

**Bias**

When we say **"high bias (underfitting)"**, we mean:

The model is **too simple** to understand the real pattern in the data.

**Real-Life Analogy:**

Imagine you’re trying to predict house prices:

* But your model only looks at **square footage** (ignores location, age, etc.).
* It’s **too simple**, so it makes poor predictions.
* That’s **underfitting** due to **high bias**.

**Variance**

**Right Side: Low Bias, High Variance (Overfitting)**

* 🔹 The green **line is very wiggly**.
* 🔹 It tries to **fit every single data point** — even the noise.
* 🔹 It has low bias (it’s flexible) but **high variance** (too sensitive to the data).

➡ **Problem**: Model fits the training data very well but **fails on new/unseen data**.

🧠 This is called **overfitting**.

**1. Parameters**

**Definition**:  
Parameters are the internal values **learned by the model** from the training data.

**Key Points**:

* Automatically learned during training
* Define the model’s predictions
* Change as training progresses

**Examples**:

* **Weights and biases** in neural networks
* **Coefficients** (w, b) in linear regression

**2. Hyperparameters**

**Definition**:  
Hyperparameters are external settings that are **set manually before training** to control how the model learns.

**Key Points**:

* Not learned from data
* Chosen using techniques like grid search or cross-validation
* Can greatly affect model performance

**features** **Selection**

### 1. ****Forward Search (Forward Selection)****

Forward selection is a **wrapper method** used for feature selection. In this approach, you start with **no features** in the model and progressively add features one at a time based on how much they improve the model's performance.

#### Steps:

* **Step 1**: Start with an empty model (no features).
* **Step 2**: Add one feature at a time, testing which feature improves model performance the most.
* **Step 3**: Continue adding features until adding new features no longer improves model performance, or performance begins to degrade.

#### Example:

Given a dataset with features A, B, C, D, and E, forward selection will:

1. Start with an empty model.
2. Add one feature, say A, and test the model's performance.
3. Add the next feature, say B, and test performance.
4. Repeat this process, adding features one by one (C, D, E), checking which gives the best performance at each step.

**Advantages**:

* Simple and easy to understand.
* Can provide a good set of features for a model.

**Disadvantages**:

* Computationally expensive since the model must be retrained at each step.
* May overfit if the number of features is too large.

### 2. ****Backward Search (Backward Elimination)****

Backward elimination is the reverse process of forward selection. In this **wrapper method**, you start with **all features** and progressively remove the least important features based on their contribution to the model's performance.

#### Steps:

* **Step 1**: Start with a model that includes all the features.
* **Step 2**: Evaluate the importance of each feature, often using performance metrics.
* **Step 3**: Remove the feature with the least importance (the one with the smallest effect on model performance).
* **Step 4**: Repeat the process until removing features no longer improves the model or reduces performance.

#### Example:

Given features A, B, C, D, and E:

1. Start with a model using all features.
2. Remove the least significant feature, say E, and test performance.
3. Remove the next least important feature, say D, and test performance.
4. Continue until only the most important features remain.

**Advantages**:

* Often results in better performance since it starts with the full feature set and removes unimportant features.
* Reduces overfitting by eliminating unnecessary features.

**Disadvantages**:

* Computationally expensive since the model is retrained after removing each feature.
* May still leave out important features if they have interactions with other features.

### 3. ****Univariate Feature Selection****

Univariate feature selection is a **filter method** where each feature is evaluated independently using statistical tests. It selects features based on their individual relationship with the target variable,

#### Methods:

* **Chi-square Test**
* **ANOVA (Analysis of Variance)**
* **Correlation**

**Steps:**

1. Apply a statistical test (e.g., chi-square, correlation, or ANOVA) to assess the relationship between each feature and the target.
2. Rank the features based on their test scores.
3. Select the top-ranked features.

**Advantages**:

* Simple and computationally efficient.
* Fast, as each feature is evaluated independently.

**4. Multivariate Feature Selection**

Multivariate feature selection evaluates the relationship between **multiple features** and the target variable simultaneously.

#### Methods:

* **Recursive Feature Elimination (RFE)**
* **Principal Component Analysis (PCA)**
* **Lasso Regression (L1 Regularization)**

1. **Recursive Feature Elimination (RFE)**: Fit a model, eliminate the least important feature, and repeat until the optimal set of features is found.
2. **PCA**: Apply PCA to reduce dimensionality and select the components that contribute the most to variance.
3. **Lasso Regression**: Apply Lasso to shrink coefficients and eliminate less important features.

### What is Reinforcement Learning (RL)?

### **Reinforcement Learning** is a type of **machine learning** where an agent **learns to make decisions** by interacting with an **environment**. It **learns through trial and error**, receiving **rewards or punishments** for its actions, and the goal is to **maximize the total reward** over time.

### Key Components of Reinforcement Learning:

| **Term** | **Meaning** |
| --- | --- |
| **Agent** | The learner or decision-maker (e.g., robot, AI model). |
| **Environment** | The world or system the agent interacts with (e.g., a maze, game, etc). |
| **State** | A snapshot of the environment at a given time. |
| **Action** | What the agent chooses to do. |
| **Reward** | Feedback from the environment — positive or negative. |
| **Policy** | The strategy the agent follows to choose actions. |

## 

## Think of it Like a Game:

Imagine teaching a robot to play a video game:

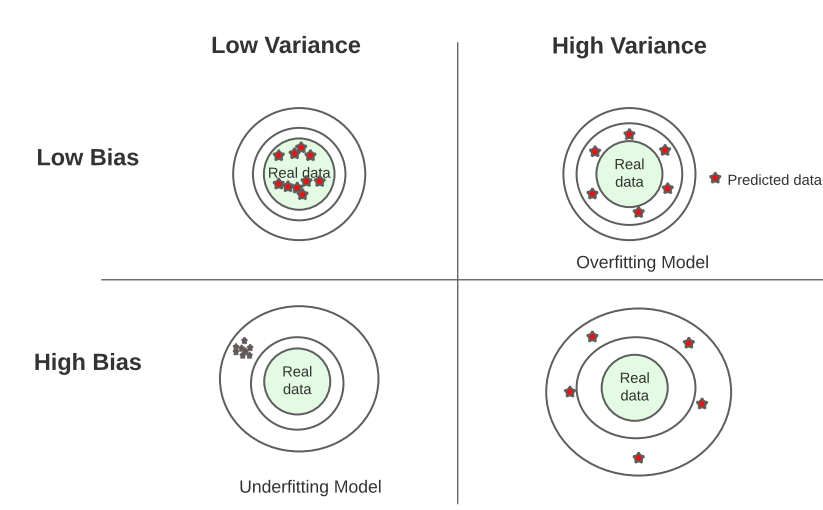
* The robot can press buttons (take **actions**).
* It sees what happens on the screen (receives an **observation** of the **environment**).
* If it wins points, it gets a **reward**.
* If it makes a wrong move, it loses points or gets a **penalty**.
* The robot **learns from these results** and keeps improving its gameplay.

## Example: Self-driving Car

* **Agent**: The car’s AI system.
* **Environment**: The road, traffic, signs, etc.
* **Action**: Accelerate, brake, turn left, turn right.
* **Reward**: Positive reward for staying in the lane, negative reward for crashing or speeding.

## 🔍 Types of Reinforcement Learning

| **Type** | **Description** |
| --- | --- |
| **Positive Reinforcement** | Reward is given for correct actions. Encourages repeating those actions. |
| **Negative Reinforcement** | Removes negative conditions for good actions (e.g., stops alarm when correct move is made). |
| **Punishment** | Penalty is given for bad actions. Helps the agent learn to avoid them. |



### 🎯 Diagram Overview:

Each dot (★) represents **predicted values** from your model.  
The center of the target ("Real data") is the **true value** we want to predict.

### 📌 Top Left: ****Low Bias, Low Variance**** ✅

* **Darts are tightly grouped and close to the center**.
* The model's predictions are both **accurate** (low bias) and **consistent** (low variance).
* ✅ **Best scenario** – model generalizes well to new data.

### 📌 Top Right: ****Low Bias, High Variance**** ⚠️ Overfitting

* **Darts are spread out but centered around the target**.
* The model is **accurate on training data** but **inconsistent** on new data (high variance).
* ⚠️ **Overfitting**: the model memorizes the training data and fails on unseen data.

### 📌 Bottom Left: ****High Bias, Low Variance**** ⚠️ Underfitting

* **Darts are tightly grouped but far from the center**.
* The model is **consistently wrong** – makes the same mistake every time.
* ⚠️ **Underfitting**: the model is too simple and can’t learn the pattern.

### 📌 Bottom Right: ****High Bias, High Variance**** ❌

* **Darts are scattered and far from the center**.
* The model is **inaccurate** and **inconsistent**.
* ❌ **Worst case** – model neither learns well nor generalizes.

**Clustering method**

The **Elbow Method** is a popular technique used in **clustering** (especially in **K-Means**) to help decide the **optimal number of clusters (K)**.

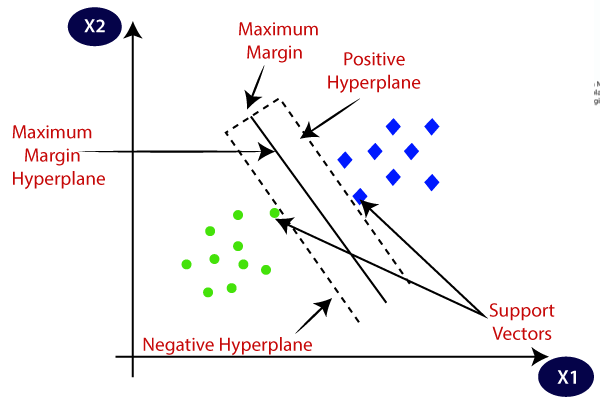
It helps you answer:

"How many clusters should I choose?"

It does this by plotting the **within-cluster sum of squared errors (WCSS)** against different values of **K**.

## 📉 How It Works:

1. Run **K-Means clustering** for different values of **K** (e.g., from 1 to 10).
2. For each K, calculate the **WCSS**:
   * WCSS = sum of squared distances from each point to its cluster center
3. Plot **K vs. WCSS**.
4. Look for the **“elbow point”** – the value of K **after which WCSS starts to decrease slowly**.



| **Element** | **Meaning** |
| --- | --- |
| **Purple circles** | One class (Class -1) |
| **Pink diamonds** | Another class (Class +1) |
| **Maximum Margin Hyperplane** | The main solid line that separates the two classes. It is placed in such a way that it maximizes the distance from the nearest points of both classes. |
| **Positive Hyperplane** | Dashed line passing through the closest pink point(s); all pink points should lie on or beyond this. |
| **Negative Hyperplane** | Dashed line passing through the closest purple point(s); all purple points should lie on or beyond this. |
| **Support Vectors** | Points lying **exactly on the dashed hyperplanes**. These are the critical points — they define the position and orientation of the maximum margin hyperplane. |
| **Maximum Margin** | The distance between the **positive and negative hyperplanes** — this is what SVM tries to maximize. |