# Module 3

## Other Supervised Learning Models

## 📌 Introduction to Supervised Learning

Supervised learning involves training models on labeled data to predict outcomes for new, unseen data. The key objective is to minimize errors while ensuring generalization to new datasets, develop models that can generalize well to unseen data.

In Module 3, we focus on:

* **K-Nearest Neighbors (KNN)** – An instance-based algorithm that classifies data based on similarity.
* **Support Vector Machines (SVMs)** – A powerful classification algorithm that finds the optimal decision boundary.
* **Bias-Variance Tradeoff** – Understanding and balancing model complexity.
* **Ensemble Learning (Bagging & Boosting)** – Improving model performance by combining multiple classifiers.

## 📌 Support Vector Machines (SVM)

### 🔹 What is Support Vector Machines (SVM)?

Support Vector Machines (SVMs) are supervised learning algorithms used for both classification and regression, with a primary focus on finding the optimal decision boundary that best separates different classes in a dataset.

✔ SVM is powerful for **high-dimensional** spaces where clear class separation is required.

✔ It **works well for both linearly and non-linearly separable data** by mapping data into higher dimensions using kernel functions.

✔ It aims to maximize the margin between data points belonging to different classes.

Unlike traditional classifiers like KNN or Logistic Regression, SVM does not rely on probability scores but instead finds the most optimal hyperplane that distinctly classifies data points.

Use SVM when:

✔ **Use SVM for small- to medium-sized datasets with complex, non-linear boundaries**.  
✔ **Use SVM when interpretability is not a major concern** but accuracy is the priority.  
✔ **Avoid SVM for extremely large datasets due to high computational costs.**

### 🔹 How SVM works

SVM classifies data by finding the best possible hyperplane that separates different classes while maximizing the margin (distance between the closest data points of different classes, known as support vectors).

Key Concept: Support Vectors

✔ Support Vectors are the closest data points to the hyperplane.

✔ These points define the margin and influence the decision boundary.

✔ The model does not consider other data points when forming the boundary, making SVM highly robust to noise.

1. **Identifying the Optimal Hyperplane**

* In two-dimensional space, the hyperplane is simply a line that separates data points.
* In three-dimensional space, the hyperplane is a plane that divides the dataset.
* In higher-dimensional spaces, the hyperplane is a mathematical decision boundary.

1. **Hard Margin vs Soft Margin**

✔ Hard Margin SVM (Strict separation)

* + The model assumes that all data points can be perfectly separated by a linear boundary.
  + Works well for **clean**, noise-free datasets.
  + **Limitations** → If classes overlap even slightly, the model fails completely

✔ Soft Margin SVM (Tolerates misclassifications)

* + Introduces a flexibility parameter (C) that allows some misclassification to balance generalization.
  + The lower the C parameter, the more misclassification is allowed (higher bias, lower variance).
  + The higher the C, the stricter the boundary (lower bias, higher variance).

✔ In real-world applications, Soft Margin SVM is preferred as real-world data is rarely perfectly separable.

1. **Kernel Trick: Handling Non-Linearly Separable Data**

In many datasets, a simple linear boundary cannot separate classes. To solve this, SVM uses kernel functions to project data into higher dimensions, making it easier to separate.

Common Kernels in SVM

✔ **Linear Kernel:**

* + Works well if data is linearly separable (can be separated by a straight line).
  + **Fast and efficient for simple problems.**
  + The decision function is simply:

✔ **Polynomial Kernel:**

* + Suitable when **curved decision boundaries** are required.
  + **Captures complex interactions between features.**
  + The Kernel function:

✔ **Radial Basis Function (RBF) Kernel:**

* + Maps data into **infinite-dimensional** **space**, making even complex datasets separable.
  + **The most commonly used kernel for handling non-linearly separable data.**
  + The function:

✔ **Sigmoid Kernel:**

* + Mimics neural networks and is used for **binary classification problems**.
  + The function:

✔ **Choosing the right Kernel:**

* + Use **Linear** **Kernel** if the dataset is linearly separable.
  + Use RBF Kernel for complex, high-dimensional problems.
  + Use Polynomial Kernel if interactions between features matter.

1. **Kernel Trick: Handling Non-Linearly Separable Data**

To improve SVM performance, two **main hyperparameters** need tuning:

✔ **C (Regularization Parameter):**

* + Controls the tradeoff between bias and variance.
  + Higher C → Stricter separation, lower bias, but more prone to overfitting.
  + Lower C → More flexible boundary, higher bias, but better generalization.

✔ **Gamma (γ Parameter for RBF Kernel):**

* + Controls the influence of each training example.
  + Higher γ → Model becomes too sensitive to individual points, leading to overfitting.
  + Lower γ → SVM fails to capture patterns properly.

✔ **Using Grid Search for Hyperparameter Tuning:**

* + Hyperparameters C and gamma should be fine-tuned using cross-validation to find the best combination.

## 📌 K-Nearest Neighbors (KNN)

### 🔹 What is K-Nearest Neighbors (KNN)?

K-Nearest Neighbors (KNN) is a non-parametric, instance-based learning algorithm used for both classification and regression. Unlike other machine learning algorithms that explicitly learn a model during training, KNN memorizes the dataset and makes predictions based on similarity.

KNN assumes that data points close to each other share similar properties and that classification can be determined based on proximity to labeled examples.

✔ It is a **lazy learning algorithm**, meaning no actual training phase occurs—instead, computations happen at the time of prediction.

✔ It is **simple and effective**, working well with small datasets and low-dimensional feature spaces.

✔ It **does not assume any distribution** about the data, making it useful for non-linear decision boundaries.

Use KNN when:

✔ Use KNN when the dataset is small and interpretability is needed.

✔ Avoid KNN for large datasets due to slow prediction speed.

### 🔹 How NN Works

The core principle of KNN is that it assigns a label (classification) or predicts a value (regression) for a new data point based on its K closest neighbors in the training set.

The process involves four key steps:

1. Choose a value for K (the number of nearest neighbors to consider).
2. Calculate the distance between the new data point and all training data points.
3. Find the K-nearest neighbors (smallest distances).
4. Make a prediction:
   * For classification → Assign the most frequent class among the K neighbors (majority vote).
   * For regression → Take the average (or weighted average) of the target values of the K neighbors.

✔ The performance of KNN depends on choosing an optimal K value and an appropriate distance metric.

### 🔹 Choosing the Right K Value

Selecting the optimal K value is a crucial step in achieving a well-performing KNN model. The choice of K impacts the bias-variance tradeoff:

* Small K (e.g., K=1, K=3) → Leads to Overfitting

✔The model becomes too sensitive to noise and classifies based on just one or two points.

✔High variance and low bias (too flexible).

* Large K (e.g., K=20, K=50) → Leads to Underfitting

✔The model becomes too general and fails to capture local structures in data.

✔Low variance and high bias (too rigid).

* Optimal K → Usually found via cross-validation.

### 🔹 Finding the Optimal K Using Accuracy Analysis

We determined the best value of K by training multiple KNN models with different values of K and evaluating their accuracy on a test set.

The process followed these steps:

1. **Define a range of K values to test**
   * We started with a range of small to large values of K (e.g., from 1 to 50).
   * The goal was to observe how accuracy changes with increasing K.
2. **Train multiple KNN models**
   * A separate KNN model was trained for each selected K value.
   * Each model made predictions on the test set and the accuracy was recorded.
3. **Plot accuracy vs. K values**
   * The accuracy scores were plotted against the corresponding K values.
   * This helped visualize how accuracy changes with different K values.
4. **Identify the best K**
   * The optimal K was chosen as the one that maximized accuracy while maintaining stability.
   * If accuracy fluctuated significantly, we selected a slightly larger K for smoother performance.

**Interpreting the Results:**

* If the accuracy is highest at very low K values (K=1, K=3), the model is likely overfitting to the training data.
* If the accuracy remains stable for mid-range K values (K=5 to K=15), these values are usually good choices.
* If the accuracy declines for large K values (K > 30), the model is underfitting, failing to capture important decision boundaries.

✔ The best practice is to select the K value that provides the highest accuracy while preventing excessive fluctuations due to noise.

### 🔹 Distance Metrics in KNN

Since KNN relies on measuring distances, selecting the right distance metric is critical for performance.

1. **Euclidean Distance (Most Common)**

The default metric, representing the shortest straight-line distance between two points:

✔ Works well with continuous numerical data.

✔ Assumes all features are equally important.

1. **Manhattan Distance (City Block Distance)**

The sum of absolute differences between features values

✔ More robust for high-dimensional data.

✔ Works well when features have different units of measurement.

1. **Minkowski Distance (Generalized Distance Metric)**

A generalization of Euclidean and Manhattan distances:

* + When **p = 1**, Minkowski reduces to **Manhattan Distance**.
  + When **p = 2**, Minkowski reduces to **Euclidean Distance**.

✔ Allows flexibility in distance calculations.

### 🔹 Handling Imbalanced Classes in KNN

KNN can be biased towards the majority class if the dataset is imbalanced.

To address this, we can:

✔ **Use Weighted KNN** → Assign higher weights to closer neighbors, reducing the influence of distant points.

✔ **Use Data Resampling** → Oversample the minority class or undersample the majority class.

Weighted KNN formula:

where:

* + is the inverse distance weight .

✔ This approach ensures that closer neighbors contribute more to predictions.

### 🔹 Handling Feature Scaling in KNN

Since distance-based algorithms are sensitive to feature magnitudes, feature scaling is essential.

✔ Min-Max Scaling → Scales data to the [0,1] range.

✔ Standardization (Z-score Normalization) → Centers data with mean μ=0 and standard deviation 𝜎 = 1.

✔ Feature scaling ensures that all features contribute equally to distance calculations.

## 📌 Bias-Variance Tradeoff.

### 🔹 What is the Bias-Variance Tradeoff?

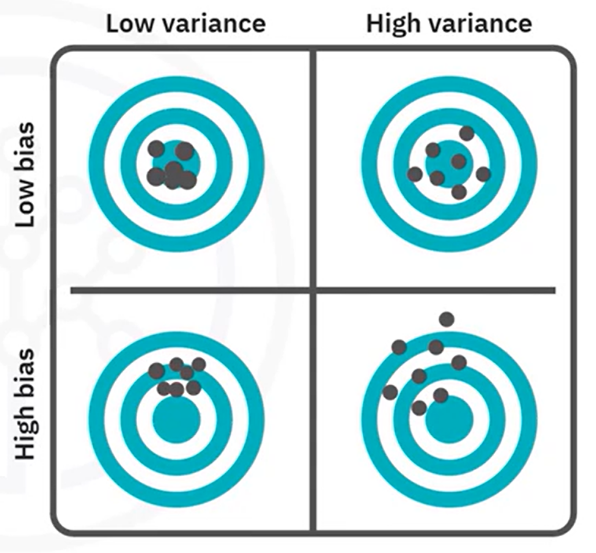
The **Bias-Variance Tradeoff** is a **fundamental concept in machine learning** that describes the **tension between model complexity and generalization**. The goal is to develop a model that **performs well on both training and unseen test data**, meaning it **neither underfits nor overfits**.

When a model has high bias, it oversimplifies the data and fails to learn meaningful patterns.

When a model has high variance, it memorizes the training data instead of generalizing patterns.

✔ **Bias** measures how well the model represents the true relationship between inputs and outputs.  
✔ **Variance** measures how sensitive the model is to variations in training data.

A model with **high bias** tends to underfit the data, missing important patterns.  
A model with **high variance** tends to overfit the data, capturing noise instead of actual patterns.



Think of bias as how on-target or off-target the darts are.

Variance measures how spread out the darts are, representing precision.

The top two boards demonstrate low bias, meaning they are more accurate, while the bottom two show higher bias, making them less accurate. Think of bias as how on-target or off-target the darts are.

The dart boards on the right display higher variance, meaning the darts are more spread out, while the boards on the left show lower variance, with the darts grouped closer together.

As shown on the top left board, achieving a high score requires both low bias for accuracy and low variance for precision.

### 🔹 What is Bias?

Bias represents systematic errors introduced when a model makes strong assumptions about the data’s structure. It is the measure of how far off the model’s predictions are from the true values.

✔ **High Bias (Underfitting)**

* + The model is too simple and fails to capture the underlying pattern in the data.
  + It assumes a rigid structure (e.g., trying to fit a straight line to a highly curved dataset).
  + Leads to poor accuracy on both training and test data.

✔ **Characteristics of High-Bias Models**

* + Fails to learn from training data.
  + High training error and high-test error.
  + Predictions do not change much when training on different datasets.

✔ **Examples of High Bias Models**

* + Linear Regression on non-linear data.
  + A decision tree with only one or two splits.
  + A low-degree polynomial regression model trying to fit complex data.

### 🔹 What is Bias?

Variance refers to how much a model’s predictions change when trained on different subsets of the data. A high-variance model is too complex and captures noise instead of the true pattern.

✔ High Variance (Overfitting)

* + The model memorizes training data, making it highly sensitive to small variations.
  + Works extremely well on training data but fails to generalize to unseen data.
  + Any small fluctuations in the dataset can drastically change predictions.

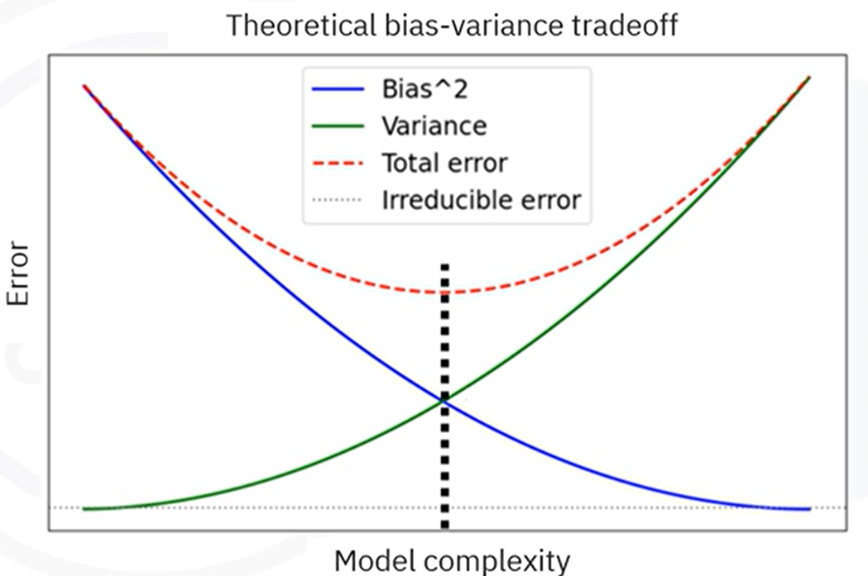
✔ Characteristics of High-Variance Models

* + Very low training error but high test error.
  + Highly sensitive to small changes in the dataset.
  + Predictions change drastically with different training samples.

✔ Examples of High-Variance Models

* + A deep decision tree without pruning.
  + KNN with K=1 (memorizing every training point).
  + A high-degree polynomial regression model fitting every data point exactly.

### 🔹 Balancing Bias and Variance

This plot illustrates how bias and variance changes as the model becomes more complex and better at predicting the data it’s trained on.

As model complexity increases, bias, represented by the blue curve, tends to decline, while variance, shown by the green curve, rises.

When model complexity is low, bias is high, leading to poor predictions even on training data. This is known as underfitting.

Conversely, high model complexity results in high variance, meaning the model becomes overly sensitive to the training data and performs poorly on unseen data, resulting in overfitting.

However, there's a crossover point marked by the vertical dashed line where the model's complexity is just right.

There will always be some generalization error that cannot be eliminated, such as random noise in the data.

A model must find balance between **bias and variance** to **generalize well** on new data.

✔ A good model is one that finds the right balance between bias and variance.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model Complexity | Bias | Variance | Training Error | Test Error |
| Simple Model (Underfitting) | 🔴 High | ✅ Low | ❌ High | ❌ High |
| Balanced Model (Good Fit) | ✅ Low | ✅ Low | ✅ Low | ✅ Low |
| Overly Complex Model (Overfitting) | ✅ Low | 🔴 High | ✅ Very Low | ❌ High |

## 📌 Bias-Variance and Ensemble Learning

### 🔹 What is Ensemble Learning?

Ensemble Learning is a **machine learning technique** that **combines multiple models** to produce a **stronger, more robust predictive model**. Instead of relying on a **single classifier**, ensemble learning aggregates predictions from multiple weak models, improving **generalization, accuracy, and stability**.

✔ **It mitigates the bias-variance tradeoff** by balancing model complexity.  
✔ **It reduces overfitting (high variance) and underfitting (high bias).**  
✔ **It is particularly useful for improving weak learners** such as decision trees.

### 🔹 How Bias and Variance Relate to Ensemble Learning

Bias and variance contribute **differently** to model performance. **Ensemble methods help control them** by combining multiple models.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ensemble Method | |  | | --- | |  |   Reduces Bias? | Reduces Variance? | |  | | --- | |  |   How It Works |
| Bagging (Bootstrap Aggregating) | ❌ No | ✅ Yes | Averages multiple independent models to reduce overfitting. |
| Boosting | ✅ Yes | ❌ No | Sequentially trains weak models to improve prediction accuracy. |

✔ **Bagging is useful when variance is high (to stabilize predictions).**  
✔ **Boosting is useful when bias is high (to improve weak models).**

### 🔹 Reducing Variance - Bagging

What is bagging?

✔ Bagging (Bootstrap Aggregating) is an ensemble technique that reduces variance by training multiple independent models on different random subsets of the data.

✔ It ensures that each individual model learns slightly different aspects of the dataset, reducing overfitting.

✔ The final prediction is made by aggregating predictions:

* + Classification → Majority voting (the most common class among models is chosen).
  + Regression → Averaging predictions from all models.

How Bagging Works

1. Create multiple bootstrap samples → Randomly select subsets of training data with replacement (some samples may appear more than once).
2. Train multiple independent models on different subsets.
3. Each model makes a prediction, and the final result is obtained by majority voting (classification) or averaging (regression).

✔ Since models are trained on different subsets of data, their errors are less correlated, reducing overall variance.

Why does Bagging work?

* It smooths out fluctuations by training on diverse data samples.
* Works best with high-variance models (e.g., deep decision trees).
* Not very useful for high-bias models (like logistic regression), because adding weak models does not fix fundamental underfitting.

### 🔹 Reducing Bias - Boosting

What is Boosting?

✔ Boosting is an ensemble technique that reduces bias by sequentially training models, where each new model focuses on correcting errors made by previous models.

✔ Unlike Bagging, which trains models independently, Boosting builds models sequentially, making them more dependent on previous iterations.

✔ The final prediction is a weighted sum of all weak models.

How Boosting Works

1. **Train an initial weak model (e.g., a shallow decision tree).**
2. **Identify misclassified or high-error data points** from the previous model.
3. **Assign higher weights** to these "hard-to-classify" points so that the next model focuses on them.
4. **Train a new model on the updated dataset** with adjusted weights.
5. **Repeat for multiple iterations**, creating a sequence of progressively better models.
6. **Combine the predictions** from all weak models using a weighted sum.

✔ **Boosting turns weak learners into a strong ensemble by improving prediction step by step.**

### 🔹 Comparing Bagging and Boosting

✔ **Use Bagging when variance is high** (to stabilize predictions).  
✔ **Use Boosting when bias is high** (to improve weak models).  
✔ **Both methods improve generalization and performance**.

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Objective | How It Works | Best Use Case |
| Bagging | Reduce variance | Train multiple independent models on different random subsets of data | When variance is high (e.g., Random Forests) |
| Boosting | Reduce bias | Train models sequentially, each correcting the previous one | When bias is high (e.g., XGBoost, AdaBoost) |

## 📌 Algorithms strengths and weaknesses

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Strengths** | **Weaknesses** |
| **KNN** | ✔Simple, non-parametric, works well with small data. | ✖Computationally expensive for large datasets. |
| **Decision Tree** | ✔Fast inference, interpretable rules | ✖Prone to overfitting without pruning. |
| **SVM** | ✔Effective for high-dimensional spaces. | ✖Sensitive to noise, requires careful tuning. |
| **Logistic Reg.** | ✔Works well for linearly separable data. | ✖Struggles with complex decision boundaries. |