error)
High Variance / Overfitting
(Low Train Error, High Test Error)
Min Test Error: 0.100
at depth=5
"Sweet Spot"
(Good Balance)
Error Rate (1 - Accuracy)
Maximum Depth of Tree (Model Complexity)
0.0
0.1
0.2
0.3
0.4
0.5
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

Random Forest



Random Forest - Bagging



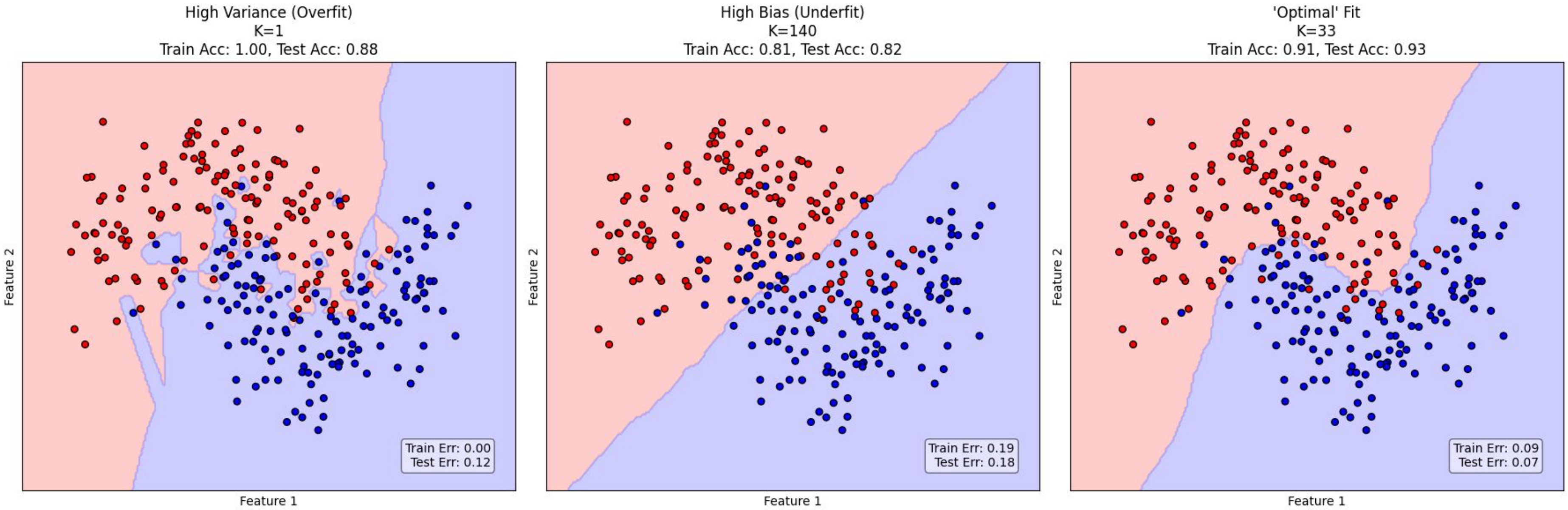
- The single deep Decision Tree (left) has a very jagged, complex boundary, fitting noise.

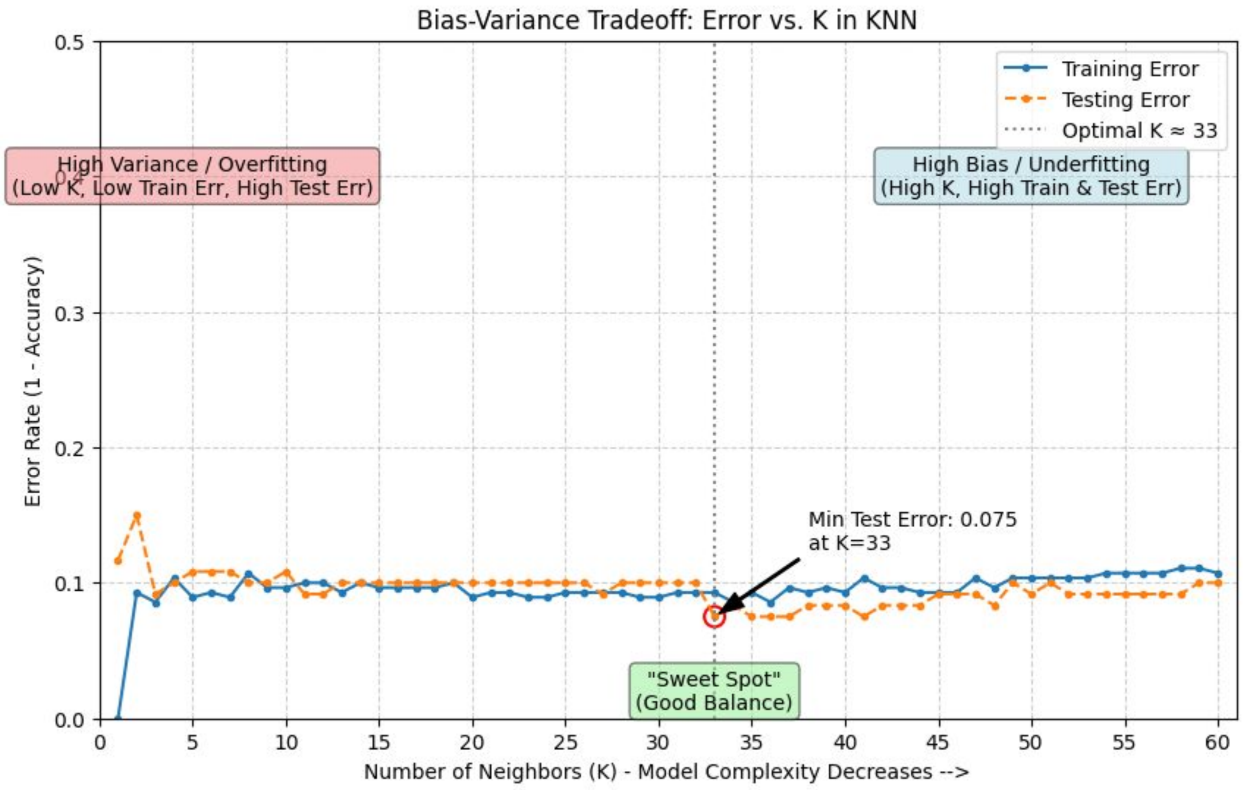
- The Random Forest (right), even though its internal trees are deep, has a much smoother boundary.

- This smoothness comes from averaging the predictions of many trees. It captures the general trend without fitting individual noisy points.

- Consequently, the Random Forest usually achieves better Testing Accuracy (0.92) compared to the severely overfit single tree (0.87), demonstrating reduced variance.

Bias Variance in k NN




Bias-Variance Tradeoff: Error vs. K in KNN
Training Error
Testing Error
Optimal K ≈ 33
High Variance / Overfitting
(Low K, Low Train Err, High Test Err)
High Bias / Underfitting
(High K, High Train & Test Err)
Min Test Error: 0.075
at K=33
"Sweet Spot"
(Good Balance)
Error Rate (1 - Accuracy)
Number of Neighbors (K) - Model Complexity Decreases -->
0.5
0.3
0.2
0.1
0.0
0
5
10
15
20
25
30
35
40
45
50
55
60

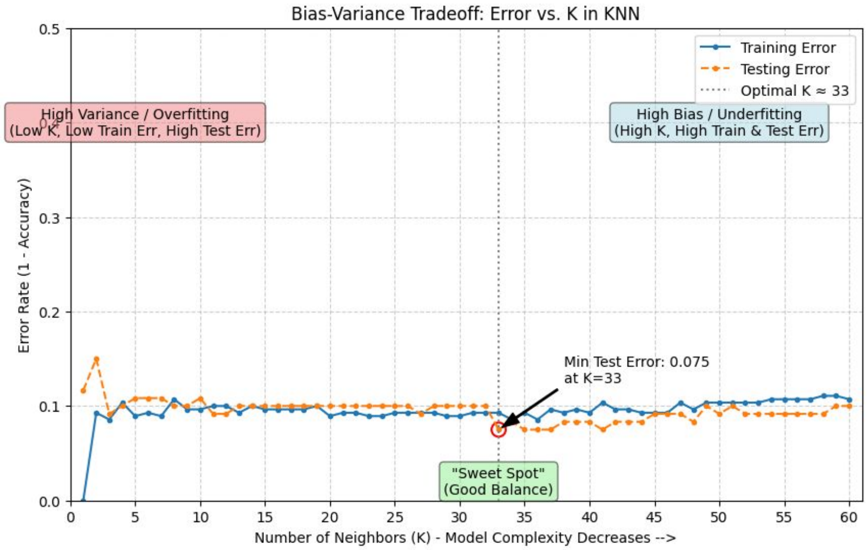
kNN Bias Variance Trade -Off

1. **Low K (Left side, e.g., K=1):**
   - Training Error is very low (often zero for K=1). The model perfectly fits the training data.
   - Testing Error is high.
   - The model is too complex locally (HIGH VARIANCE) and fits the noise (OVERFITTING). It doesn't generalize well.

2. **Increasing K (Moving right):**
   - Training Error generally increases. As K gets larger, the model becomes smoother and cannot fit every training point perfectly.
   - Testing Error decreases initially, reaches a minimum, and then starts to increase again.
   - The point where Testing Error is minimized (around K = 33 in this run) represents the 'optimal' K value, providing the best tradeoff.

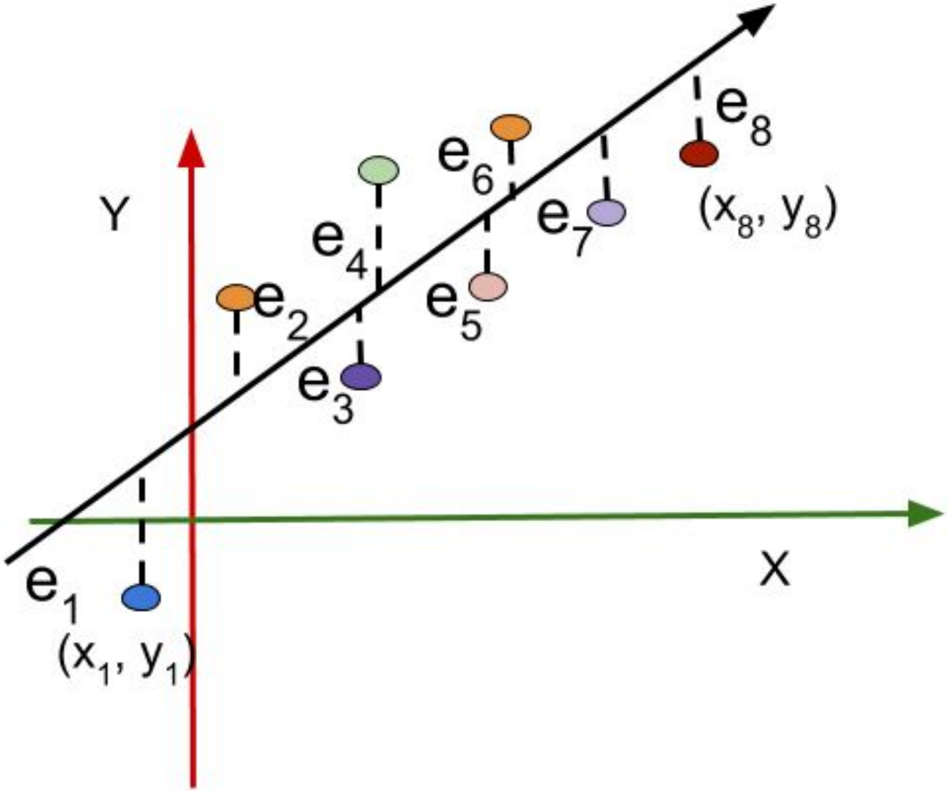
3. **High K (Right side):**
   - Training Error continues to increase and eventually levels off.
   - Testing Error increases after the minimum point.
   - The model becomes too simple (HIGH BIAS) and fails to capture the underlying pattern (UNDERFITTING). The decision boundary becomes overly smooth.

Important Note on Complexity for KNN:
   - Unlike Decision Tree Depth, where higher depth means more complexity, for KNN, *smaller* K means *more* complexity (more flexible boundary),
   - and *larger* K means *less* complexity (smoother, simpler boundary).
   - The goal is still to find the 'sweet spot' that minimizes the Testing Error.

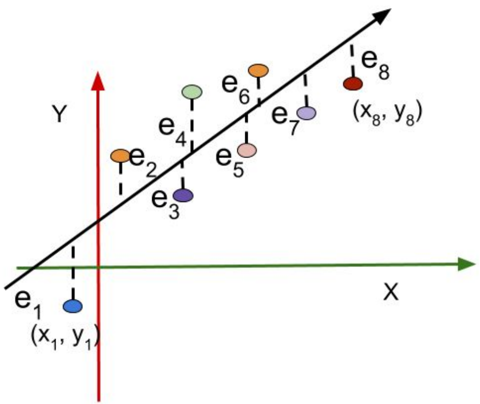


Regularized Least Squares

Linear Least Square Regression



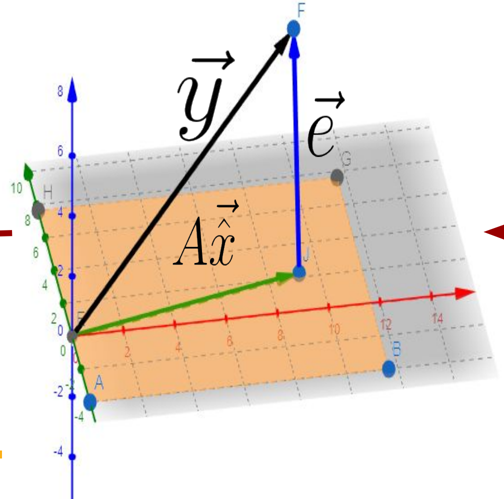
$$y_1 = mx_1 + c + e_1$$
$$y_2 = mx_2 + c + e_2$$
$$y_3 = mx_3 + c + e_3$$
$$y_4 = mx_4 + c + e_4$$
$$y_5 = mx_5 + c + e_5$$
$$y_6 = mx_6 + c + e_6$$
$$y_7 = mx_7 + c + e_7$$
$$y_8 = mx_8 + c + e_8$$



$$
\begin{aligned}
y_1 &= mx_1 + c + e_1\\
y_2 &= mx_2 + c + e_2\\
y_3 &= mx_3 + c + e_3\\
y_4 &= mx_4 + c + e_4\\
y_5 &= mx_5 + c + e_5\\
y_6 &= mx_6 + c + e_6\\
y_7 &= mx_7 + c + e_7\\
y_8 &= mx_8 + c + e_8
\end{aligned}
$$

$$
\begin{bmatrix} y_1\\ y_2\\ y_3\\ y_4\\ y_5\\ y_6\\ y_7\\ y_8 \end{bmatrix}
=
\begin{bmatrix} x_1 & 1\\ x_2 & 1\\ x_3 & 1\\ x_4 & 1\\ x_5 & 1\\ x_6 & 1\\ x_7 & 1\\ x_8 & 1 \end{bmatrix}
\begin{bmatrix} m\\ c \end{bmatrix}
+
\begin{bmatrix} e_1\\ e_2\\ e_3\\ e_4\\ e_5\\ e_6\\ e_7\\ e_8 \end{bmatrix}
$$

$$\vec{y} = A\vec{x} + \vec{e}$$



$$\boxed{\vec{x} = (A^T A)^{-1} A^T \vec{y}}$$

Via Calculus

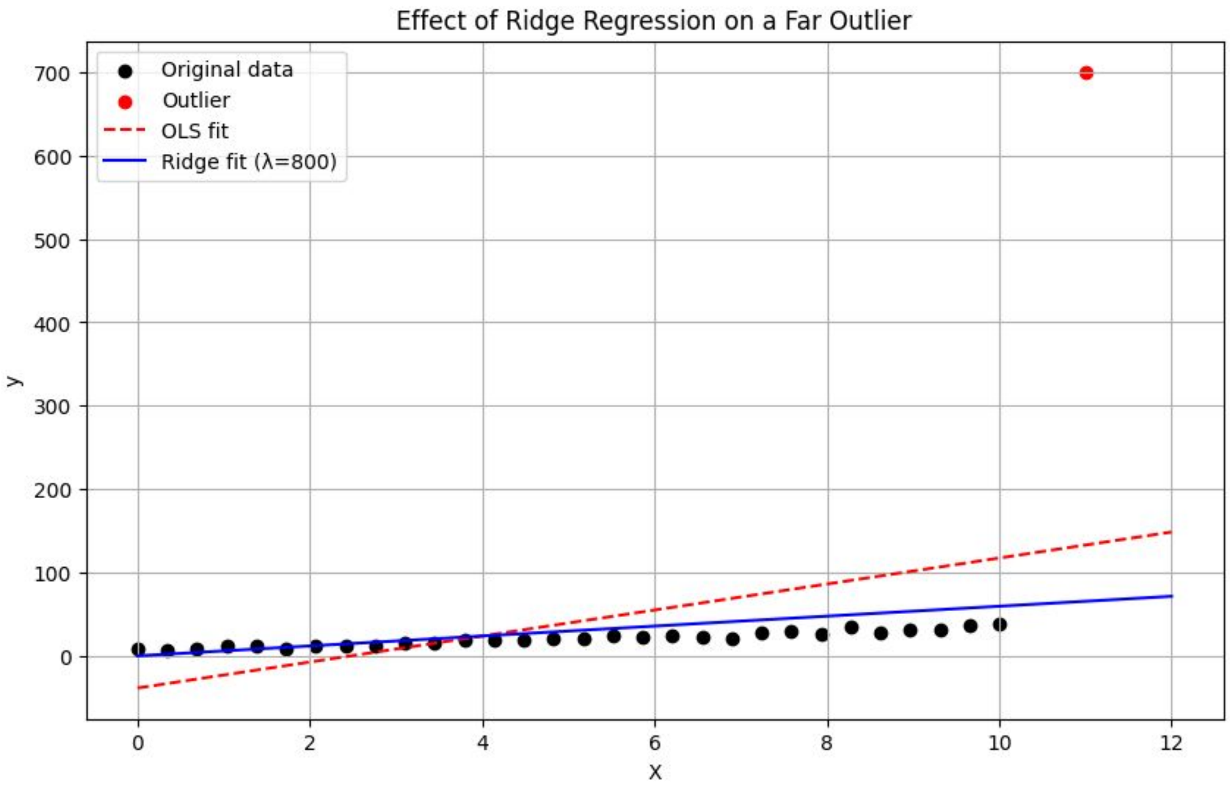
$$\min_x ||Ax - b||_2^2$$

Regularized Least Square [Ridge Regression]

$$\min_{x} ||Ax - b||_2^2 + \lambda ||x||_2^2$$

- **Ordinary Least Squares (OLS)** solution:
  $\hat{w}_{\text{OLS}} = (A^T A)^{-1} A^T b$
- **Ridge Regression** solution:
  $\hat{w}_{\text{ridge}} = (A^T A + \lambda I)^{-1} A^T b$

Effect of Ridge Regression



Principled Way of Doing ML

1. Get Data
   - Encode data: Choose a good representation
   - Preprocessing: feature selection, dimensionality reduction
   - random split (Train-Test), fix random seed for reproducibility
2. Generate ***Hypothesis Class***: space of possible solutions ($\mathcal{H}$)
   - make suitable model assumptions (application domain specific)
   - $y = h(x)$ where $h \in \mathcal{H}$ (space of all solutions)
3. **Characterize Loss function: *L(guess, actual)***
4. **Finalize the algorithm – find the best hyperparameters by doing crossvalidation on the training dataset**
5. Run algorithm: Retrain the entire Training data using the best hyperparameter found during cross validation.
6. Validate Result: Testing on unseen data
   - - performance evaluation metrics (Linear Regression: R^2)
   - Classification: Confusion Matrix (Acc., Prec., Recall, Macro F1-score), RoC curves, AuC